# SOUND4 BIGVOICE.CL Library [1.1.11]

Generated by Doxygen 1.9.8

**1 SOUND4 BIGVOICE.CL Library** 1

1.1 General library information . . . 1

1.2 General C usage . . . 1

**2 Generic C++ access to SOUND4 .CL librairies** 3

2.1 General usage . . . 3

2.2 Communicating with the instance . . . 3

2.3 Metadata . . . 4

2.4 Advanced tasks . . . 4

2.4.1 Custom storage . . . 4

2.4.2 Using a bus . . . 4

2.4.3 Convert helpers . . . 4

2.4.4 Adding a health point in the web server . . . 4

**3 JSON syntax** 5

3.1 Commands . . . 5

3.2 Answers . . . 6

3.3 Examples . . . 6

3.3.1 Get status . . . 6

3.3.2 Get all changes since last request . . . 6

3.3.3 Check license . . . 6

3.3.4 Set Processor name . . . 7

3.3.5 Get presetlist . . . 7

3.3.6 Load a preset . . . 7

3.3.7 Get information on preset on air . . . 7

3.3.8 Set input gain . . . 7

3.3.9 Set bypass . . . 7

3.3.10 Export a preset . . . 7

3.3.11 Import a preset . . . 8

**4 JSON syntax for internal setup** 9

4.1 Get process and modules information . . . 9

4.2 Authentication management . . . 9

4.2.1 Difference between setup and admin account . . . 9

4.2.2 Access encoding . . . 10

4.2.3 Change setup account . . . 10

4.2.3.1 Set the setup/admin account . . . 10

4.2.3.2 Set the setup/admin account with clear password . . . 10

4.2.3.3 Remove the setup/admin account . . . 10

4.2.4 Get user info . . . 10

4.2.5 Add or modify a user . . . 10

4.2.6 Removes a user . . . 11

4.2.7 List users . . . 11

4.2.8 Get/set unlogged access . . . 11
4.2.9 Get realm . . . 11
4.2.10 Get API version . . . 11
4.2.11 Get setup is admin . . . 11
4.2.12 Get/set setup account . . . 11
4.2.13 Get/set custom storage . . . 11
4.3 Get process' interface description . . . 12
4.3.1 Get XML decription . . . 12
4.3.2 Get Process Image . . . 12
4.3.3 Get Process Ico . . . 12

**5 Environment variables and parameters** **13**
5.1 Parameters defaulting from environment . . . 13
5.1.1 Webserver authentification setup : . . . 13
5.1.2 Webserver secure : . . . 14
5.1.3 Network access : . . . 14
5.2 Parameters . . . 14
5.2.1 Run mode: . . . 14
5.2.2 Web server options: . . . 15
5.3 Preset sharing . . . 15

**6 Deprecated List** **17**

**7 Topic Index** **19**
7.1 Topics . . . 19

**8 Namespace Index** **21**
8.1 Namespace List . . . 21

**9 Hierarchical Index** **23**
9.1 Class Hierarchy . . . 23

**10 Class Index** **25**
10.1 Class List . . . 25

**11 File Index** **27**
11.1 File List . . . 27

**12 Topic Documentation** **29**
12.1 Audio convert helpers . . . 29
12.1.1 Detailed Description . . . 29
12.1.2 Function Documentation . . . 29
12.1.2.1 bigvoice_AudioConvertFrom() . . . 29
12.1.2.2 bigvoice_AudioConvertTo() . . . 30
12.1.2.3 bigvoice_AudioMonoFromLiveStereo() . . . 30

12.1.2.4 bigvoice_AudioMonoToLiveStereo() . . . 30
12.1.2.5 bigvoice_MonoToStereo() . . . 32
12.1.2.6 bigvoice_MonoToStereo_Planar() . . . 32
12.1.2.7 bigvoice_StereoToMono() . . . 33
12.1.2.8 bigvoice_StereoToMono_Planar() . . . 33
12.2 Audio convert helpers (C++) . . . 33
12.2.1 Detailed Description . . . 34
12.2.2 Function Documentation . . . 34
12.2.2.1 AudioConvertFrom() . . . 34
12.2.2.2 AudioConvertTo() . . . 34
12.2.2.3 AudioMonoFromLiveStereo() . . . 35
12.2.2.4 AudioMonoToLiveStereo() . . . 35
12.2.2.5 MonoToStereo() . . . 35
12.2.2.6 MonoToStereo_Planar() . . . 36
12.2.2.7 StereoToMono() . . . 36
12.2.2.8 StereoToMono_Planar() . . . 37
12.3 Audio managing (any audio) . . . 37
12.3.1 Detailed Description . . . 38
12.3.2 Enumeration Type Documentation . . . 38
12.3.2.1 bigvoice_SampleFormat . . . 38
12.3.3 Function Documentation . . . 39
12.3.3.1 bigvoice_AddAudio() . . . 39
12.3.3.2 bigvoice_AddPadAudio() . . . 39
12.3.3.3 bigvoice_GetAudio() . . . 39
12.3.3.4 bigvoice_GetBytesFromFormat() . . . 40
12.3.3.5 bigvoice_GetFormatFromName() . . . 40
12.3.3.6 bigvoice_GetFormatName() . . . 40
12.3.3.7 bigvoice_GetMaxPacketFrame() . . . 41
12.3.3.8 bigvoice_GetOutputCount() . . . 41
12.4 Auxiliary outputs . . . 41
12.4.1 Detailed Description . . . 42
12.4.2 Function Documentation . . . 42
12.4.2.1 bigvoice_ActivateAuxiliaryAudioID() . . . 42
12.4.2.2 bigvoice_FreeAuxiliaryAudioInfos() . . . 42
12.4.2.3 bigvoice_GetAuxiliaryAudioEstimatedDelay() . . . 43
12.4.2.4 bigvoice_GetAuxiliaryAudioOutputInfos() . . . 43
12.5 Client's JSon config interfacing . . . 43
12.5.1 Detailed Description . . . 44
12.5.2 Function Documentation . . . 44
12.5.2.1 bigvoice_DeleteClient() . . . 44
12.5.2.2 bigvoice_FreeJsonAnswer() . . . 44
12.5.2.3 bigvoice_NewClient() . . . 44

12.5.2.4 bigvoice_ProcessJson() . . . 45
12.5.2.5 bigvoice_SaveState() . . . 45
12.6 Custom preset managagement . . . 46
12.6.1 Detailed Description . . . 47
12.6.2 Typedef Documentation . . . 47
12.6.2.1 bigvoice_storage_exists . . . 47
12.6.2.2 bigvoice_storage_getall . . . 47
12.6.2.3 bigvoice_storage_getall_free . . . 48
12.6.2.4 bigvoice_storage_reader . . . 48
12.6.2.5 bigvoice_storage_reader_free . . . 48
12.6.2.6 bigvoice_storage_remove . . . 49
12.6.2.7 bigvoice_storage_rename . . . 49
12.6.2.8 bigvoice_storage_writer . . . 49
12.6.2.9 fs_char . . . 50
12.6.3 Enumeration Type Documentation . . . 50
12.6.3.1 bigvoice_PresetChange_Kind . . . 50
12.6.4 Function Documentation . . . 50
12.6.4.1 bigvoice_PresetManager_InformChange() . . . 50
12.6.4.2 bigvoice_SetPresetManager() . . . 51
12.7 Direct processing . . . 51
12.7.1 Detailed Description . . . 52
12.7.2 Function Documentation . . . 52
12.7.2.1 bigvoice_GetBufferIn() . . . 52
12.7.2.2 bigvoice_GetBufferOut() . . . 52
12.7.2.3 bigvoice_GetEstimatedDelay() . . . 53
12.7.2.4 bigvoice_ProcessAudio() . . . 53
12.7.2.5 bigvoice_ProcessAudio_Planar() . . . 54
12.8 Examples . . . 54
12.9 Global setup . . . 55
12.9.1 Detailed Description . . . 55
12.9.2 Macro Definition Documentation . . . 56
12.9.2.1 BIGVOICE_AUDIO_INPUT_COUNT . . . 56
12.9.2.2 BIGVOICE_AUDIO_OUTPUT_COUNT . . . 56
12.9.2.3 BIGVOICE_AUDIOFRAME_COUNT . . . 56
12.9.2.4 BIGVOICE_CHANNEL_COUNT . . . 56
12.9.2.5 BIGVOICE_SAMPLE_RATE . . . 57
12.9.3 Typedef Documentation . . . 57
12.9.3.1 bigvoice_loggerfn . . . 57
12.9.4 Enumeration Type Documentation . . . 57
12.9.4.1 bigvoice_LogSeverity . . . 57
12.9.5 Function Documentation . . . 58
12.9.5.1 bigvoice_GetAudioInputCount() . . . 58

12.9.5.2 bigvoice_GetAudioOutputCount() . . . 58
12.9.5.3 bigvoice_GetChannelCount() . . . 58
12.9.5.4 bigvoice_GetChunkSizeInFrames() . . . 59
12.9.5.5 bigvoice_GetPossibleChunkSizeInFrames() . . . 59
12.9.5.6 bigvoice_GetSampleRate() . . . 59
12.9.5.7 bigvoice_GetVersion() . . . 60
12.9.5.8 bigvoice_SetLoggerCallback() . . . 60
12.9.5.9 bigvoice_SetLogSeverity() . . . 60
12.9.5.10 SOUND4_GetProcessInfo() . . . 60
12.10 Instance tracing (Windows only) . . . 61
12.10.1 Detailed Description . . . 61
12.10.2 Function Documentation . . . 61
12.10.2.1 bigvoice_SetInstanceTracing() . . . 61
12.10.2.2 bigvoice_SetInstanceTracingProcessActivity() . . . 61
12.11 Instance's bus . . . 62
12.11.1 Detailed Description . . . 62
12.11.2 Function Documentation . . . 62
12.11.2.1 bigvoice_FreeBus() . . . 62
12.11.2.2 bigvoice_NewBus() . . . 62
12.11.2.3 bigvoice_SetInstanceBus() . . . 63
12.12 Metadata Management . . . 63
12.12.1 Detailed Description . . . 63
12.12.2 Function Documentation . . . 63
12.12.2.1 bigvoice_FreeMetadataInfos() . . . 63
12.12.2.2 bigvoice_GetMetadataInfos() . . . 64
12.12.2.3 bigvoice_SetMetadata() . . . 64
12.12.2.4 bigvoice_SetMetadataMulti() . . . 64
12.13 Process creation/destruction . . . 65
12.13.1 Detailed Description . . . 65
12.13.2 Function Documentation . . . 65
12.13.2.1 bigvoice_ExitProcess() . . . 65
12.13.2.2 bigvoice_FreeParameters() . . . 66
12.13.2.3 bigvoice_FreeParameterValue() . . . 66
12.13.2.4 bigvoice_GetParameter() . . . 66
12.13.2.5 bigvoice_GetProcessChunkFrames() . . . 66
12.13.2.6 bigvoice_InitProcess() . . . 67
12.13.2.7 bigvoice_InitProcess2() . . . 67
12.13.2.8 bigvoice_InitProcess3() . . . 68
12.13.2.9 bigvoice_NewParameters() . . . 68
12.13.2.10 bigvoice_SetParameter() . . . 69
12.13.2.11 bigvoice_TerminateProcess() . . . 69
12.14 Sample using C interface . . . 69

12.14.1 Sample to create the processing instance . . . 69
12.14.2 Sample processing loop . . . 70
12.14.3 Sample to stop the processing instance . . . 71
12.15 Sample using C++ interface wrapper . . . 71
12.15.1 Setup and sanity checks . . . 71
12.15.2 Sample to create the processing instance . . . 71
12.15.3 Communicating with the instance . . . 71
12.15.4 Starting a web-server . . . 72
12.15.5 Sample processing loop . . . 72
12.16 Sample using C++ interface wrapper with dynamic library loading . . . 72
12.16.1 Dynamic loading . . . 72
12.16.2 Setup and sanity checks . . . 72
12.16.3 Sample to create the processing instance . . . 72
12.16.4 Communicating with the instance . . . 72
12.16.5 Starting a web-server . . . 73
12.16.6 Sample processing loop . . . 73
12.16.7 Custom preset manager . . . 73
12.16.8 Custom preset manager loading . . . 74
12.17 Sample using Generic C++ interface (multi-process compatible) . . . 74
12.17.1 Loading the library . . . 74
12.17.2 Setup and configure . . . 74
12.17.3 Create the processing instance . . . 75
12.17.4 Processing loop . . . 75
12.17.5 Starting a web-server . . . 75
12.17.6 Communicating with the instance . . . 75
12.17.7 Custom preset manager example . . . 75
12.17.8 Custom preset manager loading . . . 76
12.18 Update Management . . . 76
12.18.1 Detailed Description . . . 77
12.18.2 Function Documentation . . . 77
12.18.2.1 bigvoice_StartUpdateThread() . . . 77
12.18.2.2 bigvoice_StopUpdateThread() . . . 77
12.18.2.3 bigvoice_WaitUpdateThreadReady() . . . 77
12.19 Web server . . . 78
12.19.1 Detailed Description . . . 78
12.19.2 Macro Definition Documentation . . . 78
12.19.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 78
12.19.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 79
12.19.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 79
12.19.3 Function Documentation . . . 79
12.19.3.1 bigvoice_Webserver() . . . 79
12.19.3.2 bigvoice_Webserver_FreeString() . . . 80

12.19.3.3 bigvoice_Webserver_GetAppHealth() . . . 80
12.19.3.4 bigvoice_Webserver_SetAppHealth() . . . 80
12.19.3.5 bigvoice_Webserver_Status() . . . 81
12.19.3.6 bigvoice_Webserver_Stop() . . . 81
12.19.3.7 bigvoice_Webserver_tcp() . . . 81
12.19.3.8 bigvoice_Webserver_tcp2() . . . 82

**13 Namespace Documentation** **83**

13.1 sound4 Namespace Reference . . . 83
13.1.1 Detailed Description . . . 84
13.1.2 Typedef Documentation . . . 84
13.1.2.1 log_cb_t . . . 84
13.1.3 Enumeration Type Documentation . . . 84
13.1.3.1 LogSeverity . . . 84
13.1.3.2 PresetChange_Kind . . . 85
13.1.3.3 SampleFormat . . . 85
13.1.4 Function Documentation . . . 85
13.1.4.1 _log_cb_c() . . . 85
13.1.5 Variable Documentation . . . 86
13.1.5.1 _log_cb . . . 86
13.2 sound4::bigvoice Namespace Reference . . . 86
13.2.1 Detailed Description . . . 87
13.2.2 Typedef Documentation . . . 87
13.2.2.1 log_cb_t . . . 87
13.2.2.2 LogSeverity . . . 87
13.2.2.3 PresetChange_Kind . . . 87
13.2.2.4 SampleFormat . . . 87
13.2.3 Function Documentation . . . 88
13.2.3.1 _log_cb_c() . . . 88
13.2.3.2 GetAudioInputCount() . . . 88
13.2.3.3 GetAudioOutputCount() . . . 88
13.2.3.4 GetBytesFromFormat() . . . 88
13.2.3.5 GetChannelCount() . . . 89
13.2.3.6 GetChunkSizeInFrames() . . . 89
13.2.3.7 GetFormatFromName() . . . 89
13.2.3.8 GetFormatName() . . . 90
13.2.3.9 GetPossibleChunkSizeInFrames() . . . 90
13.2.3.10 GetSampleRate() . . . 91
13.2.3.11 GetVersion() . . . 91
13.2.3.12 SanityCheck() . . . 91
13.2.3.13 SetLoggerCallback() . . . 91
13.2.3.14 SetLogSeverity() . . . 92

13.2.4 Variable Documentation . . . 92
13.2.4.1 _log_cb . . . 92
13.2.4.2 ChunkFrames . . . 92
13.2.4.3 InputChannels . . . 92
13.2.4.4 InputSampleSize . . . 92
13.2.4.5 OutputChannels . . . 93
13.2.4.6 OutputSampleSize . . . 93
13.2.4.7 process_name . . . 93
13.2.4.8 process_shortname . . . 93
13.2.4.9 SampleRate . . . 93
13.3 sound4::bigvoice::dyn Namespace Reference . . . 94
13.3.1 Detailed Description . . . 95
13.3.2 Typedef Documentation . . . 95
13.3.2.1 log_cb_t . . . 95
13.3.2.2 LogSeverity . . . 95
13.3.2.3 PresetChange_Kind . . . 95
13.3.2.4 SampleFormat . . . 95
13.3.3 Function Documentation . . . 95
13.3.3.1 _log_cb_c() . . . 95
13.3.3.2 GetAudioInputCount() . . . 95
13.3.3.3 GetAudioOutputCount() . . . 96
13.3.3.4 GetBytesFromFormat() . . . 96
13.3.3.5 GetChannelCount() . . . 97
13.3.3.6 GetChunkSizeInFrames() . . . 97
13.3.3.7 GetFormatFromName() . . . 97
13.3.3.8 GetFormatName() . . . 98
13.3.3.9 GetPossibleChunkSizeInFrames() . . . 98
13.3.3.10 GetSampleRate() . . . 99
13.3.3.11 GetVersion() . . . 99
13.3.3.12 SanityCheck() . . . 99
13.3.3.13 SetLoggerCallback() . . . 101
13.3.3.14 SetLogSeverity() . . . 101
13.3.4 Variable Documentation . . . 101
13.3.4.1 _log_cb . . . 101
13.3.4.2 ChunkFrames . . . 101
13.3.4.3 InputChannels . . . 102
13.3.4.4 InputSampleSize . . . 102
13.3.4.5 OutputChannels . . . 102
13.3.4.6 OutputSampleSize . . . 102
13.3.4.7 process_name . . . 102
13.3.4.8 process_shortname . . . 103
13.3.4.9 SampleRate . . . 103

13.4 sound4::bigvoice::dyn::helper Namespace Reference . . . 103
13.4.1 Detailed Description . . . 104
13.4.2 Function Documentation . . . 104
13.4.2.1 AudioConvertFrom() . . . 104
13.4.2.2 AudioConvertTo() . . . 104
13.4.2.3 AudioMonoFromLiveStereo() . . . 105
13.4.2.4 AudioMonoToLiveStereo() . . . 105
13.4.2.5 MonoToStereo() . . . 105
13.4.2.6 MonoToStereo_Planar() . . . 106
13.4.2.7 StereoToMono() . . . 106
13.4.2.8 StereoToMono_Planar() . . . 107
13.4.2.9 UTF8ToWString() . . . 107
13.4.2.10 WStringToUTF8() . . . 108
13.5 sound4::bigvoice::helper Namespace Reference . . . 108
13.5.1 Detailed Description . . . 108
13.5.2 Function Documentation . . . 109
13.5.2.1 AudioConvertFrom() . . . 109
13.5.2.2 AudioConvertTo() . . . 109
13.5.2.3 AudioMonoFromLiveStereo() . . . 110
13.5.2.4 AudioMonoToLiveStereo() . . . 110
13.5.2.5 MonoToStereo() . . . 110
13.5.2.6 MonoToStereo_Planar() . . . 111
13.5.2.7 StereoToMono() . . . 111
13.5.2.8 StereoToMono_Planar() . . . 112
13.5.2.9 UTF8ToWString() . . . 112
13.5.2.10 WStringToUTF8() . . . 112
13.6 sound4::helper Namespace Reference . . . 113
13.6.1 Detailed Description . . . 113
13.6.2 Function Documentation . . . 113
13.6.2.1 UTF8ToWString() . . . 113
13.6.2.2 WStringToUTF8() . . . 113

**14 Class Documentation** **115**

14.1 sound4::AuxiliaryOutputInfos Struct Reference . . . 115
14.1.1 Detailed Description . . . 115
14.1.2 Member Data Documentation . . . 115
14.1.2.1 buffer . . . 115
14.1.2.2 channel_count . . . 116
14.1.2.3 description . . . 116
14.1.2.4 id . . . 116
14.1.2.5 sample_rate . . . 116
14.2 bigvoice_CBus Struct Reference . . . 116

14.2.1 Detailed Description . . . 116
14.3 bigvoice_CClientInstance Struct Reference . . . 116
14.3.1 Detailed Description . . . 117
14.4 bigvoice_CInstance Struct Reference . . . 117
14.4.1 Detailed Description . . . 117
14.5 bigvoice_CParameters Struct Reference . . . 117
14.5.1 Detailed Description . . . 117
14.6 sound4::bigvoice::CBus Class Reference . . . 117
14.6.1 Detailed Description . . . 118
14.6.2 Constructor & Destructor Documentation . . . 118
14.6.2.1 CBus() . . . 118
14.6.2.2 ~CBus() . . . 118
14.6.3 Member Function Documentation . . . 118
14.6.3.1 Get() . . . 118
14.7 sound4::bigvoice::dyn::CBus Class Reference . . . 118
14.7.1 Detailed Description . . . 119
14.7.2 Constructor & Destructor Documentation . . . 119
14.7.2.1 CBus() . . . 119
14.7.2.2 ~CBus() . . . 119
14.7.3 Member Function Documentation . . . 119
14.7.3.1 Get() . . . 119
14.8 sound4::CBus Class Reference . . . 119
14.8.1 Detailed Description . . . 120
14.8.2 Constructor & Destructor Documentation . . . 120
14.8.2.1 CBus() . . . 120
14.8.2.2 ~CBus() . . . 120
14.8.3 Member Function Documentation . . . 120
14.8.3.1 Get() . . . 120
14.8.4 Friends And Related Symbol Documentation . . . 120
14.8.4.1 CInstance . . . 120
14.9 sound4::bigvoice::CInstance::CClient Class Reference . . . 121
14.9.1 Detailed Description . . . 121
14.9.2 Constructor & Destructor Documentation . . . 121
14.9.2.1 CClient() [1/2] . . . 121
14.9.2.2 CClient() [2/2] . . . 121
14.9.2.3 ~CClient() . . . 121
14.9.3 Member Function Documentation . . . 121
14.9.3.1 operator=() . . . 121
14.9.3.2 ProcessJson() . . . 121
14.10 sound4::bigvoice::dyn::CInstance::CClient Class Reference . . . 122
14.10.1 Detailed Description . . . 122
14.10.2 Constructor & Destructor Documentation . . . 122

14.10.2.1 CClient() [1/2] . . . 122
14.10.2.2 CClient() [2/2] . . . 123
14.10.2.3 ~CClient() . . . 123
14.10.3 Member Function Documentation . . . 123
14.10.3.1 operator=() . . . 123
14.10.3.2 ProcessJson() . . . 123
14.11 sound4::CInstance::CClient Class Reference . . . 123
14.11.1 Detailed Description . . . 124
14.11.2 Constructor & Destructor Documentation . . . 124
14.11.2.1 CClient() [1/2] . . . 124
14.11.2.2 CClient() [2/2] . . . 124
14.11.2.3 ~CClient() . . . 124
14.11.3 Member Function Documentation . . . 124
14.11.3.1 operator=() . . . 124
14.11.3.2 ProcessJson() . . . 124
14.12 sound4::bigvoice::dyn::CDynLib Class Reference . . . 125
14.12.1 Detailed Description . . . 126
14.12.2 Constructor & Destructor Documentation . . . 127
14.12.2.1 CDynLib() . . . 127
14.12.3 Member Function Documentation . . . 127
14.12.3.1 IsOk() . . . 127
14.12.3.2 Load() . . . 127
14.12.4 Member Data Documentation . . . 127
14.12.4.1 AddAudio . . . 127
14.12.4.2 AddPadAudio . . . 128
14.12.4.3 AudioConvertFrom . . . 128
14.12.4.4 AudioConvertTo . . . 128
14.12.4.5 AudioMonoFromLiveStereo . . . 128
14.12.4.6 AudioMonoToLiveStereo . . . 128
14.12.4.7 DeleteClient . . . 128
14.12.4.8 ExitProcess . . . 128
14.12.4.9 FreeBus . . . 129
14.12.4.10 FreeJsonAnswer . . . 129
14.12.4.11 FreeMetadataInfos . . . 129
14.12.4.12 FreeParameters . . . 129
14.12.4.13 FreeParameterValue . . . 129
14.12.4.14 GetAudio . . . 129
14.12.4.15 GetAudioInputCount . . . 129
14.12.4.16 GetAudioOutputCount . . . 130
14.12.4.17 GetBufferIn . . . 130
14.12.4.18 GetBufferOut . . . 130
14.12.4.19 GetBytesFromFormat . . . 130

14.12.4.20 GetChannelCount . . . 130
14.12.4.21 GetChunkSizeInFrames . . . 130
14.12.4.22 GetEstimatedDelay . . . 130
14.12.4.23 GetFormatFromName . . . 131
14.12.4.24 GetFormatName . . . 131
14.12.4.25 GetMaxPacketFrame . . . 131
14.12.4.26 GetMetadataInfos . . . 131
14.12.4.27 GetOutputCount . . . 131
14.12.4.28 GetParameter . . . 131
14.12.4.29 GetPossibleChunkSizeInFrames . . . 131
14.12.4.30 GetProcessChunkFrames . . . 132
14.12.4.31 GetSampleRate . . . 132
14.12.4.32 GetVersion . . . 132
14.12.4.33 InitProcess . . . 132
14.12.4.34 InitProcess2 . . . 132
14.12.4.35 InitProcess3 . . . 132
14.12.4.36 MonoToStereo . . . 132
14.12.4.37 MonoToStereo_Planar . . . 133
14.12.4.38 NewBus . . . 133
14.12.4.39 NewClient . . . 133
14.12.4.40 NewParameters . . . 133
14.12.4.41 PresetManager_InformChange . . . 133
14.12.4.42 ProcessAudio . . . 133
14.12.4.43 ProcessAudio_Planar . . . 133
14.12.4.44 ProcessJson . . . 134
14.12.4.45 SaveState . . . 134
14.12.4.46 SetInstanceBus . . . 134
14.12.4.47 SetInstanceTracing . . . 134
14.12.4.48 SetInstanceTracingProcessActivity . . . 134
14.12.4.49 SetLoggerCallback . . . 134
14.12.4.50 SetLogSeverity . . . 134
14.12.4.51 SetMetadata . . . 135
14.12.4.52 SetMetadataMulti . . . 135
14.12.4.53 SetParameter . . . 135
14.12.4.54 SetPresetManager . . . 135
14.12.4.55 StartUpdateThread . . . 135
14.12.4.56 StereoToMono . . . 135
14.12.4.57 StereoToMono_Planar . . . 135
14.12.4.58 StopUpdateThread . . . 136
14.12.4.59 TerminateProcess . . . 136
14.12.4.60 WaitUpdateThreadReady . . . 136
14.12.4.61 Webserver . . . 136

14.12.4.62 Webserver_FreeString . . . 136
14.12.4.63 Webserver_GetAppHealth . . . 136
14.12.4.64 Webserver_SetAppHealth . . . 136
14.12.4.65 Webserver_Status . . . 137
14.12.4.66 Webserver_Stop . . . 137
14.12.4.67 Webserver_tcp . . . 137
14.12.4.68 Webserver_tcp2 . . . 137
14.13 sound4::bigvoice::dyn::helper::CDynLoader Class Reference . . . 137
14.13.1 Detailed Description . . . 138
14.13.2 Constructor & Destructor Documentation . . . 138
14.13.2.1 CDynLoader() [1/3] . . . 138
14.13.2.2 CDynLoader() [2/3] . . . 138
14.13.2.3 CDynLoader() [3/3] . . . 138
14.13.2.4 ~CDynLoader() . . . 138
14.13.3 Member Function Documentation . . . 138
14.13.3.1 Close() . . . 138
14.13.3.2 GetSymbol() . . . 138
14.13.3.3 GetSymbol_safe() . . . 139
14.13.3.4 GetThisLibraryPath() . . . 139
14.13.3.5 IsOk() . . . 139
14.13.3.6 Load() . . . 139
14.13.3.7 operator bool() . . . 139
14.13.3.8 operator=() . . . 139
14.14 sound4::helper::CDynLoader Class Reference . . . 140
14.14.1 Detailed Description . . . 140
14.14.2 Constructor & Destructor Documentation . . . 140
14.14.2.1 CDynLoader() [1/3] . . . 140
14.14.2.2 CDynLoader() [2/3] . . . 140
14.14.2.3 CDynLoader() [3/3] . . . 140
14.14.2.4 ~CDynLoader() . . . 141
14.14.3 Member Function Documentation . . . 141
14.14.3.1 Close() . . . 141
14.14.3.2 GetSymbol() . . . 141
14.14.3.3 GetSymbol_safe() . . . 141
14.14.3.4 GetThisLibraryPath() . . . 141
14.14.3.5 IsOk() . . . 141
14.14.3.6 Load() . . . 142
14.14.3.7 operator bool() . . . 142
14.14.3.8 operator=() . . . 142
14.15 sound4::bigvoice::CInstance Class Reference . . . 142
14.15.1 Detailed Description . . . 143
14.15.2 Constructor & Destructor Documentation . . . 144

14.15.2.1 CInstance() [1/2] . . . 144
14.15.2.2 CInstance() [2/2] . . . 144
14.15.2.3 ~CInstance() . . . 144
14.15.3 Member Function Documentation . . . 144
14.15.3.1 ActivateAuxiliaryAudioID() . . . 144
14.15.3.2 AddAudio() . . . 144
14.15.3.3 Create() . . . 145
14.15.3.4 GetAudio() . . . 145
14.15.3.5 GetAuxiliaryAudioOutputInfos() . . . 146
14.15.3.6 GetBufferIn() . . . 146
14.15.3.7 GetBufferOut() . . . 146
14.15.3.8 GetChunkFrames() . . . 147
14.15.3.9 GetEstimatedDelay() . . . 147
14.15.3.10 GetMetadataInfos() . . . 147
14.15.3.11 GetParam() . . . 147
14.15.3.12 GetWebServerAppHealth() . . . 148
14.15.3.13 IsOk() . . . 148
14.15.3.14 NewClient() . . . 148
14.15.3.15 OnUpdateThreadStart() . . . 148
14.15.3.16 OnUpdateThreadStop() . . . 149
14.15.3.17 operator=() . . . 149
14.15.3.18 PresetManager_InformChange() . . . 149
14.15.3.19 ProcessAnyAudio() . . . 149
14.15.3.20 ProcessAudio() . . . 150
14.15.3.21 ProcessAudio_Planar() . . . 150
14.15.3.22 SaveState() . . . 150
14.15.3.23 SetBus() . . . 151
14.15.3.24 SetInstanceTracing() . . . 151
14.15.3.25 SetInstanceTracingProcessActivity() . . . 151
14.15.3.26 SetMetadata() . . . 152
14.15.3.27 SetMetadataMulti() . . . 152
14.15.3.28 SetParam() . . . 152
14.15.3.29 SetPresetManager() . . . 153
14.15.3.30 SetWebServerAppHealth() . . . 153
14.15.3.31 StartWebServer() . . . 153
14.15.3.32 Stop() . . . 154
14.15.3.33 StopWebServer() . . . 154
14.15.3.34 Terminate() . . . 154
14.16 sound4::bigvoice::dyn::CInstance Class Reference . . . 155
14.16.1 Detailed Description . . . 156
14.16.2 Constructor & Destructor Documentation . . . 156
14.16.2.1 CInstance() [1/2] . . . 156

14.16.2.2 CInstance() [2/2] . . . 156
14.16.2.3 ~CInstance() . . . 156
14.16.3 Member Function Documentation . . . 156
14.16.3.1 AddAudio() . . . 156
14.16.3.2 Create() . . . 157
14.16.3.3 GetAudio() . . . 157
14.16.3.4 GetBufferIn() . . . 158
14.16.3.5 GetBufferOut() . . . 158
14.16.3.6 GetChunkFrames() . . . 158
14.16.3.7 GetEstimatedDelay() . . . 159
14.16.3.8 GetMetadataInfos() . . . 159
14.16.3.9 GetParam() . . . 159
14.16.3.10 GetWebServerAppHealth() . . . 160
14.16.3.11 IsOk() . . . 160
14.16.3.12 NewClient() . . . 160
14.16.3.13 OnUpdateThreadStart() . . . 160
14.16.3.14 OnUpdateThreadStop() . . . 160
14.16.3.15 operator=() . . . 160
14.16.3.16 PresetManager_InformChange() . . . 160
14.16.3.17 ProcessAnyAudio() . . . 161
14.16.3.18 ProcessAudio() . . . 161
14.16.3.19 ProcessAudio_Planar() . . . 162
14.16.3.20 SaveState() . . . 163
14.16.3.21 SetBus() . . . 163
14.16.3.22 SetInstanceTracing() . . . 163
14.16.3.23 SetInstanceTracingProcessActivity() . . . 164
14.16.3.24 SetMetadata() . . . 164
14.16.3.25 SetMetadataMulti() . . . 164
14.16.3.26 SetParam() . . . 165
14.16.3.27 SetPresetManager() . . . 165
14.16.3.28 SetWebServerAppHealth() . . . 165
14.16.3.29 StartWebServer() . . . 166
14.16.3.30 Stop() . . . 166
14.16.3.31 StopWebServer() . . . 166
14.16.3.32 Terminate() . . . 167
14.17 sound4::CInstance Class Reference . . . 167
14.17.1 Detailed Description . . . 168
14.17.2 Constructor & Destructor Documentation . . . 169
14.17.2.1 CInstance() [1/3] . . . 169
14.17.2.2 CInstance() [2/3] . . . 169
14.17.2.3 CInstance() [3/3] . . . 169
14.17.2.4 ~CInstance() . . . 169

14.17.3 Member Function Documentation . . . 170
14.17.3.1 AddAudio() . . . 170
14.17.3.2 Create() . . . 170
14.17.3.3 GetAudio() . . . 171
14.17.3.4 GetBufferIn() . . . 171
14.17.3.5 GetBufferOut() . . . 171
14.17.3.6 GetChunkFrames() . . . 172
14.17.3.7 GetEstimatedDelay() . . . 172
14.17.3.8 GetMetadataInfos() . . . 172
14.17.3.9 GetParam() . . . 172
14.17.3.10 GetWebServerAppHealth() . . . 173
14.17.3.11 IsOk() . . . 173
14.17.3.12 NewClient() . . . 173
14.17.3.13 OnUpdateThreadStart() . . . 173
14.17.3.14 OnUpdateThreadStop() . . . 174
14.17.3.15 operator=() . . . 174
14.17.3.16 PresetManager_InformChange() . . . 174
14.17.3.17 ProcessAnyAudio() . . . 174
14.17.3.18 ProcessAudio() . . . 175
14.17.3.19 ProcessAudio_Planar() . . . 175
14.17.3.20 SaveState() . . . 175
14.17.3.21 SetBus() . . . 176
14.17.3.22 SetInstanceTracing() . . . 176
14.17.3.23 SetInstanceTracingProcessActivity() . . . 176
14.17.3.24 SetMetadata() . . . 177
14.17.3.25 SetMetadataMulti() . . . 177
14.17.3.26 SetParam() . . . 177
14.17.3.27 SetPresetManager() . . . 178
14.17.3.28 SetWebServerAppHealth() . . . 178
14.17.3.29 StartWebServer() . . . 178
14.17.3.30 Stop() . . . 179
14.17.3.31 StopWebServer() . . . 179
14.17.3.32 Terminate() . . . 179
14.18 sound4::bigvoice::CPresetLoader Class Reference . . . 180
14.18.1 Detailed Description . . . 180
14.18.2 Constructor & Destructor Documentation . . . 180
14.18.2.1 CPresetLoader() . . . 180
14.18.2.2 ~CPresetLoader() . . . 180
14.18.3 Member Function Documentation . . . 181
14.18.3.1 Exists() . . . 181
14.18.3.2 GetAll() . . . 181
14.18.3.3 IsReadOnly() . . . 181

14.18.3.4 Read() . . . 181
14.18.3.5 Remove() . . . 181
14.18.3.6 Rename() . . . 181
14.18.3.7 Write() . . . 181
14.19 sound4::bigvoice::dyn::CPresetLoader Class Reference . . . 182
14.19.1 Detailed Description . . . 182
14.19.2 Constructor & Destructor Documentation . . . 182
14.19.2.1 CPresetLoader() . . . 182
14.19.2.2 ~CPresetLoader() . . . 182
14.19.3 Member Function Documentation . . . 182
14.19.3.1 Exists() . . . 182
14.19.3.2 GetAll() . . . 183
14.19.3.3 IsReadOnly() . . . 183
14.19.3.4 Read() . . . 183
14.19.3.5 Remove() . . . 183
14.19.3.6 Rename() . . . 183
14.19.3.7 Write() . . . 183
14.20 sound4::CPresetLoader Class Reference . . . 184
14.20.1 Detailed Description . . . 184
14.20.2 Constructor & Destructor Documentation . . . 184
14.20.2.1 CPresetLoader() . . . 184
14.20.2.2 ~CPresetLoader() . . . 184
14.20.3 Member Function Documentation . . . 184
14.20.3.1 Exists() . . . 184
14.20.3.2 GetAll() . . . 185
14.20.3.3 IsReadOnly() . . . 185
14.20.3.4 Read() . . . 185
14.20.3.5 Remove() . . . 185
14.20.3.6 Rename() . . . 185
14.20.3.7 Write() . . . 185
14.21 sound4::CProcessor Class Reference . . . 186
14.21.1 Detailed Description . . . 186
14.21.2 Constructor & Destructor Documentation . . . 187
14.21.2.1 CProcessor() . . . 187
14.21.3 Member Function Documentation . . . 187
14.21.3.1 GetAudioInputCount() . . . 187
14.21.3.2 GetAudioOutputCount() . . . 187
14.21.3.3 GetBytesFromFormat() . . . 187
14.21.3.4 GetChannelCount() . . . 188
14.21.3.5 GetChunkSizeInFrames() . . . 188
14.21.3.6 GetFormatFromName() . . . 188
14.21.3.7 GetFormatName() . . . 189

14.21.3.8 GetPossibleChunkSizeInFrames() . . . 189
14.21.3.9 GetPrefixSymbol() . . . 190
14.21.3.10 GetSampleRate() . . . 190
14.21.3.11 GetVersion() . . . 190
14.21.3.12 IsOk() . . . 190
14.21.3.13 Load() . . . 190
14.21.3.14 SetLoggerCallback() . . . 191
14.21.3.15 SetLogSeverity() . . . 191
14.21.4 Friends And Related Symbol Documentation . . . 191
14.21.4.1 CBus . . . 191
14.21.4.2 CInstance . . . 191
14.22 sound4::bigvoice::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 192
14.22.1 Detailed Description . . . 192
14.22.2 Constructor & Destructor Documentation . . . 192
14.22.2.1 DynFuncHolder() [1/2] . . . 192
14.22.2.2 DynFuncHolder() [2/2] . . . 192
14.22.3 Member Function Documentation . . . 192
14.22.3.1 IsOk() . . . 192
14.22.3.2 operator bool() . . . 193
14.22.3.3 operator T*() . . . 193
14.23 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 193
14.23.1 Detailed Description . . . 193
14.23.2 Constructor & Destructor Documentation . . . 194
14.23.2.1 DynFuncHolder() [1/2] . . . 194
14.23.2.2 DynFuncHolder() [2/2] . . . 194
14.23.3 Member Function Documentation . . . 194
14.23.3.1 IsOk() . . . 194
14.23.3.2 operator bool() . . . 194
14.23.3.3 operator T*() . . . 194
14.24 MyPresetManager Class Reference . . . 195
14.24.1 Detailed Description . . . 195
14.24.2 Constructor & Destructor Documentation . . . 196
14.24.2.1 MyPresetManager() [1/2] . . . 196
14.24.2.2 ~MyPresetManager() [1/2] . . . 196
14.24.2.3 MyPresetManager() [2/2] . . . 196
14.24.2.4 ~MyPresetManager() [2/2] . . . 196
14.24.3 Member Function Documentation . . . 196
14.24.3.1 Exists() [1/2] . . . 196
14.24.3.2 Exists() [2/2] . . . 196
14.24.3.3 GetAll() [1/2] . . . 196
14.24.3.4 GetAll() [2/2] . . . 197
14.24.3.5 IsReadOnly() [1/2] . . . 197

14.24.3.6 IsReadOnly() [2/2] . . . 197
14.24.3.7 Read() [1/2] . . . 197
14.24.3.8 Read() [2/2] . . . 197
14.24.3.9 Remove() [1/2] . . . 197
14.24.3.10 Remove() [2/2] . . . 198
14.24.3.11 Rename() [1/2] . . . 198
14.24.3.12 Rename() [2/2] . . . 198
14.24.3.13 SetupPath() [1/2] . . . 198
14.24.3.14 SetupPath() [2/2] . . . 198
14.24.3.15 Write() [1/2] . . . 198
14.24.3.16 Write() [2/2] . . . 199
14.25 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference . . . 199
14.25.1 Detailed Description . . . 199
14.26 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference . . . 199
14.26.1 Detailed Description . . . 200
14.27 sound4::helper::SampleFormat< T > Struct Template Reference . . . 201
14.27.1 Detailed Description . . . 201
14.28 SOUND4_AuxiliaryOutputInfos Struct Reference . . . 201
14.28.1 Detailed Description . . . 202
14.28.2 Member Data Documentation . . . 202
14.28.2.1 buffer . . . 202
14.28.2.2 channel_count . . . 202
14.28.2.3 description . . . 202
14.28.2.4 id . . . 202
14.28.2.5 sample_rate . . . 202
14.29 SOUND4_ProcessInfo Struct Reference . . . 202
14.29.1 Detailed Description . . . 203
14.29.2 Member Data Documentation . . . 203
14.29.2.1 name . . . 203
14.29.2.2 prefix . . . 203
14.29.2.3 shortname . . . 203
14.29.2.4 struct_size . . . 203
14.29.2.5 version . . . 203

**15 File Documentation** **205**

15.1 doc_examples.h File Reference . . . 205
15.2 doc_examples.h . . . 205
15.3 doc_gencpp.h File Reference . . . 205
15.4 doc_gencpp.h . . . 205
15.5 doc_json.h File Reference . . . 205
15.6 doc_json.h . . . 205
15.7 doc_main.h File Reference . . . 206

15.8 doc_main.h . . . 206
15.9 doc_params.h File Reference . . . 206
15.10 doc_params.h . . . 206
15.11 doc_only/documentation.h File Reference . . . 206
15.12 doc_only/documentation.h . . . 206
15.13 documentation.h File Reference . . . 206
15.14 documentation.h . . . 206
15.15 sound4.bigvoice.cl-pipe-cxx.cpp File Reference . . . 206
15.15.1 Function Documentation . . . 207
15.15.1.1 GetEnv() . . . 207
15.15.1.2 main() . . . 207
15.15.1.3 MyLogger() . . . 207
15.15.1.4 ProcessLoop() . . . 207
15.15.1.5 Usage() . . . 208
15.16 sound4.bigvoice.cl-pipe-cxx.cpp . . . 208
15.17 sound4.bigvoice.cl-pipe-dyn.cpp File Reference . . . 210
15.17.1 Function Documentation . . . 210
15.17.1.1 GetEnv() . . . 210
15.17.1.2 main() . . . 211
15.17.1.3 MyLogger() . . . 211
15.17.1.4 ProcessLoop() . . . 211
15.17.1.5 Usage() . . . 211
15.18 sound4.bigvoice.cl-pipe-dyn.cpp . . . 212
15.19 sound4.bigvoice.cl-pipe.cpp File Reference . . . 215
15.19.1 Macro Definition Documentation . . . 215
15.19.1.1 CHANNEL_COUNT . . . 215
15.19.1.2 CHUNK_SIZE . . . 215
15.19.1.3 SAMPLE_RATE . . . 216
15.19.2 Function Documentation . . . 216
15.19.2.1 main() . . . 216
15.19.2.2 MyLogger() . . . 216
15.19.2.3 ProcessLoop() . . . 216
15.19.2.4 Usage() . . . 216
15.20 sound4.bigvoice.cl-pipe.cpp . . . 217
15.21 sound4cl-pipe.cpp File Reference . . . 220
15.21.1 Function Documentation . . . 220
15.21.1.1 GetEnv() . . . 220
15.21.1.2 main() . . . 221
15.21.1.3 ProcessLoop() . . . 221
15.22 sound4cl-pipe.cpp . . . 222
15.23 sound4.bigvoice.cl.h File Reference . . . 225
15.23.1 Macro Definition Documentation . . . 228

15.23.1.1 BIGVOICE_DECL . . . 228
15.23.1.2 BIGVOICE_HAS_CLOUDBUS . . . 228
15.23.1.3 BIGVOICE_HAS_WEBSERVER . . . 228
15.23.2 Typedef Documentation . . . 228
15.23.2.1 TraceLoggingHProvider . . . 228
15.24 sound4.bigvoice.cl.h . . . 228
15.25 sound4.bigvoice.cl.hpp File Reference . . . 233
15.25.1 Function Documentation . . . 235
15.25.1.1 bigvoice_custom_exists() . . . 235
15.25.1.2 bigvoice_custom_getall() . . . 235
15.25.1.3 bigvoice_custom_getall_free() . . . 235
15.25.1.4 bigvoice_custom_reader() . . . 235
15.25.1.5 bigvoice_custom_reader_free() . . . 235
15.25.1.6 bigvoice_custom_remove() . . . 235
15.25.1.7 bigvoice_custom_rename() . . . 236
15.25.1.8 bigvoice_custom_writer() . . . 236
15.26 sound4.bigvoice.cl.hpp . . . 236
15.27 sound4.bigvoice.cl_dyn.hpp File Reference . . . 244
15.27.1 Detailed Description . . . 246
15.27.2 Macro Definition Documentation . . . 246
15.27.2.1 UNIXLIKE . . . 246
15.27.3 Function Documentation . . . 247
15.27.3.1 bigvoice_custom_exists() . . . 247
15.27.3.2 bigvoice_custom_getall() . . . 247
15.27.3.3 bigvoice_custom_getall_free() . . . 247
15.27.3.4 bigvoice_custom_reader() . . . 247
15.27.3.5 bigvoice_custom_reader_free() . . . 247
15.27.3.6 bigvoice_custom_remove() . . . 247
15.27.3.7 bigvoice_custom_rename() . . . 248
15.27.3.8 bigvoice_custom_writer() . . . 248
15.28 sound4.bigvoice.cl_dyn.hpp . . . 248
15.29 sound4cl.hpp File Reference . . . 262
15.29.1 Detailed Description . . . 263
15.29.2 Macro Definition Documentation . . . 263
15.29.2.1 UNIXLIKE . . . 263
15.30 sound4cl.hpp . . . 264
15.31 sound4cl_cdef.h File Reference . . . 277
15.31.1 Macro Definition Documentation . . . 278
15.31.1.1 SOUND4_INVALID_WEBSERVER_ID . . . 278
15.31.1.2 SOUND4_WEBSERVER_HTTP_OK . . . 278
15.31.1.3 SOUND4_WEBSERVER_HTTPS_OK . . . 278
15.31.2 Typedef Documentation . . . 278

15.31.2.1 fs_char . . . . . 278
15.31.2.2 TraceLoggingHProvider . . . . . 279
15.32 sound4cl_cdef.h . . . . . 279

**Index** **283**

# Chapter 1

# SOUND4 BIGVOICE.CL Library

## 1.1 General library information

The library interface is in C to avoid any compatibility issue.

If you are using C++, you might be better using Generic C++ access to SOUND4 .CL librairies.

**Note**

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

To access the presets/settings, you need to use JSON syntax.

Have a look at our Examples.

## 1.2 General C 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 managagement

You may communicate with the instance with JSON syntax using :

- the socket opened with bigvoice_StartUpdateThread
- the direct API, see Client's JSon config interfacing.

# Chapter 2

# Generic C++ access to SOUND4 .CL librairies

This is the easier way to integrate, and works for all .CL engines.

## 2.1 General usage

See Sample using Generic C++ interface (multi-process compatible).

The main steps are the following :

1. Load the want library with CProcessor::Load
2. Configure logging with CProcessor::SetLogSeverity and CProcessor::SetLoggerCallback
3. Allocate an CInstance
4. Configure parameters with CInstance::SetParam (see Environment variables and parameters)
5. Create the instance with CInstance::Create.
6. Create a web server with CInstance::StartWebServer
7. Send the audio to the instance using CInstance::ProcessAudio

**Note**

All `std::string` are **UTF-8** encoded

## 2.2 Communicating with the instance

If you need to communicate with the instance, you can create a CInstance::NewClient and use it with JSON syntax.

## 2.3 Metadata

You can update the metadata by calling CInstance::SetMetadata or CInstance::SetMetadataMulti.

This may be used by plugins, like Stream.CL.

## 2.4 Advanced tasks

### 2.4.1 Custom storage

You can use your own storage implementation by deriving CPresetLoader.

You have then to select it in your instance with CInstance::SetPresetManager.

Whenever some change from storage happens, you should inform the instance with CInstance::PresetManager_InformChange

### 2.4.2 Using a bus

You can group instances for processes which supports it with CInstance::SetBus

### 2.4.3 Convert helpers

You can use helpers to convert audio from different formats using Audio convert helpers (C++).

### 2.4.4 Adding a health point in the web server

Your application can define exactly what will be returned by the web server on the /apphealth path. Use CInstance::SetWebServerAppHealth.

# Chapter 3

# JSON syntax

JSON protocol to configure the process and get its state.

See also JSON syntax for internal setup

## 3.1 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 <= 0.9.7, the state resetted also for filtered out objects. So avoid alterning `object` filter between calls, or you will miss some changes.

If a property "`propname`" is an array, you can use "`propname[0]`" instead of "`propname`" for get and set to access only the first element.

```
{ "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

## 3.2 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.

## 3.3 Examples

### 3.3.1 Get status

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

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

### 3.3.2 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":{...}},
       ...
       }
```

### 3.3.3 Check license

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

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

### 3.3.4 Set Processor name

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

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

### 3.3.5 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"},
           ...
       ]}}
```

### 3.3.6 Load a preset

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

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

### 3.3.7 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"}}}
```

### 3.3.8 Set input gain

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

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

### 3.3.9 Set bypass

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

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

### 3.3.10 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

### 3.3.11 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"}
```

# Chapter 4

# 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.

## 4.1 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}
       }}}
```

## 4.2 Authentication management

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

Secret stored is

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

### 4.2.1 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.

### 4.2.2 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

### 4.2.3 Change setup account

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

#### 4.2.3.1 Set the setup/admin account

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

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

#### 4.2.3.2 Set the setup/admin account with clear password

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

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

#### 4.2.3.3 Remove the setup/admin account

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

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

### 4.2.4 Get user info

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

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

### 4.2.5 Add or modify a user

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

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

### 4.2.6 Removes a user

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

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

### 4.2.7 List users

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

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

### 4.2.8 Get/set unlogged access

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

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

### 4.2.9 Get realm

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

answer:
```
{"_auth":{"get":{"realm":"Authentification is required"}}}
```

### 4.2.10 Get API version

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

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

### 4.2.11 Get setup is admin

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

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

### 4.2.12 Get/set setup account

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

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

### 4.2.13 Get/set custom storage

Implementation defined extra storage, stored in auth.json
```
{"_auth":{"get":{"storage":null}}}
```

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

## 4.3 Get process' interface description

### 4.3.1 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 !

### 4.3.2 Get Process Image

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

answer:

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

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

### 4.3.3 Get Process Ico

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

answer:

```
{"_ico":"AAABAAUAEBAAAAEAIABo..." }
```

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

# Chapter 5

# Environment variables and parameters

## 5.1 Parameters defaulting from environment

The following environment variables are used by software library:

**Note**

Those variables can be passed as parameters when creating the instance.

**Note**

boolvalue are (case insensitive):

- 0,false,no,n : disable
- 1,true,yes,y,(empty),(any) : enable

### 5.1.1 Webserver authentification setup :

- `SETUP_USER` : The setup user name
- `SETUP_SECRET` : The setup user password
- `ADMIN_USER` : The admin user name
- `ADMIN_SECRET` : The admin user password

**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

### 5.1.2 Webserver secure :

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

### 5.1.3 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_BUNDLE : optional certificates bundle for HTTPS verification
- LICENSE_CERT_CHECK : if "false", do not check HTTPS certificates

## 5.2 Parameters

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

### 5.2.1 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=boolvalue : force the storage in read-only mode
- STATE_RO=boolvalue : force the state in read-only mode (but can save presets/auth)
- WAIT_LICENSE=boolvalue : force to wait the license load before starting (always in file mode)
- FAST_START=boolvalue : do not to try loading the license before starting (never in file mode)
- LOG_HEADER=string : use string as header for logs in instance as much as possible

### 5.2.2 Web server options:

- `COOKIE_DOMAIN=domain` : Force login cookie's domain
- `COOKIE_PATH=path` : Force login cookie's path
- `COOKIE_SECURE=mode` : Force login cookie's secure mode [true|false]
- `COOKIE_SAMESITE=mode` : Force login cookie's secure mode [default|strict|lax|none]

## 5.3 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 intances 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 immediatly 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 loosing changes.

# Chapter 6

# Deprecated List

**Member BIGVOICE_AUDIO_INPUT_COUNT**

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

**Member BIGVOICE_AUDIO_OUTPUT_COUNT**

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

**Member BIGVOICE_AUDIOFRAME_COUNT**

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

**Member BIGVOICE_CHANNEL_COUNT**

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

**Member bigvoice_GetChunkSizeInFrames ()**

Use bigvoice_GetPossibleChunkSizeInFrames

**Member bigvoice_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)**

Use bigvoice_InitProcess3

**Member bigvoice_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_↩ Key_ID, const char *Access_Key_Secret, const char *save_path, const struct bigvoice_CParameters *parameters)**

Use bigvoice_InitProcess3

**Member BIGVOICE_SAMPLE_RATE**

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

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

Use bigvoice_Webserver_tcp2

**Member sound4::bigvoice::ChunkFrames**

use GetPossibleChunkSizeInFrames

**Member sound4::bigvoice::dyn::ChunkFrames**

use GetPossibleChunkSizeInFrames()

**Member sound4::bigvoice::dyn::GetChunkSizeInFrames (CDynLib &dynlib)**

Use GetPossibleChunkSizeInFrames

**Member sound4::bigvoice::dyn::InputChannels**

Use GetChannelCount and GetAudioInputCount

**Member sound4::bigvoice::dyn::InputSampleSize**

Use GetPossibleChunkSizeInFrames and GetAudioInputCount

**Member sound4::bigvoice::dyn::OutputChannels**

Use GetChannelCount and GetAudioOutputCount

**Member sound4::bigvoice::dyn::OutputSampleSize**

Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

**Member sound4::bigvoice::dyn::SampleRate**

Use GetSampleRate

**Member sound4::bigvoice::GetChunkSizeInFrames ()**

Use GetPossibleChunkSizeInFrames

**Member sound4::bigvoice::InputChannels**

Use GetChannelCount and GetAudioInputCount

**Member sound4::bigvoice::InputSampleSize**

Use GetPossibleChunkSizeInFrames and GetAudioInputCount

**Member sound4::bigvoice::OutputChannels**

Use GetChannelCount and GetAudioOutputCount

**Member sound4::bigvoice::OutputSampleSize**

Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

**Member sound4::bigvoice::SampleRate**

Use GetSampleRate

**Member sound4::CProcessor::GetChunkSizeInFrames ()**

Use GetPossibleChunkSizeInFrames

# Chapter 7

# Topic Index

## 7.1 Topics

Here is a list of all topics with brief descriptions:


Audio convert helpers . . . 29
Audio convert helpers (C++) . . . 33
Audio managing (any audio) . . . 37
Auxiliary outputs . . . 41
Client's JSon config interfacing . . . 43
Custom preset managagement . . . 46
Direct processing . . . 51
Examples . . . 54
Global setup . . . 55
Instance tracing (Windows only) . . . 61
Instance's bus . . . 62
Metadata Management . . . 63
Process creation/destruction . . . 65
Sample using C interface . . . 69
Sample using C++ interface wrapper . . . 71
Sample using C++ interface wrapper with dynamic library loading . . . 72
Sample using Generic C++ interface (multi-process compatible) . . . 74
Update Management . . . 76
Web server . . . 78

# Chapter 8

# Namespace Index

## 8.1 Namespace List

Here is a list of all namespaces with brief descriptions:


sound4 . . . . . 83
sound4::bigvoice . . . . . 86
sound4::bigvoice::dyn . . . . . 94
sound4::bigvoice::dyn::helper . . . . . 103
sound4::bigvoice::helper . . . . . 108
sound4::helper . . . . . 113

# Chapter 9

# Hierarchical Index

## 9.1 Class Hierarchy

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


sound4::AuxiliaryOutputInfos . . . 115
bigvoice_CBus . . . 116
bigvoice_CClientInstance . . . 116
bigvoice_CInstance . . . 117
bigvoice_CParameters . . . 117
sound4::bigvoice::CBus . . . 117
sound4::bigvoice::dyn::CBus . . . 118
sound4::CBus . . . 119
sound4::bigvoice::CInstance::CClient . . . 121
sound4::bigvoice::dyn::CInstance::CClient . . . 122
sound4::CInstance::CClient . . . 123
sound4::bigvoice::dyn::CDynLib . . . 125
sound4::bigvoice::dyn::helper::CDynLoader . . . 137
sound4::helper::CDynLoader . . . 140
sound4::bigvoice::CInstance . . . 142
sound4::bigvoice::dyn::CInstance . . . 155
sound4::CInstance . . . 167
sound4::bigvoice::CPresetLoader . . . 180
sound4::bigvoice::dyn::CPresetLoader . . . 182
MyPresetManager . . . 195
sound4::CPresetLoader . . . 184
MyPresetManager . . . 195
sound4::CProcessor . . . 186
sound4::bigvoice::dyn::helper::DynFuncHolder< T > . . . 192
sound4::helper::DynFuncHolder< T > . . . 193
sound4::bigvoice::dyn::helper::SampleFormat< T > . . . 199
sound4::bigvoice::helper::SampleFormat< T > . . . 199
sound4::helper::SampleFormat< T > . . . 201
SOUND4_AuxiliaryOutputInfos . . . 201
SOUND4_ProcessInfo . . . 202

# Chapter 10

# Class Index

## 10.1 Class List

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


sound4::AuxiliaryOutputInfos . . . 115
bigvoice_CBus . . . 116
bigvoice_CClientInstance . . . 116
bigvoice_CInstance . . . 117
bigvoice_CParameters . . . 117
sound4::bigvoice::CBus
    Shared Bus . . . 117
sound4::bigvoice::dyn::CBus
    Shared Bus . . . 118
sound4::CBus
    Shared Bus . . . 119
sound4::bigvoice::CInstance::CClient . . . 121
sound4::bigvoice::dyn::CInstance::CClient . . . 122
sound4::CInstance::CClient . . . 123
sound4::bigvoice::dyn::CDynLib
    Dynamic library interface . . . 125
sound4::bigvoice::dyn::helper::CDynLoader
    Helper to load dynamic library . . . 137
sound4::helper::CDynLoader
    Helper to load dynamic library . . . 140
sound4::bigvoice::CInstance
    Instance handling class . . . 142
sound4::bigvoice::dyn::CInstance
    Instance handling class . . . 155
sound4::CInstance
    Instance handling class . . . 167
sound4::bigvoice::CPresetLoader
    Custom preset handler helper . . . 180
sound4::bigvoice::dyn::CPresetLoader
    Custom preset handler helper . . . 182
sound4::CPresetLoader
    Custom preset handler helper . . . 184
sound4::CProcessor
    Dynamic library interface . . . 186
sound4::bigvoice::dyn::helper::DynFuncHolder< T > 
    Helper to wrap a function pointer . . . 192

sound4::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 193
MyPresetManager
[Custom preset manager] . . . 195
sound4::bigvoice::dyn::helper::SampleFormat< T >
Helper for Sample format types . . . 199
sound4::bigvoice::helper::SampleFormat< T >
Helper for Sample format types . . . 199
sound4::helper::SampleFormat< T >
Helper for Sample format types . . . 201
SOUND4_AuxiliaryOutputInfos . . . 201
SOUND4_ProcessInfo . . . 202

# Chapter 11

# File Index

## 11.1 File List

Here is a list of all files with brief descriptions:


doc_examples.h . . . 205
doc_gencpp.h . . . 205
doc_json.h . . . 205
doc_main.h . . . 206
doc_params.h . . . 206
doc_only/documentation.h . . . 206
documentation.h . . . 206
sound4.bigvoice.cl-pipe-cxx.cpp
C++ sample using C++ interface wrapper . . . 206
sound4.bigvoice.cl-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading . . . 210
sound4.bigvoice.cl-pipe.cpp
C++ sample using only C interface . . . 215
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided . . . 220
sound4.bigvoice.cl.h
C interface for library . . . 225
sound4.bigvoice.cl.hpp
C++ wrapper for library . . . 233
sound4.bigvoice.cl_dyn.hpp
C++ wrapper for dynamic loading library . . . 244
sound4cl.hpp
C++ wrapper for dynamic loading library . . . 262
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries . . . 277

# Chapter 12

# Topic Documentation

## 12.1 Audio convert helpers

Helpers to convert audio.

### 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)

### 12.1.1 Detailed Description

Helpers to convert audio in an optimized way

### 12.1.2 Function Documentation

#### 12.1.2.1 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**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

#### 12.1.2.2 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**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

#### 12.1.2.3 bigvoice_AudioMonoFromLiveStereo()

```
void bigvoice_AudioMonoFromLiveStereo (
            const uint8_t * payload,
            float * output )
```

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

**Parameters**

| | |
|---|---|
| *payload* | Livewire Live Stereo raw buffer |
| *output* | audio output mono buffer (64 byte aligned) |

#### 12.1.2.4 bigvoice_AudioMonoToLiveStereo()

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

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

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *payload* | Livewire Live Stereo raw buffer |

#### 12.1.2.5 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**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *output* | audio output stereo buffer (64 byte aligned) |
| *nFrame* | the number of frames |

#### 12.1.2.6 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**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *outputL* | audio output left buffer (64 byte aligned) |
| *outputR* | audio output right buffer (64 byte aligned) |
| *nFrame* | the number of frames |

#### 12.1.2.7 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**

| | |
|---|---|
| *input* | audio input stereo buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

#### 12.1.2.8 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**

| | |
|---|---|
| *inputL* | audio input left buffer (64 byte aligned) |
| *inputR* | audio input right buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

## 12.2 Audio convert helpers (C++)

Helpers to convert audio (C++)

### Functions

- void sound4::CProcessor::AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)

- void sound4::CProcessor::AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
- void sound4::CProcessor::StereoToMono (const float *input, float *output, size_t nFrame)
- void sound4::CProcessor::MonoToStereo (const float *input, float *output, size_t nFrame)
- void sound4::CProcessor::StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void sound4::CProcessor::MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void sound4::CProcessor::AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- void sound4::CProcessor::AudioMonoToLiveStereo (const float *input, uint8_t *payload)

### 12.2.1 Detailed Description

Helpers to convert audio in an optimized way

### 12.2.2 Function Documentation

#### 12.2.2.1 AudioConvertFrom()

```
void sound4::CProcessor::AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            SampleFormat fmt )  [inline]
```

Convert the payload to floating-point array

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 637 of file sound4cl.hpp.

#### 12.2.2.2 AudioConvertTo()

```
void sound4::CProcessor::AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            SampleFormat fmt )  [inline]
```

Convert floating-point array to payload

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 650 of file sound4cl.hpp.

#### 12.2.2.3 AudioMonoFromLiveStereo()

```
void sound4::CProcessor::AudioMonoFromLiveStereo (
            const uint8_t * payload,
            float * output )  [inline]
```

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

**Parameters**

| | |
|---|---|
| *payload* | Livewire Live Stereo raw buffer |
| *output* | audio output mono buffer (64 byte aligned) |

Definition at line 709 of file sound4cl.hpp.

#### 12.2.2.4 AudioMonoToLiveStereo()

```
void sound4::CProcessor::AudioMonoToLiveStereo (
            const float * input,
            uint8_t * payload )  [inline]
```

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

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *payload* | Livewire Live Stereo raw buffer |

Definition at line 718 of file sound4cl.hpp.

#### 12.2.2.5 MonoToStereo()

```
void sound4::CProcessor::MonoToStereo (
            const float * input,
```

```
            float * output,
            size_t nFrame )  [inline]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *output* | audio output stereo buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 674 of file sound4cl.hpp.

#### 12.2.2.6 MonoToStereo_Planar()

```
void sound4::CProcessor::MonoToStereo_Planar (
            const float * input,
            float * outputL,
            float * outputR,
            size_t nFrame )  [inline]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *outputL* | audio output left buffer (64 byte aligned) |
| *outputR* | audio output right buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 700 of file sound4cl.hpp.

#### 12.2.2.7 StereoToMono()

```
void sound4::CProcessor::StereoToMono (
            const float * input,
            float * output,
            size_t nFrame )  [inline]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input stereo buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 662 of file sound4cl.hpp.

#### 12.2.2.8 StereoToMono_Planar()

```
void sound4::CProcessor::StereoToMono_Planar (
            const float * inputL,
            const float * inputR,
            float * output,
            size_t nFrame )  [inline]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *inputL* | audio input left buffer (64 byte aligned) |
| *inputR* | audio input right buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 687 of file sound4cl.hpp.

## 12.3 Audio managing (any audio)

Process audio in any format and size.

### 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)

## 12.3.1 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.

## 12.3.2 Enumeration Type Documentation

### 12.3.2.1 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 879 of file sound4.bigvoice.cl.h.

### 12.3.3 Function Documentation

#### 12.3.3.1 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**

| | |
|---|---|
| *instance* | the processing instance |
| *payload* | the binary raw audio buffer |
| *nFrame* | the number of frames in the payload |
| *fmt* | 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

#### 12.3.3.2 bigvoice_AddPadAudio()

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

Add silence to finish the current audio chunk and process it

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

number of added silence frame

#### 12.3.3.3 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**

| | |
|---|---|
| *instance* | the processing instance |
| *payload* | the binary raw audio buffer to store output |
| *max_nFrame* | the maximum number of frames that can be stored in the payload |
| *fmt* | the wanted format of the payload |

**Returns**

number of frames put in the buffer

#### 12.3.3.4 bigvoice_GetBytesFromFormat()

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

Get the format byte size

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

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

#### 12.3.3.5 bigvoice_GetFormatFromName()

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

Convert string to format

**Parameters**

| | |
|---|---|
| *name* | the format string |

**Returns**

the matching format, or INVALID_FORMAT if invalid.

#### 12.3.3.6 bigvoice_GetFormatName()

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

Convert format to string

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

the string name of the format

#### 12.3.3.7 bigvoice_GetMaxPacketFrame()

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

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

**Parameters**

| | |
|---|---|
| *instance* | The process instance |

**Returns**

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

#### 12.3.3.8 bigvoice_GetOutputCount()

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

Get the available output size

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

number of available frames at output

## 12.4 Auxiliary outputs

### Classes

- struct SOUND4_AuxiliaryOutputInfos

**Functions**

- struct SOUND4_AuxiliaryOutputInfos ** bigvoice_GetAuxiliaryAudioOutputInfos (struct bigvoice_CInstance *instance)
- void bigvoice_FreeAuxiliaryAudioInfos (struct SOUND4_AuxiliaryOutputInfos **infos)
- int bigvoice_ActivateAuxiliaryAudioID (struct bigvoice_CInstance *instance, const char *id)
- unsigned int bigvoice_GetAuxiliaryAudioEstimatedDelay (struct bigvoice_CInstance *instance, const char *id)

### 12.4.1 Detailed Description

For advanced processing with multiple simultaneous outputs.

Auxiliary outputs are filled at each call to bigvoice_ProcessAudio or bigvoice_ProcessAudio_Planar.

You can use their buffer to retreive their output.

### 12.4.2 Function Documentation

#### 12.4.2.1 bigvoice_ActivateAuxiliaryAudioID()

```
int bigvoice_ActivateAuxiliaryAudioID (
            struct bigvoice_CInstance * instance,
            const char * id )
```

Activate an auxiliary output.

This is required to activate the output generation

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |
| *id* | the auxiliary output ID to activate |

**Returns**

0 is success

#### 12.4.2.2 bigvoice_FreeAuxiliaryAudioInfos()

```
void bigvoice_FreeAuxiliaryAudioInfos (
            struct SOUND4_AuxiliaryOutputInfos ** infos )
```

Frees the information returned by bigvoice_GetAuxiliaryAudioOutputInfos.

**Parameters**

| | |
|---|---|
| *infos* | the infos to free |

#### 12.4.2.3 bigvoice_GetAuxiliaryAudioEstimatedDelay()

```
unsigned int bigvoice_GetAuxiliaryAudioEstimatedDelay (
            struct bigvoice_CInstance * instance,
            const char * id )
```

Get the delay (estimated) of current processing algorithm for a specific auxiliary output

**Remarks**

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

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |
| *id* | the auxiliary output ID. Use NULL or empty string for main output delay. |

**Returns**

Estimated delay in frames, or 0 if unknown id.

#### 12.4.2.4 bigvoice_GetAuxiliaryAudioOutputInfos()

```
struct SOUND4_AuxiliaryOutputInfos ** bigvoice_GetAuxiliaryAudioOutputInfos (
            struct bigvoice_CInstance * instance )
```

Get the list of possible auxiliary outputs.

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

a list of SOUND4_AuxiliaryOutputInfos, null terminated, of auxiliary outputs. May be null. Should be freed with bigvoice_FreeAuxiliaryAudioInfos

## 12.5 Client's JSon config interfacing

Communicate with the instance.

**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)

## 12.5.1 Detailed Description

Allow low-level JSON configuration. see also JSON syntax

## 12.5.2 Function Documentation

### 12.5.2.1 bigvoice_DeleteClient()

```
void bigvoice_DeleteClient (
            struct bigvoice_CClientInstance * client )
```

Deletes a client

**Parameters**

| | |
|---|---|
| *client* | the client instance |

### 12.5.2.2 bigvoice_FreeJsonAnswer()

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

Free a JSON answer returned by bigvoice_ProcessJson

**Parameters**

| | |
|---|---|
| *json_str* | the JSON request |

### 12.5.2.3 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**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

A new client instance

#### 12.5.2.4 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**

| | |
|---|---|
| *client* | the client instance |
| *json_str* | the JSON request |
| *need_save* | on return will be 1 is saving is required (maybe delayed) |

**Returns**

a JSON answer (to be freed with bigvoice_FreeJsonAnswer)

#### 12.5.2.5 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**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

0 if success, <0 on error

## 12.6 Custom preset managagement

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)

### 12.6.1 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 overriden 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)

### 12.6.2 Typedef Documentation

#### 12.6.2.1 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**

| | |
|---|---|
| *filename* | the name of the file to test. |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

-1 if not exists, 0 if it exists

Definition at line 515 of file sound4.bigvoice.cl.h.

#### 12.6.2.2 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**

| | |
|---|---|
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

A NULL terminated array of file names

Definition at line 524 of file sound4.bigvoice.cl.h.

#### 12.6.2.3 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**

| | |
|---|---|
| *all* | the returned value from bigvoice_storage_getall |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

Definition at line 532 of file sound4.bigvoice.cl.h.

#### 12.6.2.4 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**

| | |
|---|---|
| *filename* | the name of the file to read. |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

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

Definition at line 488 of file sound4.bigvoice.cl.h.

#### 12.6.2.5 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**

| | |
|---|---|
| *content* | the returned value from bigvoice_storage_reader |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

Definition at line 496 of file sound4.bigvoice.cl.h.

#### 12.6.2.6 bigvoice_storage_remove

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

Function to call to remove a stored file.

**Parameters**

| | |
|---|---|
| *filename* | the name of the file to remove. |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

-1 if failed, 0 if OK

Definition at line 541 of file sound4.bigvoice.cl.h.

#### 12.6.2.7 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**

| | |
|---|---|
| *from* | The source file name (must exist) |
| *to* | The destination file name (must not exist) |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

-1 if failed, 0 if OK

Definition at line 551 of file sound4.bigvoice.cl.h.

#### 12.6.2.8 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**

| | |
|---|---|
| *filename* | the name of the file to write. |
| *content* | the **UTF-8** content to write to the file |
| *userdata* | the userdata passed to bigvoice_SetPresetManager |

**Returns**

-1 if failed, 0 if OK

Definition at line 506 of file sound4.bigvoice.cl.h.

#### 12.6.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 476 of file sound4.bigvoice.cl.h.

### 12.6.3 Enumeration Type Documentation

#### 12.6.3.1 bigvoice_PresetChange_Kind

```
enum bigvoice_PresetChange_Kind
```

Kind of change on a storage path

**Enumerator**

| | |
|---|---|
| change_kind_created | Path has been created. |
| change_kind_modified | Path has been modified. |
| change_kind_deleted | Path has been deleted. |

Definition at line 584 of file sound4.bigvoice.cl.h.

### 12.6.4 Function Documentation

#### 12.6.4.1 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 conjonction with bigvoice_SetPresetManager

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |
| *relative_path* | The path which changed, relative to the root storage |
| *change_kind* | What happened to this path |

#### 12.6.4.2 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**

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

## 12.7 Direct processing

Optimized processing with direct in->out.

### 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)

### 12.7.1 Detailed Description

Optimized processing with direct in->out.

### 12.7.2 Function Documentation

#### 12.7.2.1 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**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

An audio input dedicated buffer.

#### 12.7.2.2 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**

| | |
|---|---|
| *instance* | the processing instance |

**Returns**

An audio output dedicated buffer.

#### 12.7.2.3 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.

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

**Returns**

Estimated delay in frames

#### 12.7.2.4 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

Ouput 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**

| | |
|---|---|
| *instance* | the processing instance |
| *input* | audio input buffer (64 byte aligned) or NULL to use internal buffer bigvoice_GetBufferIn |
| *output* | audio output buffer (64 byte aligned) or NULL to use internal buffer bigvoice_GetBufferOut |

#### 12.7.2.5 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

Ouput 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**

| | |
|---|---|
| *instance* | the processing instance |
| *input* | array of audio input buffers for each input channel (no alignment required) |
| *output* | array of audio output buffers for each input channel (no alignment required) |

## 12.8 Examples

List all examples.

The C header is sound4.bigvoice.cl.h.

You can also use the C++ wrappers :

- sound4cl.hpp (using dynamic library loading, compatible with all .CL engines)
- sound4.bigvoice.cl.hpp
- sound4.bigvoice.cl_dyn.hpp (using dynamic library loading)

See Sample using C interface

See Sample using Generic C++ interface (multi-process compatible)

See Sample using C++ interface wrapper

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

## 12.9 Global setup

Get information on the library.

### Classes

- struct SOUND4_ProcessInfo

### 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 struct SOUND4_ProcessInfo ∗ SOUND4_GetProcessInfo ()
- 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)

### 12.9.1 Detailed Description

Get information on the library. Set logging mode.

### 12.9.2 Macro Definition Documentation

#### 12.9.2.1 BIGVOICE_AUDIO_INPUT_COUNT

```
#define BIGVOICE_AUDIO_INPUT_COUNT 1
```

This version number of inputs for this process.

**Deprecated** For binary compatibility, this should be runtime checked to be the same as bigvoice_GetAudioInputCount()

Definition at line 58 of file sound4.bigvoice.cl.h.

#### 12.9.2.2 BIGVOICE_AUDIO_OUTPUT_COUNT

```
#define BIGVOICE_AUDIO_OUTPUT_COUNT 1
```

This version number of outputs for this process.

**Deprecated** For binary compatibility, this should be runtime checked to be the same as bigvoice_GetAudioOutputCount()

Definition at line 64 of file sound4.bigvoice.cl.h.

#### 12.9.2.3 BIGVOICE_AUDIOFRAME_COUNT

```
#define BIGVOICE_AUDIOFRAME_COUNT 12
```

This version number of frames in a process chunk.

**Deprecated** For binary compatibility, this should be runtime checked to be the same as bigvoice_GetChunkSizeInFrames()

Definition at line 46 of file sound4.bigvoice.cl.h.

#### 12.9.2.4 BIGVOICE_CHANNEL_COUNT

```
#define BIGVOICE_CHANNEL_COUNT 1
```

This version number of channels.

**Deprecated** For binary compatibility, this should be runtime checked to be the same as bigvoice_GetChannelCount()

Definition at line 52 of file sound4.bigvoice.cl.h.

#### 12.9.2.5 BIGVOICE_SAMPLE_RATE

```
#define BIGVOICE_SAMPLE_RATE 48000
```

This input/output sample-rate.

**Deprecated** For binary compatibility, this should be runtime checked to be the same as bigvoice_GetSampleRate()

Definition at line 70 of file sound4.bigvoice.cl.h.

### 12.9.3 Typedef Documentation

#### 12.9.3.1 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 185 of file sound4.bigvoice.cl.h.

### 12.9.4 Enumeration Type Documentation

#### 12.9.4.1 bigvoice_LogSeverity

```
enum bigvoice_LogSeverity
```

The logging levels

**Enumerator**

| | |
|---|---|
| none | Not used. |
| fatal | fatal error |
| error | error |
| warning | warning |
| info | info |
| verbose | verbose |
| verbose2 | verbose2 |
| verbose3 | verbose3 |
| verbose4 | verbose4 |
| verbose5 | verbose5 |

Definition at line 166 of file sound4.bigvoice.cl.h.

### 12.9.5 Function Documentation

#### 12.9.5.1 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

#### 12.9.5.2 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

#### 12.9.5.3 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

#### 12.9.5.4 bigvoice_GetChunkSizeInFrames()

```
unsigned int bigvoice_GetChunkSizeInFrames ( )
```

Get the default 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

#### 12.9.5.5 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

#### 12.9.5.6 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

#### 12.9.5.7 bigvoice_GetVersion()

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

Get the process version string

**Returns**

version string

#### 12.9.5.8 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**

| | |
|---|---|
| *logger* | A C callback called for logging. |

#### 12.9.5.9 bigvoice_SetLogSeverity()

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

Change the global log severity

Logs above this level will not be printed.

**Parameters**

| | |
|---|---|
| *severity* | Log severity |

#### 12.9.5.10 SOUND4_GetProcessInfo()

```
const struct SOUND4_ProcessInfo * SOUND4_GetProcessInfo ( )
```

Get this processor description

Common to all SOUND4 processors.

**Returns**

the process description

## 12.10 Instance tracing (Windows only)

This allows instances to use windows event tracing.

### Functions

- void bigvoice_SetInstanceTracing (struct bigvoice_CParameters *params, TraceLoggingHProvider tracing_provider, GUID activity_guid)
- void bigvoice_SetInstanceTracingProcessActivity (struct bigvoice_CInstance *instance, GUID activity_guid)

### 12.10.1 Detailed Description

This allows instances to use windows event tracing

### 12.10.2 Function Documentation

#### 12.10.2.1 bigvoice_SetInstanceTracing()

```
void bigvoice_SetInstanceTracing (
            struct bigvoice_CParameters * params,
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid )
```

Set the instance's tracing provider before creating instance

**Parameters**

| | |
|---|---|
| *params* | the parameters to configure |
| *tracing_provider* | the provider as created with TRACELOGGING_DEFINE_PROVIDER |
| *activity_guid* | the activity guid, used for creation, updates and destruction |

#### 12.10.2.2 bigvoice_SetInstanceTracingProcessActivity()

```
void bigvoice_SetInstanceTracingProcessActivity (
            struct bigvoice_CInstance * instance,
            GUID activity_guid )
```

Set the instance's process activity for tracing (optional). Should be called only in process's thread.

**Parameters**

| | |
|---|---|
| *instance* | the instance to configure |
| *activity_guid* | the activity guid |

## 12.11 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)

### 12.11.1 Detailed Description

This allows instances to exchange real-time data.

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

### 12.11.2 Function Documentation

#### 12.11.2.1 bigvoice_FreeBus()

```
void bigvoice_FreeBus (
            struct bigvoice_CBus * bus )
```

Frees the bus allocated with bigvoice_NewBus

**Parameters**

| | |
|---|---|
| *bus* | The bus to free |

#### 12.11.2.2 bigvoice_NewBus()

```
struct bigvoice_CBus * bigvoice_NewBus ( )
```

Creates a new bus

**Returns**

the new bus, to be freed with bigvoice_FreeBus

#### 12.11.2.3 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**

| | |
|---|---|
| *params* | the parameters to configure |
| *bus* | the bus to use |

## 12.12 Metadata Management

Send/update metadata.

### Functions

- void bigvoice_SetMetadata (struct bigvoice_CInstance *instance, const char *key, const char *value)
- void bigvoice_SetMetadataMulti (struct bigvoice_CInstance *instance, const char **keyvalue)
- const char ** bigvoice_GetMetadataInfos (struct bigvoice_CInstance *instance)
- void bigvoice_FreeMetadataInfos (struct bigvoice_CInstance *instance, const char **infos)

### 12.12.1 Detailed Description

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

### 12.12.2 Function Documentation

#### 12.12.2.1 bigvoice_FreeMetadataInfos()

```
void bigvoice_FreeMetadataInfos (
            struct bigvoice_CInstance * instance,
            const char ** infos )
```

Frees the information returned by bigvoice_GetMetadataInfos.

**Parameters**

| | |
|---|---|
| *instance* | the instance |
| *infos* | the infos |

#### 12.12.2.2 bigvoice_GetMetadataInfos()

```
const char ** bigvoice_GetMetadataInfos (
            struct bigvoice_CInstance * instance )
```

Get information on known metadata keys.

**Parameters**

| | |
|---|---|
| *instance* | 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

#### 12.12.2.3 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).

**See also**

bigvoice_SetMetadataMulti

**Parameters**

| | |
|---|---|
| *instance* | the instance |
| *key* | 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. |
| *value* | the metadata value or nullptr to erase it. |

#### 12.12.2.4 bigvoice_SetMetadataMulti()

```
void bigvoice_SetMetadataMulti (
            struct bigvoice_CInstance * instance,
            const char ** keyvalue )
```

Adds multiple metadata to the process.

This works like bigvoice_SetMetadata but with multiple metadata at once.

**See also**

bigvoice_SetMetadata

**Parameters**

| | |
|---|---|
| *instance* | the instance |
| *keyvalue* | a list of key/value pair, ending with null key. |

## 12.13 Process creation/destruction

Create and destroy a process instance.

### 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 *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct bigvoice_CInstance * bigvoice_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct bigvoice_CParameters *parameters)
- struct bigvoice_CInstance * bigvoice_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_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)

### 12.13.1 Detailed Description

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

### 12.13.2 Function Documentation

#### 12.13.2.1 bigvoice_ExitProcess()

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

Quit and destroy the instance

**Parameters**

| | |
|---|---|
| *instance* | the instance to destroy |

#### 12.13.2.2 bigvoice_FreeParameters()

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

Free the parameters created with bigvoice_NewParameters.

#### 12.13.2.3 bigvoice_FreeParameterValue()

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

Free a parameter value returned by bigvoice_GetParameter

**Parameters**

| | |
|---|---|
| *value* | the returned value |

#### 12.13.2.4 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

#### 12.13.2.5 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**

| | |
|---|---|
| *instance* | 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

#### 12.13.2.6 bigvoice_InitProcess()

```
struct bigvoice_CInstance * bigvoice_InitProcess (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_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

#### 12.13.2.7 bigvoice_InitProcess2()

```
struct bigvoice_CInstance * bigvoice_InitProcess2 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_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

#### 12.13.2.8 bigvoice_InitProcess3()

```
struct bigvoice_CInstance * bigvoice_InitProcess3 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_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 managagement.

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

**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.

See also Environment variables and parameters

**Parameters**

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

**Returns**

A new process instance

#### 12.13.2.9 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).

#### 12.13.2.10 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

#### 12.13.2.11 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**

| | |
|---|---|
| *instance* | the instance to terminate |

**Returns**

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

## 12.14 Sample using C interface

C example.

See full example cpp/sound4.bigvoice.cl-pipe.cpp

### 12.14.1 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);
     }*/
```

### 12.14.2 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];

    #if defined(_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;
        }
    }
```

### 12.14.3 Sample to stop the processing instance

see Process creation/destruction

```
#if (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);
```

## 12.15 Sample using C++ interface wrapper

C++ example (static link)

See full example cpp/sound4.bigvoice.cl-pipe-cxx.cpp

### 12.15.1 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");
```

### 12.15.2 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;
    }
```

### 12.15.3 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;
```

### 12.15.4 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;
}
```

### 12.15.5 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;
    }
}
```

## 12.16 Sample using C++ interface wrapper with dynamic library loading

C++ example (dynamic link) See full example cpp/sound4.bigvoice.cl-pipe-dyn.cpp.

### 12.16.1 Dynamic loading

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

### 12.16.2 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");
```

### 12.16.3 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;
}
```

### 12.16.4 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;
```

### 12.16.5 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;
    }
```

### 12.16.6 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;
        }
    }
```

### 12.16.7 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;
            std::error_code ec;
            if ( !std::filesystem::exists(m_preset_path,ec ) ) {
                return false;
            }
            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;
            std::error_code ec;
            for (auto it = std::filesystem::directory_iterator(m_preset_path.c_str(),
      std::filesystem::directory_options::skip_permission_denied, ec);
                 !ec && it != std::filesystem::directory_iterator{};
                  it.increment(ec) )
            {
                if (it->is_regular_file(ec)) {
                    list.push_back(it->path().lexically_relative(m_preset_path));
                }
            }
            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;
        }
};
```

### 12.16.8 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);
    }
```

## 12.17 Sample using Generic C++ interface (multi-process compatible)

C++ example (dynamic link, any process)

This wrapper abstracts the processing specifics, working with any SOUND4 .CL process.

It uses sound4cl.hpp header.

See full example cpp/sound4cl-pipe.cpp

### 12.17.1 Loading the library

```
    std::string libname =  "libsound4.bigvoice.cl.so";
    sound4::CProcessor processor;
    if (!processor.Load(libname)) {
        std::cerr « "Failed to load processing library" « std::endl;
        return 1;
    }
```

### 12.17.2 Setup and configure

```
    processor.SetLogSeverity(sound4::verbose5);
    processor.SetLoggerCallback([](sound4::LogSeverity severity,const std::string& msg) {
            std::cerr « msg « std::endl;
        });

    sound4::CInstance instance(processor);
    instance.SetParam("ADMIN_USER","admin");
    instance.SetParam("ADMIN_SECRET","admin");
```

### 12.17.3 Create the processing instance

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

### 12.17.4 Processing loop

```
    size_t input_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
    size_t output_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
    while (!stopthread)
    {
        // Read a chunk from stdin
        if (std::fread(instance.GetBufferIn(), sizeof(float), input_sample_count, stdin) !=
      input_sample_count) {
            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(), sizeof(float), output_sample_count, stdout) !=
      output_sample_count) {
            std::cerr « "Failed to write audio output" « std::endl;
            stopthread=true;
        }
    }
```

### 12.17.5 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;
    }
```

### 12.17.6 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;
```

### 12.17.7 Custom preset manager example

```
class MyPresetManager : public sound4::CPresetLoader {
public:
    MyPresetManager() = default;
    virtual ~MyPresetManager() = default;

    bool SetupPath(const std::filesystem::path& a_preset_path)
        {
            m_preset_path=a_preset_path;
            std::error_code ec;
            if ( !std::filesystem::exists(m_preset_path,ec ) ) {
                return false;
            }
            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;
            std::error_code ec;
            for (auto it = std::filesystem::directory_iterator(m_preset_path.c_str(),
      std::filesystem::directory_options::skip_permission_denied, ec);
                !ec && it != std::filesystem::directory_iterator{};
                 it.increment(ec) )
            {
                if (it->is_regular_file(ec)) {
                    list.push_back(it->path().lexically_relative(m_preset_path));
                }
            }
            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;
        }
};
```

### 12.17.8 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);
    }
```

## 12.18 Update Management

Manage the process update.

#### 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)

### 12.18.1 Detailed Description

Manage the process update and communications.

### 12.18.2 Function Documentation

#### 12.18.2.1 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**

| | |
|---|---|
| *instance* | the instance |
| *port* | the port to listen for incoming JSON connections. If 0, the JSON server is not accessible |

#### 12.18.2.2 bigvoice_StopUpdateThread()

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

Stop the update thread

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |

#### 12.18.2.3 bigvoice_WaitUpdateThreadReady()

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

Wait for the update thread to be ready

**Parameters**

| | |
|---|---|
| *instance* | the processing instance |
| *milliseconds* | maximum time to wait in milliseconds |

**Returns**

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

## 12.19 Web server

Web server configuring.

### 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)
- void bigvoice_Webserver_SetAppHealth (struct bigvoice_CInstance *instance, int httpcode, const char *contenttype, const char *content)
- void bigvoice_Webserver_GetAppHealth (struct bigvoice_CInstance *instance, int *httpcode, char **contenttype, char **content)
- void bigvoice_Webserver_FreeString (char *str)

### 12.19.1 Detailed Description

Run a web server to configure the processor

The web server also expose `/health` which should always answer 200.

The web server also expose `/licvalid` which should answer 200 when license is valid, 402 when it is not valid, other standard code for other errors.

You can also use `/livalid/[module]` (where `[module]` is the name of a module) to check for validity of the [module] extension in the same way. For instance '/livalid/stream' will check the stream extension.

See also how to setup authentication with Environment variables and parameters. Their use is documented in bigvoice_InitProcess3.

### 12.19.2 Macro Definition Documentation

#### 12.19.2.1 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 1266 of file sound4.bigvoice.cl.h.

#### 12.19.2.2 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 1271 of file sound4.bigvoice.cl.h.

#### 12.19.2.3 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 1275 of file sound4.bigvoice.cl.h.

### 12.19.3 Function Documentation

#### 12.19.3.1 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**

| | |
|---|---|
| *listenport* | The port to listen for HTTP connections (0 to disable) |
| *listenport_secure* | The port to listen for HTTPS connections (0 to disable) |
| *instance* | The process instance to connect to |

**Returns**

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

#### 12.19.3.2 bigvoice_Webserver_FreeString()

```
void bigvoice_Webserver_FreeString (
            char * str )
```

Free a a string returned by bigvoice_Webserver_GetAppHealth

**Parameters**

| | |
|---|---|
| *str* | the string to free |

#### 12.19.3.3 bigvoice_Webserver_GetAppHealth()

```
void bigvoice_Webserver_GetAppHealth (
            struct bigvoice_CInstance * instance,
            int * httpcode,
            char ** contenttype,
            char ** content )
```

Get the current web server /apphealth returned http code,content-type and content

**Parameters**

| | |
|---|---|
| *instance* | The process instance |
| *httpcode* | The http code pointer |
| *contenttype* | The http Content-Type header pointer (to be freed with bigvoice_Webserver_FreeString) |
| *content* | The http body content pointer (to be freed with bigvoice_Webserver_FreeString) |

#### 12.19.3.4 bigvoice_Webserver_SetAppHealth()

```
void bigvoice_Webserver_SetAppHealth (
            struct bigvoice_CInstance * instance,
            int httpcode,
            const char * contenttype,
            const char * content )
```

Set the web server /apphealth status info

The hosting application can use this to customize the handling of /apphealth path.

**Parameters**

| | |
|---|---|
| *instance* | The process instance |
| *httpcode* | The http code |
| *contenttype* | The http Content-Type header |
| *content* | The http body content |

#### 12.19.3.5 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**

| | |
|---|---|
| *id* | 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

#### 12.19.3.6 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**

| | |
|---|---|
| *id* | The web server Id, returned by bigvoice_Webserver_tcp or bigvoice_Webserver |
| *timeout_ms* | The timeout to wait gracefull shutdown |

**Returns**

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

#### 12.19.3.7 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

Same as bigvoice_Webserver_tcp2, but with default parameters (from env)

**Deprecated** Use bigvoice_Webserver_tcp2

#### 12.19.3.8 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**

| | |
|---|---|
| *listenport* | The port to listen for HTTP connections |
| *listenport_secure* | The port to listen for HTTPS connections |
| *socket_ip* | The ip of the process to connect to |
| *socket_port* | The port of the process's JSON server to connect to |
| *parameters* | The parameters to use, created with bigvoice_NewParameters |

**Returns**

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

# Chapter 13

# Namespace Documentation

## 13.1 sound4 Namespace Reference

### Namespaces

- namespace bigvoice
- namespace helper

### Classes

- struct AuxiliaryOutputInfos
- class CBus

  *Shared Bus.*
- class CInstance

  *Instance handling class.*
- class CPresetLoader

  *Custom preset handler helper.*
- class CProcessor

  *Dynamic library interface.*

### Typedefs

- using log_cb_t = std::function< void(LogSeverity, const std::string &)>

### Enumerations

- enum SampleFormat {
  INVALID_FORMAT , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE , S24_NATIVE , S32_NATIVE ,
  F32_NATIVE }
- enum LogSeverity {
  none , fatal , error , warning ,
  info , verbose , verbose2 , verbose3 ,
  verbose4 , verbose5 }
- enum PresetChange_Kind { change_kind_created , change_kind_modified , change_kind_deleted }

**Functions**

- static void _log_cb_c (sound4cl_LogSeverity severity, const char *c_msg)

**Variables**

- static log_cb_t _log_cb

### 13.1.1 Detailed Description

SOUND4 namespace

### 13.1.2 Typedef Documentation

#### 13.1.2.1 log_cb_t

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

Log callback type

Definition at line 291 of file sound4cl.hpp.

### 13.1.3 Enumeration Type Documentation

#### 13.1.3.1 LogSeverity

```
enum sound4::LogSeverity
```

Log Severity levels

**Enumerator**

| | |
|---|---|
| none | Not used. |
| fatal | fatal error |
| error | error |
| warning | warning |
| info | info |
| verbose | verbose |
| verbose2 | verbose2 |
| verbose3 | verbose3 |
| verbose4 | verbose4 |
| verbose5 | verbose5 |

Definition at line 275 of file sound4cl.hpp.

#### 13.1.3.2 PresetChange_Kind

```
enum sound4::PresetChange_Kind
```

**Enumerator**

| | |
|---|---|
| change_kind_created | has been created |
| change_kind_modified | has been modified |
| change_kind_deleted | has been deleted |

Definition at line 890 of file sound4cl.hpp.

#### 13.1.3.3 SampleFormat

```
enum sound4::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 60 of file sound4cl.hpp.

### 13.1.4 Function Documentation

#### 13.1.4.1 _log_cb_c()

```
static void sound4::_log_cb_c (
            sound4cl_LogSeverity severity,
            const char * c_msg )  [inline], [static]
```

Definition at line 298 of file sound4cl.hpp.

### 13.1.5 Variable Documentation

#### 13.1.5.1 _log_cb

```
log_cb_t sound4::_log_cb  [static]
```

internal helpers for C log callbacks

Definition at line 296 of file sound4cl.hpp.

## 13.2 sound4::bigvoice Namespace Reference

**Namespaces**

- namespace dyn
- namespace helper

**Classes**

- class CBus

  *Shared Bus.*
- class CInstance

  *Instance handling class.*
- class CPresetLoader

  *Custom preset handler helper.*

**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

**Functions**

- static std::string GetVersion ()
- static unsigned int GetChunkSizeInFrames ()
- static std::vector< unsigned int > GetPossibleChunkSizeInFrames ()
- static unsigned int GetChannelCount ()
- static unsigned int GetAudioInputCount ()
- static unsigned int GetAudioOutputCount ()
- static unsigned int GetSampleRate ()
- static void SanityCheck (bool a_bCheckFrames=true)
- static std::string GetFormatName (const SampleFormat fmt)
- static SampleFormat GetFormatFromName (const std::string &name)
- static unsigned int GetBytesFromFormat (const SampleFormat fmt)
- static void _log_cb_c (bigvoice_LogSeverity severity, const char *c_msg)
- static void SetLogSeverity (LogSeverity severity)
- static void SetLoggerCallback (log_cb_t cb)

**Variables**

- static constexpr const char * process_name = "SOUND4 BIGVOICE.CL"
- static constexpr const char * process_shortname = "bigvoice"
- const size_t ChunkFrames = 12
- const size_t InputChannels = 1 * 1
- const size_t OutputChannels = 1 * 1
- const size_t InputSampleSize =ChunkFrames*InputChannels
- const size_t OutputSampleSize =ChunkFrames*OutputChannels
- const size_t SampleRate = 48000
- static log_cb_t _log_cb

### 13.2.1 Detailed Description

bigvoice namespace

### 13.2.2 Typedef Documentation

#### 13.2.2.1 log_cb_t

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

Log callback type

Definition at line 405 of file sound4.bigvoice.cl.hpp.

#### 13.2.2.2 LogSeverity

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

Log Severity levels

Definition at line 401 of file sound4.bigvoice.cl.hpp.

#### 13.2.2.3 PresetChange_Kind

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

Definition at line 471 of file sound4.bigvoice.cl.hpp.

#### 13.2.2.4 SampleFormat

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

Definition at line 63 of file sound4.bigvoice.cl.hpp.

### 13.2.3 Function Documentation

#### 13.2.3.1 _log_cb_c()

```
static void sound4::bigvoice::_log_cb_c (
            bigvoice_LogSeverity severity,
            const char * c_msg )  [inline], [static]
```

Definition at line 412 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.2 GetAudioInputCount()

```
static unsigned int sound4::bigvoice::GetAudioInputCount ( )  [inline], [static]
```

Get the number of inputs for this process

**Remarks**

All inputs have GetChannelCount() channels

**Returns**

The process input count

Definition at line 311 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.3 GetAudioOutputCount()

```
static unsigned int sound4::bigvoice::GetAudioOutputCount ( )  [inline], [static]
```

Get the number of outputs for this process

**Remarks**

All outputs have GetChannelCount() channels

**Returns**

The process outputs count

Definition at line 323 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.4 GetBytesFromFormat()

```
static unsigned int sound4::bigvoice::GetBytesFromFormat (
            const SampleFormat fmt )  [inline], [static]
```

Get the format byte size

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

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

Definition at line 393 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.5 GetChannelCount()

```
static unsigned int sound4::bigvoice::GetChannelCount ( )  [inline], [static]
```

Get the number of channels needed by this process

**Returns**

The process channels count

Definition at line 299 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.6 GetChunkSizeInFrames()

```
static unsigned int sound4::bigvoice::GetChunkSizeInFrames ( )  [inline], [static]
```

Get the default 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 GetChannelCount samples

**Returns**

The process chunk size in frame

**Deprecated** Use GetPossibleChunkSizeInFrames

Definition at line 272 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.7 GetFormatFromName()

```
static SampleFormat sound4::bigvoice::GetFormatFromName (
            const std::string & name )  [inline], [static]
```

Convert string to format

**Parameters**

| | |
|---|---|
| *name* | the format string |

**Returns**

the matching format, or INVALID_FORMAT if invalid.

Definition at line 382 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.8 GetFormatName()

```
static std::string sound4::bigvoice::GetFormatName (
            const SampleFormat fmt )  [inline], [static]
```

Convert format to string

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

the string name of the format

Definition at line 371 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.9 GetPossibleChunkSizeInFrames()

```
static std::vector< unsigned int > sound4::bigvoice::GetPossibleChunkSizeInFrames ( )  [inline],
[static]
```

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 GetChannelCount samples

**Returns**

An vector of possible process chunk size in frame

Definition at line 285 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.10 GetSampleRate()

```
static unsigned int sound4::bigvoice::GetSampleRate ( )  [inline], [static]
```

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

Definition at line 335 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.11 GetVersion()

```
static std::string sound4::bigvoice::GetVersion ( )  [inline], [static]
```

Get the process version string

**Returns**

version string

Definition at line 257 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.12 SanityCheck()

```
static void sound4::bigvoice::SanityCheck (
            bool a_bCheckFrames = true )  [inline], [static]
```

Checks for library compatibility with const/defines

Throws std::runtime_error on failure

Definition at line 345 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.13 SetLoggerCallback()

```
static void sound4::bigvoice::SetLoggerCallback (
            log_cb_t cb )  [inline], [static]
```

Sets the log callback. If not set, console output will be used.

Definition at line 426 of file sound4.bigvoice.cl.hpp.

#### 13.2.3.14 SetLogSeverity()

```
static void sound4::bigvoice::SetLogSeverity (
            LogSeverity severity )  [inline], [static]
```

Sets the log severity (global)

Definition at line 419 of file sound4.bigvoice.cl.hpp.

### 13.2.4 Variable Documentation

#### 13.2.4.1 _log_cb

```
log_cb_t sound4::bigvoice::_log_cb  [static]
```

internal helpers for C log callbacks

Definition at line 410 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.2 ChunkFrames

```
const size_t sound4::bigvoice::ChunkFrames = 12
```

Number of frames in a chunk

**Deprecated** use GetPossibleChunkSizeInFrames

Definition at line 68 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.3 InputChannels

```
const size_t sound4::bigvoice::InputChannels = 1 * 1
```

Number of float values in an input frame

**Deprecated** Use GetChannelCount and GetAudioInputCount

Definition at line 73 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.4 InputSampleSize

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

Number of float values for an input chunk

**Deprecated** Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Definition at line 83 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.5 OutputChannels

```
const size_t sound4::bigvoice::OutputChannels = 1 * 1
```

Number of float values in an output frame

**Deprecated** Use GetChannelCount and GetAudioOutputCount

Definition at line 78 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.6 OutputSampleSize

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

Number of float values for an output chunk

**Deprecated** Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Definition at line 88 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.7 process_name

```
constexpr const char* sound4::bigvoice::process_name = "SOUND4 BIGVOICE.CL"  [static], [constexpr]
```

Definition at line 61 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.8 process_shortname

```
constexpr const char* sound4::bigvoice::process_shortname = "bigvoice"  [static], [constexpr]
```

Definition at line 62 of file sound4.bigvoice.cl.hpp.

#### 13.2.4.9 SampleRate

```
const size_t sound4::bigvoice::SampleRate = 48000
```

Sample rate of library

**Deprecated** Use GetSampleRate

Definition at line 93 of file sound4.bigvoice.cl.hpp.

## 13.3 sound4::bigvoice::dyn Namespace Reference

### Namespaces

- namespace helper

### Classes

- class CBus

  *Shared Bus.*
- class CDynLib

  *Dynamic library interface.*
- class CInstance

  *Instance handling class.*
- class CPresetLoader

  *Custom preset handler helper.*

### 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

### Functions

- static std::string GetVersion (CDynLib &dynlib)
- static unsigned int GetChunkSizeInFrames (CDynLib &dynlib)
- static std::vector< unsigned int > GetPossibleChunkSizeInFrames (CDynLib &dynlib)
- static unsigned int GetChannelCount (CDynLib &dynlib)
- static unsigned int GetAudioInputCount (CDynLib &dynlib)
- static unsigned int GetAudioOutputCount (CDynLib &dynlib)
- static unsigned int GetSampleRate (CDynLib &dynlib)
- static void SanityCheck (CDynLib &dynlib, bool a_bCheckFrames=true)
- static std::string GetFormatName (CDynLib &dynlib, const SampleFormat fmt)
- static SampleFormat GetFormatFromName (CDynLib &dynlib, const std::string &name)
- static unsigned int GetBytesFromFormat (CDynLib &dynlib, const SampleFormat fmt)
- static void _log_cb_c (bigvoice_LogSeverity severity, const char *c_msg)
- static void SetLogSeverity (CDynLib &dynlib, LogSeverity severity)
- static void SetLoggerCallback (CDynLib &dynlib, log_cb_t cb)

### Variables

- static constexpr const char * process_name = "SOUND4 BIGVOICE.CL"
- static constexpr const char * process_shortname = "bigvoice"
- const size_t ChunkFrames = 12
- const size_t InputChannels = 1 * 1
- const size_t OutputChannels = 1 * 1
- const size_t InputSampleSize =ChunkFrames*InputChannels
- const size_t OutputSampleSize =ChunkFrames*OutputChannels
- const size_t SampleRate = 48000
- static log_cb_t _log_cb

### 13.3.1 Detailed Description

Dynamic loading namespace

### 13.3.2 Typedef Documentation

#### 13.3.2.1 log_cb_t

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

Log callback type

Definition at line 863 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.2.2 LogSeverity

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

Log Severity levels

Definition at line 859 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.2.3 PresetChange_Kind

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

Definition at line 937 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.2.4 SampleFormat

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

Definition at line 497 of file sound4.bigvoice.cl_dyn.hpp.

### 13.3.3 Function Documentation

#### 13.3.3.1 _log_cb_c()

```
static void sound4::bigvoice::dyn::_log_cb_c (
            bigvoice_LogSeverity severity,
            const char * c_msg )  [inline], [static]
```

Definition at line 870 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.2 GetAudioInputCount()

```
static unsigned int sound4::bigvoice::dyn::GetAudioInputCount (
            CDynLib & dynlib )  [inline], [static]
```

Get the number of inputs for this process

**Remarks**

All inputs have GetChannelCount() channels

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

The process input count

Definition at line 762 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.3 GetAudioOutputCount()

```
static unsigned int sound4::bigvoice::dyn::GetAudioOutputCount (
            CDynLib & dynlib )  [inline], [static]
```

Get the number of outputs for this process

**Remarks**

All outputs have GetChannelCount() channels

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

The process outputs count

Definition at line 775 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.4 GetBytesFromFormat()

```
static unsigned int sound4::bigvoice::dyn::GetBytesFromFormat (
            CDynLib & dynlib,
            const SampleFormat fmt )  [inline], [static]
```

Get the format byte size

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *fmt* | the format |

**Returns**

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

Definition at line 851 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.5 GetChannelCount()

```
static unsigned int sound4::bigvoice::dyn::GetChannelCount (
            CDynLib & dynlib )  [inline], [static]
```

Get the number of channels needed by this process

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

The process channels count

Definition at line 749 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.6 GetChunkSizeInFrames()

```
static unsigned int sound4::bigvoice::dyn::GetChunkSizeInFrames (
            CDynLib & dynlib )  [inline], [static]
```

Get the default 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 GetChannelCount samples

**Deprecated** Use GetPossibleChunkSizeInFrames

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

The process chunk size in frame

Definition at line 716 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.7 GetFormatFromName()

```
static SampleFormat sound4::bigvoice::dyn::GetFormatFromName (
            CDynLib & dynlib,
            const std::string & name )  [inline], [static]
```

Convert string to format

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *name* | the format string |

**Returns**

the matching format, or INVALID_FORMAT if invalid.

Definition at line 839 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.8 GetFormatName()

```
static std::string sound4::bigvoice::dyn::GetFormatName (
            CDynLib & dynlib,
            const SampleFormat fmt )  [inline], [static]
```

Convert format to string

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *fmt* | the format |

**Returns**

the string name of the format

Definition at line 827 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.9 GetPossibleChunkSizeInFrames()

```
static std::vector< unsigned int > sound4::bigvoice::dyn::GetPossibleChunkSizeInFrames (
            CDynLib & dynlib )  [inline], [static]
```

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 GetChannelCount samples

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

An vector of possible process chunk size in frame

Definition at line 730 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.10 GetSampleRate()

```
static unsigned int sound4::bigvoice::dyn::GetSampleRate (
            CDynLib & dynlib )  [inline], [static]
```

Get the processing sample-rate

**Remarks**

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

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

The process sample-rate in Hz

Definition at line 788 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.11 GetVersion()

```
static std::string sound4::bigvoice::dyn::GetVersion (
            CDynLib & dynlib )  [inline], [static]
```

Get the process version string

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Returns**

version string

Definition at line 700 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.12 SanityCheck()

```
static void sound4::bigvoice::dyn::SanityCheck (
            CDynLib & dynlib,
            bool a_bCheckFrames = true )  [inline], [static]
```

Checks for library compatibility with const/defines

**Exceptions**

| | |
|---|---|
| *std::runtime_error* | on failure |

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *a_bCheckFrames* | if false, do not check chunk size |

Definition at line 800 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.13 SetLoggerCallback()

```
static void sound4::bigvoice::dyn::SetLoggerCallback (
            CDynLib & dynlib,
            log_cb_t cb )  [inline], [static]
```

Sets the log callback. If not set, console output will be used.

Definition at line 884 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.3.14 SetLogSeverity()

```
static void sound4::bigvoice::dyn::SetLogSeverity (
            CDynLib & dynlib,
            LogSeverity severity )  [inline], [static]
```

Sets the log severity (global)

Definition at line 877 of file sound4.bigvoice.cl_dyn.hpp.

### 13.3.4 Variable Documentation

#### 13.3.4.1 _log_cb

```
log_cb_t sound4::bigvoice::dyn::_log_cb  [static]
```

internal helpers for C log callbacks

Definition at line 868 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.2 ChunkFrames

```
const size_t sound4::bigvoice::dyn::ChunkFrames = 12
```

Number of frames in a chunk

**Deprecated** use GetPossibleChunkSizeInFrames()

Definition at line 502 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.3 InputChannels

```
const size_t sound4::bigvoice::dyn::InputChannels = 1 * 1
```

Number of float values in an input frame

**Deprecated** Use GetChannelCount and GetAudioInputCount

Definition at line 507 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.4 InputSampleSize

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

Number of float values for an input chunk

**Deprecated** Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Definition at line 517 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.5 OutputChannels

```
const size_t sound4::bigvoice::dyn::OutputChannels = 1 * 1
```

Number of float values in an output frame

**Deprecated** Use GetChannelCount and GetAudioOutputCount

Definition at line 512 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.6 OutputSampleSize

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

Number of float values for an output chunk

**Deprecated** Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Definition at line 522 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.7 process_name

```
constexpr const char* sound4::bigvoice::dyn::process_name = "SOUND4 BIGVOICE.CL"  [static],
[constexpr]
```

Definition at line 495 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.8 process_shortname

```
constexpr const char* sound4::bigvoice::dyn::process_shortname = "bigvoice"  [static], [constexpr]
```

Definition at line 496 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.4.9 SampleRate

```
const size_t sound4::bigvoice::dyn::SampleRate = 48000
```

Sample rate of library

**Deprecated** Use GetSampleRate

Definition at line 527 of file sound4.bigvoice.cl_dyn.hpp.

## 13.4 sound4::bigvoice::dyn::helper Namespace Reference

### Classes

- class CDynLoader

  *Helper to load dynamic library.*
- class DynFuncHolder

  *Helper to wrap a function pointer.*
- struct SampleFormat

  *Helper for Sample format types.*
- struct **SampleFormat**< **float** >
- struct **SampleFormat**< **int16_t** >
- struct **SampleFormat**< **int32_t** >

### Functions

- static std::string WStringToUTF8 (const std::wstring &wstr)
- static std::wstring UTF8ToWString (const std::string &str)
- static void AudioConvertFrom (CDynLib &dynlib, const uint8_t *payload, float *output, size_t nSpl, bigvoice_SampleFormat fmt)
- static void AudioConvertTo (CDynLib &dynlib, const float *input, uint8_t *payload, size_t nSpl, bigvoice_SampleFormat fmt)
- static void StereoToMono (CDynLib &dynlib, const float *input, float *output, size_t nFrame)
- static void MonoToStereo (CDynLib &dynlib, const float *input, float *output, size_t nFrame)
- static void StereoToMono_Planar (CDynLib &dynlib, const float *inputL, const float *inputR, float *output, size_t nFrame)
- static void MonoToStereo_Planar (CDynLib &dynlib, const float *input, float *outputL, float *outputR, size_t nFrame)
- static void AudioMonoFromLiveStereo (CDynLib &dynlib, const uint8_t *payload, float *output)
- static void AudioMonoToLiveStereo (CDynLib &dynlib, const float *input, uint8_t *payload)

### 13.4.1 Detailed Description

helper functions for unicode strings in Windows

### 13.4.2 Function Documentation

#### 13.4.2.1 AudioConvertFrom()

```
static void sound4::bigvoice::dyn::helper::AudioConvertFrom (
            CDynLib & dynlib,
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            bigvoice_SampleFormat fmt )  [inline], [static]
```

Convert the payload to floating-point array

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 593 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.2 AudioConvertTo()

```
static void sound4::bigvoice::dyn::helper::AudioConvertTo (
            CDynLib & dynlib,
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            bigvoice_SampleFormat fmt )  [inline], [static]
```

Convert floating-point array to payload

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 608 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.3 AudioMonoFromLiveStereo()

```
static void sound4::bigvoice::dyn::helper::AudioMonoFromLiveStereo (
            CDynLib & dynlib,
            const uint8_t * payload,
            float * output )  [inline], [static]
```

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

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *payload* | Livewire Live Stereo raw buffer |
| *output* | audio output mono buffer (64 byte aligned) |

Definition at line 677 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.4 AudioMonoToLiveStereo()

```
static void sound4::bigvoice::dyn::helper::AudioMonoToLiveStereo (
            CDynLib & dynlib,
            const float * input,
            uint8_t * payload )  [inline], [static]
```

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

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *input* | audio input mono buffer (64 byte aligned) |
| *payload* | Livewire Live Stereo raw buffer |

Definition at line 688 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.5 MonoToStereo()

```
static void sound4::bigvoice::dyn::helper::MonoToStereo (
            CDynLib & dynlib,
```

```
            const float * input,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *input* | audio input mono buffer (64 byte aligned) |
| *output* | audio output stereo buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 636 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.6 MonoToStereo_Planar()

```
static void sound4::bigvoice::dyn::helper::MonoToStereo_Planar (
            CDynLib & dynlib,
            const float * input,
            float * outputL,
            float * outputR,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *input* | audio input mono buffer (64 byte aligned) |
| *outputL* | audio output left buffer (64 byte aligned) |
| *outputR* | audio output right buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 666 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.7 StereoToMono()

```
static void sound4::bigvoice::dyn::helper::StereoToMono (
            CDynLib & dynlib,
```

```
            const float * input,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *input* | audio input stereo buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 622 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.8 StereoToMono_Planar()

```
static void sound4::bigvoice::dyn::helper::StereoToMono_Planar (
            CDynLib & dynlib,
            const float * inputL,
            const float * inputR,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *dynlib* | the dynamic library |
| *inputL* | audio input left buffer (64 byte aligned) |
| *inputR* | audio input right buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 651 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.9 UTF8ToWString()

```
static std::wstring sound4::bigvoice::dyn::helper::UTF8ToWString (
            const std::string & str )  [inline], [static]
```

Helper to convert a std::string (UTF8) to a std::wstring for Windows

**Attention**

For Windows only

Definition at line 551 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.4.2.10 WStringToUTF8()

```
static std::string sound4::bigvoice::dyn::helper::WStringToUTF8 (
            const std::wstring & wstr )  [inline], [static]
```

Helper to convert a std::wstring to a std::string (UTF8) for Windows

**Attention**

For Windows only

Definition at line 539 of file sound4.bigvoice.cl_dyn.hpp.

## 13.5 sound4::bigvoice::helper Namespace Reference

### Classes

- struct SampleFormat

  *Helper for Sample format types.*
- struct **SampleFormat**< **float** >
- struct **SampleFormat**< **int16_t** >
- struct **SampleFormat**< **int32_t** >

### Functions

- static std::string WStringToUTF8 (const std::wstring &wstr)
- static std::wstring UTF8ToWString (const std::string &str)
- static void AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, bigvoice_SampleFormat fmt)
- static void AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, bigvoice_SampleFormat fmt)
- static void StereoToMono (const float *input, float *output, size_t nFrame)
- static void MonoToStereo (const float *input, float *output, size_t nFrame)
- static void StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- static void MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- static void AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- static void AudioMonoToLiveStereo (const float *input, uint8_t *payload)

### 13.5.1 Detailed Description

helper functions for unicode strings in Windows

## 13.5.2 Function Documentation

### 13.5.2.1 AudioConvertFrom()

```
static void sound4::bigvoice::helper::AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            bigvoice_SampleFormat fmt )  [inline], [static]
```

Convert the payload to floating-point array

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 158 of file sound4.bigvoice.cl.hpp.

### 13.5.2.2 AudioConvertTo()

```
static void sound4::bigvoice::helper::AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            bigvoice_SampleFormat fmt )  [inline], [static]
```

Convert floating-point array to payload

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |
| *nSpl* | the number of samples (not frames) |
| *fmt* | the format of the payload |

Definition at line 172 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.3 AudioMonoFromLiveStereo()

```
static void sound4::bigvoice::helper::AudioMonoFromLiveStereo (
            const uint8_t * payload,
            float * output )  [inline], [static]
```

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

**Parameters**

| | |
|---|---|
| *payload* | Livewire Live Stereo raw buffer |
| *output* | audio output mono buffer (64 byte aligned) |

Definition at line 236 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.4 AudioMonoToLiveStereo()

```
static void sound4::bigvoice::helper::AudioMonoToLiveStereo (
            const float * input,
            uint8_t * payload )  [inline], [static]
```

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

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *payload* | Livewire Live Stereo raw buffer |

Definition at line 246 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.5 MonoToStereo()

```
static void sound4::bigvoice::helper::MonoToStereo (
            const float * input,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *output* | audio output stereo buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 198 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.6 MonoToStereo_Planar()

```
static void sound4::bigvoice::helper::MonoToStereo_Planar (
            const float * input,
            float * outputL,
            float * outputR,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input mono buffer (64 byte aligned) |
| *outputL* | audio output left buffer (64 byte aligned) |
| *outputR* | audio output right buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 226 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.7 StereoToMono()

```
static void sound4::bigvoice::helper::StereoToMono (
            const float * input,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input stereo buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 185 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.8 StereoToMono_Planar()

```
static void sound4::bigvoice::helper::StereoToMono_Planar (
            const float * inputL,
            const float * inputR,
            float * output,
            size_t nFrame )  [inline], [static]
```

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

**Warning**

Buffers must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *inputL* | audio input left buffer (64 byte aligned) |
| *inputR* | audio input right buffer (64 byte aligned) |
| *output* | audio output mono buffer (64 byte aligned) |
| *nFrame* | the number of frames |

Definition at line 212 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.9 UTF8ToWString()

```
static std::wstring sound4::bigvoice::helper::UTF8ToWString (
            const std::string & str )  [inline], [static]
```

Helper to convert a std::string (UTF8) to a std::wstring for Windows

**Attention**

For Windows only

Definition at line 117 of file sound4.bigvoice.cl.hpp.

#### 13.5.2.10 WStringToUTF8()

```
static std::string sound4::bigvoice::helper::WStringToUTF8 (
            const std::wstring & wstr )  [inline], [static]
```

Helper to convert a std::wstring to a std::string (UTF8) for Windows

**Attention**

For Windows only

Definition at line 105 of file sound4.bigvoice.cl.hpp.

# 13.6 sound4::helper Namespace Reference

### Classes

- class CDynLoader

  *Helper to load dynamic library.*
- class DynFuncHolder

  *Helper to wrap a function pointer.*
- struct SampleFormat

  *Helper for Sample format types.*
- struct **SampleFormat**< **float** >
- struct **SampleFormat**< **int16_t** >
- struct **SampleFormat**< **int32_t** >

### Functions

- static std::string WStringToUTF8 (const std::wstring &wstr)
- static std::wstring UTF8ToWString (const std::string &str)

## 13.6.1 Detailed Description

helper functions for unicode strings in Windows

## 13.6.2 Function Documentation

### 13.6.2.1 UTF8ToWString()

```
static std::wstring sound4::helper::UTF8ToWString (
            const std::string & str )  [inline], [static]
```

Helper to convert a std::string (UTF8) to a std::wstring for Windows

**Attention**

For Windows only

Definition at line 98 of file sound4cl.hpp.

### 13.6.2.2 WStringToUTF8()

```
static std::string sound4::helper::WStringToUTF8 (
            const std::wstring & wstr )  [inline], [static]
```

Helper to convert a std::wstring to a std::string (UTF8) for Windows

**Attention**

For Windows only

Definition at line 86 of file sound4cl.hpp.

# Chapter 14

# Class Documentation

## 14.1 sound4::AuxiliaryOutputInfos Struct Reference

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

### Public Attributes

- std::string id

  *The ID of the output (fixed across versions)*
- std::string description

  *The description of the output.*
- unsigned int sample_rate

  *The sampling rate of this output.*
- unsigned int channel_count

  *The number of channels of this output.*
- const float * buffer

  *The output buffer of the output. See sound4::bigvoice::CInstance::ProcessAudio for format.*

### 14.1.1 Detailed Description

Definition at line 50 of file sound4.bigvoice.cl.hpp.

### 14.1.2 Member Data Documentation

#### 14.1.2.1 buffer

`const float* sound4::AuxiliaryOutputInfos::buffer`

Definition at line 55 of file sound4.bigvoice.cl.hpp.

#### 14.1.2.2 channel_count

```
unsigned int sound4::AuxiliaryOutputInfos::channel_count
```

Definition at line 54 of file sound4.bigvoice.cl.hpp.

#### 14.1.2.3 description

```
std::string sound4::AuxiliaryOutputInfos::description
```

Definition at line 52 of file sound4.bigvoice.cl.hpp.

#### 14.1.2.4 id

```
std::string sound4::AuxiliaryOutputInfos::id
```

Definition at line 51 of file sound4.bigvoice.cl.hpp.

#### 14.1.2.5 sample_rate

```
unsigned int sound4::AuxiliaryOutputInfos::sample_rate
```

Definition at line 53 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.2 bigvoice_CBus Struct Reference

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

### 14.2.1 Detailed Description

Abstract holder for a instances' bus

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

- sound4.bigvoice.cl.h

## 14.3 bigvoice_CClientInstance Struct Reference

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

### 14.3.1 Detailed Description

Abstract pointer for a client instance

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

- sound4.bigvoice.cl.h

## 14.4 bigvoice_CInstance Struct Reference

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

### 14.4.1 Detailed Description

Abstract holder for a process instance

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

- sound4.bigvoice.cl.h

## 14.5 bigvoice_CParameters Struct Reference

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

### 14.5.1 Detailed Description

Abstract holder for a process instance parameters

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

- sound4.bigvoice.cl.h

## 14.6 sound4::bigvoice::CBus Class Reference

Shared Bus.

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

#### Public Member Functions

- CBus ()
- ~CBus ()
- bigvoice_CBus * Get () const

### 14.6.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 438 of file sound4.bigvoice.cl.hpp.

### 14.6.2 Constructor & Destructor Documentation

#### 14.6.2.1 CBus()

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

Definition at line 440 of file sound4.bigvoice.cl.hpp.

#### 14.6.2.2 ~CBus()

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

Definition at line 442 of file sound4.bigvoice.cl.hpp.

### 14.6.3 Member Function Documentation

#### 14.6.3.1 Get()

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

Definition at line 447 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.7 sound4::bigvoice::dyn::CBus Class Reference

Shared Bus.

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

#### Public Member Functions

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

### 14.7.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 896 of file sound4.bigvoice.cl_dyn.hpp.

### 14.7.2 Constructor & Destructor Documentation

#### 14.7.2.1 CBus()

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

Definition at line 898 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.7.2.2 ~CBus()

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

Definition at line 905 of file sound4.bigvoice.cl_dyn.hpp.

### 14.7.3 Member Function Documentation

#### 14.7.3.1 Get()

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

Definition at line 912 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.8 sound4::CBus Class Reference

Shared Bus.

```
#include <sound4cl.hpp>
```

#### Public Member Functions

- CBus (CProcessor &dynlib)
- ~CBus ()

**Protected Member Functions**

- sound4cl_CBus * Get () const

**Friends**

- class CInstance

### 14.8.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 848 of file sound4cl.hpp.

### 14.8.2 Constructor & Destructor Documentation

#### 14.8.2.1 CBus()

```
sound4::CBus::CBus (
            CProcessor & dynlib )  [inline]
```

Definition at line 850 of file sound4cl.hpp.

#### 14.8.2.2 ~CBus()

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

Definition at line 857 of file sound4cl.hpp.

### 14.8.3 Member Function Documentation

#### 14.8.3.1 Get()

```
sound4cl_CBus * sound4::CBus::Get ( ) const  [inline], [protected]
```

Definition at line 866 of file sound4cl.hpp.

### 14.8.4 Friends And Related Symbol Documentation

#### 14.8.4.1 CInstance

```
friend class CInstance  [friend]
```

Definition at line 865 of file sound4cl.hpp.

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

- sound4cl.hpp

# 14.9 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.*

## 14.9.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 982 of file sound4.bigvoice.cl.hpp.

## 14.9.2 Constructor & Destructor Documentation

### 14.9.2.1 CClient() [1/2]

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

Definition at line 984 of file sound4.bigvoice.cl.hpp.

### 14.9.2.2 CClient() [2/2]

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

### 14.9.2.3 ~CClient()

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

Definition at line 993 of file sound4.bigvoice.cl.hpp.

## 14.9.3 Member Function Documentation

### 14.9.3.1 operator=()

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

### 14.9.3.2 ProcessJson()

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

**Parameters**

| | |
|---|---|
| *request* | the JSON request. See also JSON syntax |
| *NeedSave* | if non null, returns if the saving is required (maybe delayed) |

**Returns**

the JSON answer

Definition at line 1007 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.10 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.*

### 14.10.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1444 of file sound4.bigvoice.cl_dyn.hpp.

### 14.10.2 Constructor & Destructor Documentation

#### 14.10.2.1 CClient() [1/2]

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

Definition at line 1446 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.10.2.2 CClient() [2/2]

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

#### 14.10.2.3 ~CClient()

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

Definition at line 1456 of file sound4.bigvoice.cl_dyn.hpp.

### 14.10.3 Member Function Documentation

#### 14.10.3.1 operator=()

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

#### 14.10.3.2 ProcessJson()

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

**Parameters**

| | |
|---|---|
| *request* | the JSON request. See also JSON syntax |
| *NeedSave* | if non null, returns if the saving is required (maybe delayed) |

**Returns**

the JSON answer

Definition at line 1470 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.11 sound4::CInstance::CClient Class Reference

```
#include <sound4cl.hpp>
```

**Public Member Functions**

- CClient (sound4cl_CInstance *instance, CProcessor &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.*

### 14.11.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1420 of file sound4cl.hpp.

### 14.11.2 Constructor & Destructor Documentation

#### 14.11.2.1 CClient() [1/2]

```
sound4::CInstance::CClient::CClient (
            sound4cl_CInstance * instance,
            CProcessor & dynlib )  [inline]
```

Definition at line 1422 of file sound4cl.hpp.

#### 14.11.2.2 CClient() [2/2]

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

#### 14.11.2.3 ~CClient()

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

Definition at line 1432 of file sound4cl.hpp.

### 14.11.3 Member Function Documentation

#### 14.11.3.1 operator=()

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

#### 14.11.3.2 ProcessJson()

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

**Parameters**

| | |
|---|---|
| *request* | the JSON request. See also JSON syntax |
| *NeedSave* | if non null, returns if the saving is required (maybe delayed) |

**Returns**

the JSON answer

Definition at line 1446 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.12 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={})

  *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
- helper::DynFuncHolder< decltype(bigvoice_SetMetadataMulti) > SetMetadataMulti
- helper::DynFuncHolder< decltype(bigvoice_Webserver_SetAppHealth) > Webserver_SetAppHealth
- helper::DynFuncHolder< decltype(bigvoice_Webserver_GetAppHealth) > Webserver_GetAppHealth
- helper::DynFuncHolder< decltype(bigvoice_Webserver_FreeString) > Webserver_FreeString
- helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracing) > SetInstanceTracing
- helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracingProcessActivity) > SetInstanceTracingProcessActivity

### 14.12.1 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 230 of file sound4.bigvoice.cl_dyn.hpp.

## 14.12.2 Constructor & Destructor Documentation

### 14.12.2.1 CDynLib()

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

## 14.12.3 Member Function Documentation

### 14.12.3.1 IsOk()

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

**Returns**

true if OK.

false if not OK.

Definition at line 487 of file sound4.bigvoice.cl_dyn.hpp.

### 14.12.3.2 Load()

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

**Parameters**

| | |
|---|---|
| *path* | 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 338 of file sound4.bigvoice.cl_dyn.hpp.

## 14.12.4 Member Data Documentation

### 14.12.4.1 AddAudio

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

Definition at line 267 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.2 AddPadAudio

```
helper::DynFuncHolder< decltype(bigvoice_AddPadAudio ) > sound4::bigvoice::dyn::CDynLib::Add↩
PadAudio
```

Definition at line 268 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.3 AudioConvertFrom

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

Definition at line 271 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.4 AudioConvertTo

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

Definition at line 272 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.5 AudioMonoFromLiveStereo

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

Definition at line 275 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.6 AudioMonoToLiveStereo

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

Definition at line 276 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.7 DeleteClient

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

Definition at line 278 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.8 ExitProcess

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

Definition at line 254 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.9 FreeBus

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

Definition at line 311 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.10 FreeJsonAnswer

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

Definition at line 280 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.11 FreeMetadataInfos

```
helper::DynFuncHolder< decltype(bigvoice_FreeMetadataInfos ) > sound4::bigvoice::dyn::CDyn↩
Lib::FreeMetadataInfos
```

Definition at line 297 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.12 FreeParameters

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

Definition at line 247 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.13 FreeParameterValue

```
helper::DynFuncHolder< decltype(bigvoice_FreeParameterValue ) > sound4::bigvoice::dyn::CDyn↩
Lib::FreeParameterValue
```

Definition at line 250 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.14 GetAudio

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

Definition at line 270 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.15 GetAudioInputCount

```
helper::DynFuncHolder< decltype(bigvoice_GetAudioInputCount ) > sound4::bigvoice::dyn::CDyn↩
Lib::GetAudioInputCount
```

Definition at line 241 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.16 GetAudioOutputCount

```
helper::DynFuncHolder< decltype(bigvoice_GetAudioOutputCount ) > sound4::bigvoice::dyn::CDyn↩
Lib::GetAudioOutputCount
```

Definition at line 242 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.17 GetBufferIn

```
helper::DynFuncHolder< decltype(bigvoice_GetBufferIn ) > sound4::bigvoice::dyn::CDynLib::Get↩
BufferIn
```

Definition at line 260 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.18 GetBufferOut

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

Definition at line 261 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.19 GetBytesFromFormat

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

Definition at line 265 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.20 GetChannelCount

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

Definition at line 240 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.21 GetChunkSizeInFrames

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

Definition at line 239 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.22 GetEstimatedDelay

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

Definition at line 262 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.23 GetFormatFromName

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

Definition at line 264 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.24 GetFormatName

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

Definition at line 263 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.25 GetMaxPacketFrame

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

Definition at line 266 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.26 GetMetadataInfos

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

Definition at line 296 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.27 GetOutputCount

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

Definition at line 269 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.28 GetParameter

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

Definition at line 249 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.29 GetPossibleChunkSizeInFrames

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

Definition at line 304 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.30 GetProcessChunkFrames

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

Definition at line 305 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.31 GetSampleRate

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

Definition at line 243 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.32 GetVersion

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

Definition at line 238 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.33 InitProcess

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

Definition at line 251 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.34 InitProcess2

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

Definition at line 252 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.35 InitProcess3

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

Definition at line 306 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.36 MonoToStereo

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

Definition at line 274 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.37 MonoToStereo_Planar

helper::DynFuncHolder< decltype(bigvoice_MonoToStereo_Planar ) > sound4::bigvoice::dyn::CDyn↩
Lib::MonoToStereo_Planar

Definition at line 292 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.38 NewBus

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

Definition at line 310 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.39 NewClient

helper::DynFuncHolder< decltype(bigvoice_NewClient ) > sound4::bigvoice::dyn::CDynLib::New↩
Client

Definition at line 277 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.40 NewParameters

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

Definition at line 246 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.41 PresetManager_InformChange

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

Definition at line 301 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.42 ProcessAudio

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

Definition at line 258 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.43 ProcessAudio_Planar

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

Definition at line 259 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.44 ProcessJson

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

Definition at line 279 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.45 SaveState

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

Definition at line 281 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.46 SetInstanceBus

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

Definition at line 312 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.47 SetInstanceTracing

```
helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracing ) > sound4::bigvoice::dyn::CDyn↩
Lib::SetInstanceTracing
```

Definition at line 327 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.48 SetInstanceTracingProcessActivity

```
helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracingProcessActivity ) > sound4::bigvoice↩
::dyn::CDynLib::SetInstanceTracingProcessActivity
```

Definition at line 328 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.49 SetLoggerCallback

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

Definition at line 244 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.50 SetLogSeverity

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

Definition at line 245 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.51 SetMetadata

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

Definition at line 295 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.52 SetMetadataMulti

helper::DynFuncHolder< decltype(bigvoice_SetMetadataMulti ) > sound4::bigvoice::dyn::CDynLib↩
::SetMetadataMulti

Definition at line 316 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.53 SetParameter

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

Definition at line 248 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.54 SetPresetManager

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

Definition at line 300 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.55 StartUpdateThread

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

Definition at line 255 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.56 StereoToMono

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

Definition at line 273 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.57 StereoToMono_Planar

helper::DynFuncHolder< decltype(bigvoice_StereoToMono_Planar ) > sound4::bigvoice::dyn::CDyn↩
Lib::StereoToMono_Planar

Definition at line 291 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.58 StopUpdateThread

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

Definition at line 256 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.59 TerminateProcess

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

Definition at line 253 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.60 WaitUpdateThreadReady

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

Definition at line 257 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.61 Webserver

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

Definition at line 285 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.62 Webserver_FreeString

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_FreeString ) > sound4::bigvoice::dyn::↩
CDynLib::Webserver_FreeString
```

Definition at line 322 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.63 Webserver_GetAppHealth

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_GetAppHealth ) > sound4::bigvoice::dyn::↩
CDynLib::Webserver_GetAppHealth
```

Definition at line 321 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.64 Webserver_SetAppHealth

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_SetAppHealth ) > sound4::bigvoice::dyn::↩
CDynLib::Webserver_SetAppHealth
```

Definition at line 320 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.65 Webserver_Status

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

Definition at line 287 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.66 Webserver_Stop

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

Definition at line 286 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.67 Webserver_tcp

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

Definition at line 283 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.12.4.68 Webserver_tcp2

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

Definition at line 284 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.13 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_LIBRAF
- 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)

### 14.13.1 Detailed Description

Definition at line 108 of file sound4.bigvoice.cl_dyn.hpp.

### 14.13.2 Constructor & Destructor Documentation

#### 14.13.2.1 CDynLoader() [1/3]

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

#### 14.13.2.2 CDynLoader() [2/3]

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

#### 14.13.2.3 CDynLoader() [3/3]

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

#### 14.13.2.4 ~CDynLoader()

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

Definition at line 218 of file sound4.bigvoice.cl_dyn.hpp.

### 14.13.3 Member Function Documentation

#### 14.13.3.1 Close()

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

Definition at line 179 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.2 GetSymbol()

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

Definition at line 192 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.3 GetSymbol_safe()

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

Definition at line 200 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.4 GetThisLibraryPath()

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

Definition at line 204 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.5 IsOk()

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

Definition at line 126 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.6 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_LIB
)  [inline]
```

Definition at line 173 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.7 operator bool()

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

Definition at line 123 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.13.3.8 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

## 14.14 sound4::helper::CDynLoader Class Reference

Helper to load dynamic library.

`#include <sound4cl.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_LIBRAF
- 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)

### 14.14.1 Detailed Description

Definition at line 155 of file sound4cl.hpp.

### 14.14.2 Constructor & Destructor Documentation

#### 14.14.2.1 CDynLoader() [1/3]

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

#### 14.14.2.2 CDynLoader() [2/3]

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

#### 14.14.2.3 CDynLoader() [3/3]

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

#### 14.14.2.4 ~CDynLoader()

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

Definition at line 265 of file sound4cl.hpp.

### 14.14.3 Member Function Documentation

#### 14.14.3.1 Close()

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

Definition at line 226 of file sound4cl.hpp.

#### 14.14.3.2 GetSymbol()

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

Definition at line 239 of file sound4cl.hpp.

#### 14.14.3.3 GetSymbol_safe()

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

Definition at line 247 of file sound4cl.hpp.

#### 14.14.3.4 GetThisLibraryPath()

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

Definition at line 251 of file sound4cl.hpp.

#### 14.14.3.5 IsOk()

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

Definition at line 173 of file sound4cl.hpp.

#### 14.14.3.6 Load()

```
bool sound4::helper::CDynLoader::Load (
            const std::filesystem::path & dynlib,
            uint32_t loadflags = LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR|LOAD_LIBRARY_SEARCH_APPLICATION_DIR|LOAD_LIB
)  [inline]
```

Definition at line 220 of file sound4cl.hpp.

#### 14.14.3.7 operator bool()

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

Definition at line 170 of file sound4cl.hpp.

#### 14.14.3.8 operator=()

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

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

- sound4cl.hpp

## 14.15 sound4::bigvoice::CInstance Class Reference

Instance handling class.

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

### Classes

- class CClient

### Public Member Functions

- CInstance ()
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- virtual ~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 &Access_Key_ID, const std::string &Access_Key_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)
- void SetMetadataMulti (const std::unordered_map< std::string, const char * > &list)
- std::vector< std::tuple< std::string, std::string > > GetMetadataInfos ()
- std::vector< AuxiliaryOutputInfos > GetAuxiliaryAudioOutputInfos ()
- void ActivateAuxiliaryAudioID (const std::string &id)
- virtual void OnUpdateThreadStart ()
- virtual void OnUpdateThreadStop ()
- bool StartWebServer (int http_port, int https_port=0)
- void StopWebServer (int timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- 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 ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

### 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)

## 14.15.1 Detailed Description

Holds an instance of processing

Definition at line 478 of file sound4.bigvoice.cl.hpp.

## 14.15.2 Constructor & Destructor Documentation

### 14.15.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 483 of file sound4.bigvoice.cl.hpp.

### 14.15.2.2 CInstance() [2/2]

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

### 14.15.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 494 of file sound4.bigvoice.cl.hpp.

## 14.15.3 Member Function Documentation

### 14.15.3.1 ActivateAuxiliaryAudioID()

```
void sound4::bigvoice::CInstance::ActivateAuxiliaryAudioID (
            const std::string & id )  [inline]
```

Definition at line 755 of file sound4.bigvoice.cl.hpp.

### 14.15.3.2 AddAudio()

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

Definition at line 1076 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.3 Create()

```
bool sound4::bigvoice::CInstance::Create (
            const std::string & LoginKey,
            const std::string & RadioName,
            const std::string & Access_Key_ID,
            const std::string & Access_Key_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

**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 empty or the folder is empty, a default configuration with a default preset will be loaded, unless using SetPresetManager.

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

**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**

| | |
|---|---|
| *LoginKey* | SOUND4 Login key |
| *RadioName* | SOUND4 RadioName |
| *Access_Key_ID* | SOUND4 AWS access ID |
| *Access_Key_Secret* | SOUND4 AWS access Secret |
| *save_path* | Path where we can load/save the configuration |
| *json_port* | Optional json listening port |
| *frames_per_chunk* | 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 597 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.4 GetAudio()

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

Definition at line 1082 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.5 GetAuxiliaryAudioOutputInfos()

```
std::vector< AuxiliaryOutputInfos > sound4::bigvoice::CInstance::GetAuxiliaryAudioOutputInfos
( )  [inline]
```

Get information on auxiliary audio outputs.

**Returns**

a list of AuxiliaryOutputInfos

Definition at line 736 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.6 GetBufferIn()

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

Get preallocated aligned buffer for audio input for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio input dedicated buffer.

Definition at line 897 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.7 GetBufferOut()

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

Get preallocated aligned buffer for audio output for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio output dedicated buffer.

Definition at line 915 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.8 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 634 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.9 GetEstimatedDelay()

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

Get the delay (estimated) of current processing algorithm

**Remarks**

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

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

**Returns**

Estimated delay in frames

Definition at line 879 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.10 GetMetadataInfos()

```
std::vector< std::tuple< std::string, std::string > > sound4::bigvoice::CInstance::Get↩
MetadataInfos ( )  [inline]
```

Get information on known metadata keys.

**Returns**

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

Definition at line 712 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.11 GetParam()

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

Gets a parameter value

**Parameters**

| | |
|---|---|
| *name* | the parameter name |

**Returns**

the parameter value

Definition at line 523 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.12 GetWebServerAppHealth()

```
void sound4::bigvoice::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 826 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.13 IsOk()

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

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

Definition at line 502 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.14 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 1030 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.15 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 762 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.16 OnUpdateThreadStop()

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

Can be overloaded to call at end of UpdateThread in the thread

Definition at line 766 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.17 operator=()

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

#### 14.15.3.18 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 conjonction with SetPresetManager

**Parameters**

| | |
|---|---|
| *relative_path* | The path which changed, relative to the root storage |
| *change_kind* | What happened to this path |

Definition at line 648 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.19 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**

| | |
|---|---|
| *T* | the C type for one audio sample |

**Parameters**

| | |
|---|---|
| *input* | audio to process |

**Returns**

processed audio

Definition at line 958 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.20 ProcessAudio()

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

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 924 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.21 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...], ... ]

**Parameters**

| | |
|---|---|
| *input* | has GetAudioInputCount() * GetChannelCount() channels |
| *output* | has GetAudioOutputCount() * GetChannelCount() channels |

Definition at line 940 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.22 SaveState()

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

Save the current state

Needed when CClient::ProcessJson returned with need_save to true. However it can be postponed as much as wanted.

**Remarks**

Works only when update thread is running

**Returns**

true if success

Definition at line 1046 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.23 SetBus()

```
void sound4::bigvoice::CInstance::SetBus (
            const CBus & bus )  [inline]
```

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**

| | |
|---|---|
| *bus* | the bus to use |

Definition at line 541 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.24 SetInstanceTracing()

```
void sound4::bigvoice::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid, used for creation, updates and destruction |
| *tracing_provider* | the provider as created with TRACELOGGING_DEFINE_PROVIDER |

Definition at line 1060 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.25 SetInstanceTracingProcessActivity()

```
void sound4::bigvoice::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

Set the instance's process activity for tracing (optional). Should be called only in process's thread.

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid |

Definition at line 1069 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.26 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**

| | |
|---|---|
| *key* | 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. |
| *value* | the metadata value or nullptr to erase it. |

Definition at line 664 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.27 SetMetadataMulti()

```
void sound4::bigvoice::CInstance::SetMetadataMulti (
            const std::unordered_map< std::string, const char * > & list )  [inline]
```

Adds multiple metadata to the process.

This works like SetMetadata but with multiple metadata at once.

**See also**

CInstance::SetMetadata

**Parameters**

| | |
|---|---|
| *list* | a map of key/value pair. |

Definition at line 682 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.28 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 Environment variables and parameters

**Parameters**

| | |
|---|---|
| *name* | the parameter name |
| *value* | the parameter value |

Definition at line 512 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.29 SetPresetManager()

```
void sound4::bigvoice::CInstance::SetPresetManager (
            CPresetLoader * preset_manager )  [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in Create

To dynamically update when storage change, see PresetManager_InformChange

Must be done before Create

**Parameters**

| | |
|---|---|
| *preset_manager* | The custom preset manager |

Definition at line 557 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.30 SetWebServerAppHealth()

```
void sound4::bigvoice::CInstance::SetWebServerAppHealth (
            int httpcode,
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 818 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.31 StartWebServer()

```
bool sound4::bigvoice::CInstance::StartWebServer (
            int http_port,
            int https_port = 0 )  [inline]
```

Starts a Web server on requested ports

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**

| | |
|---|---|
| *http_port* | The port to listen for HTTP connections (0 to disable) |
| *https_port* | The port to listen for HTTPS connections (0 to disable) |

**Returns**

true if ports were opened successfully.

Definition at line 782 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.32 Stop()

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

Stops the instance, its web server and its update thread.

Definition at line 856 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.33 StopWebServer()

```
void sound4::bigvoice::CInstance::StopWebServer (
            int timeout_ms = 1000 )  [inline]
```

Stop a web server started with StartWebServer

**Warning**

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

**Parameters**

| | |
|---|---|
| *timeout_ms* | The timeout to wait gracefull shutdown |

Definition at line 807 of file sound4.bigvoice.cl.hpp.

#### 14.15.3.34 Terminate()

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

Terminate the processing.

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

**Returns**

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

Definition at line 847 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

# 14.16 sound4::bigvoice::dyn::CInstance Class Reference

Instance handling class.

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

### Classes

- class CClient

### Public Member Functions

- CInstance (CDynLib &dynlib)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- virtual ~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 &Access_Key_ID, const std::string &Access_Key_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)
- void SetMetadataMulti (const std::unordered_map< std::string, const char * > &list)
- 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 timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- 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 ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

**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)

## 14.16.1 Detailed Description

Holds an instance of processing

Definition at line 944 of file sound4.bigvoice.cl_dyn.hpp.

## 14.16.2 Constructor & Destructor Documentation

### 14.16.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 949 of file sound4.bigvoice.cl_dyn.hpp.

### 14.16.2.2 CInstance() [2/2]

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

### 14.16.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 961 of file sound4.bigvoice.cl_dyn.hpp.

## 14.16.3 Member Function Documentation

### 14.16.3.1 AddAudio()

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

Definition at line 1544 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.2 Create()

```
bool sound4::bigvoice::dyn::CInstance::Create (
            const std::string & LoginKey,
            const std::string & RadioName,
            const std::string & Access_Key_ID,
            const std::string & Access_Key_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

**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 empty or the folder is empty, a default configuration with a default preset will be loaded, unless using SetPresetManager.

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

**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**

| | |
|---|---|
| *LoginKey* | SOUND4 Login key |
| *RadioName* | SOUND4 RadioName |
| *Access_Key_ID* | SOUND4 AWS access ID |
| *Access_Key_Secret* | SOUND4 AWS access Secret |
| *save_path* | Path where we can load/save the configuration |
| *json_port* | Optional json listening port |
| *frames_per_chunk* | 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 1066 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.3 GetAudio()

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

Definition at line 1550 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.4 GetBufferIn()

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

Get preallocated aligned buffer for audio input for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio input dedicated buffer.

Definition at line 1359 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.5 GetBufferOut()

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

Get preallocated aligned buffer for audio output for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio output dedicated buffer.

Definition at line 1377 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.6 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 1109 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.7 GetEstimatedDelay()

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

Get the delay (estimated) of current processing algorithm

**Remarks**

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

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

**Returns**

Estimated delay in frames

Definition at line 1341 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.8 GetMetadataInfos()

```
std::vector< std::tuple< std::string, std::string > > sound4::bigvoice::dyn::CInstance::Get↩
MetadataInfos ( )  [inline]
```

Get information on known metadata keys.

**Returns**

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

Definition at line 1198 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.9 GetParam()

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

Gets a parameter value

**Parameters**

| | |
|---|---|
| *name* | the parameter name |

**Returns**

the parameter value

Definition at line 990 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.10 GetWebServerAppHealth()

```
void sound4::bigvoice::dyn::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 1287 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.11 IsOk()

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

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

Definition at line 969 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.12 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 1494 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.13 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 1222 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.14 OnUpdateThreadStop()

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

Can be overloaded to call at end of UpdateThread in the thread

Definition at line 1226 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.15 operator=()

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

#### 14.16.3.16 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 conjonction with SetPresetManager

**Parameters**

| | |
|---|---|
| *relative_path* | The path which changed, relative to the root storage |
| *change_kind* | What happened to this path |

Definition at line 1123 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.17 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**

| | |
|---|---|
| *T* | the C type for one audio sample |

**Parameters**

| | |
|---|---|
| *input* | audio to process |

**Returns**

processed audio

Definition at line 1420 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.18 ProcessAudio()

```
void sound4::bigvoice::dyn::CInstance::ProcessAudio ( )  [inline]
```

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 1386 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.19 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...], ... ]

**Parameters**

| | |
|---|---|
| *input* | has GetAudioInputCount() * GetChannelCount() channels |
| *output* | has GetAudioOutputCount() * GetChannelCount() channels |

Definition at line 1402 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.20 SaveState()

```
bool sound4::bigvoice::dyn::CInstance::SaveState ( )  [inline]
```

Save the current state

Needed when CClient::ProcessJson returned with need_save to true. However it can be postponed as much as wanted.

**Remarks**

Works only when update thread is running

**Returns**

true if success

Definition at line 1510 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.21 SetBus()

```
void sound4::bigvoice::dyn::CInstance::SetBus (
            const CBus & bus )  [inline]
```

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**

| | |
|---|---|
| *bus* | the bus to use |

Definition at line 1007 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.22 SetInstanceTracing()

```
void sound4::bigvoice::dyn::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid, used for creation, updates and destruction |
| *tracing_provider* | the provider as created with TRACELOGGING_DEFINE_PROVIDER |

Definition at line 1524 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.23 SetInstanceTracingProcessActivity()

```
void sound4::bigvoice::dyn::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

Set the instance's process activity for tracing (optional). Should be called only in process's thread.

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid |

Definition at line 1535 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.24 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**

| | |
|---|---|
| *key* | 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. |
| *value* | the metadata value or nullptr to erase it. |

Definition at line 1141 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.25 SetMetadataMulti()

```
void sound4::bigvoice::dyn::CInstance::SetMetadataMulti (
            const std::unordered_map< std::string, const char * > & list )  [inline]
```

Adds multiple metadata to the process.

This works like SetMetadata but with multiple metadata at once.

**See also**

CInstance::SetMetadata

**Parameters**

| | |
|---|---|
| *list* | a map of key/value pair. |

Definition at line 1162 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.26 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 Environment variables and parameters

**Parameters**

| | |
|---|---|
| *name* | the parameter name |
| *value* | the parameter value |

Definition at line 979 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.27 SetPresetManager()

```
void sound4::bigvoice::dyn::CInstance::SetPresetManager (
            CPresetLoader * preset_manager )  [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in Create

To dynamically update when storage change, see PresetManager_InformChange

Must be done before Create

**Parameters**

| | |
|---|---|
| *preset_manager* | The custom preset manager |

Definition at line 1025 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.28 SetWebServerAppHealth()

```
void sound4::bigvoice::dyn::CInstance::SetWebServerAppHealth (
            int httpcode,
```

```
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 1278 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.29 StartWebServer()

```
bool sound4::bigvoice::dyn::CInstance::StartWebServer (
            int http_port,
            int https_port = 0 )  [inline]
```

Starts a Web server on requested ports

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**

| | |
|---|---|
| *http_port* | The port to listen for HTTP connections (0 to disable) |
| *https_port* | The port to listen for HTTPS connections (0 to disable) |

**Returns**

true if ports were opened successfully.

Definition at line 1242 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.30 Stop()

```
void sound4::bigvoice::dyn::CInstance::Stop ( )  [inline]
```

Stops the instance, its web server and its update thread.

Definition at line 1318 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.31 StopWebServer()

```
void sound4::bigvoice::dyn::CInstance::StopWebServer (
            int timeout_ms = 1000 )  [inline]
```

Stop a web server started with StartWebServer

**Warning**

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

**Parameters**

| | |
|---|---|
| *timeout_ms* | The timeout to wait gracefull shutdown |

Definition at line 1267 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.32 Terminate()

```
int sound4::bigvoice::dyn::CInstance::Terminate ( )  [inline]
```

Terminate the processing.

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

**Returns**

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

Definition at line 1309 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.17 sound4::CInstance Class Reference

Instance handling class.

```
#include <sound4cl.hpp>
```

### Classes

- class CClient

### Public Member Functions

- CInstance (CProcessor &dynlib)
- CInstance (CProcessor &dynlib, void *_instance)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- virtual ~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 &Access_Key_ID, const std::string &Access_Key_Secret, const std::filesystem::path &save_path, int json_port=0, unsigned int frames_per_chunk=0)
- 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)
- void SetMetadataMulti (const std::unordered_map< std::string, const char * > &list)
- 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 timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- int Terminate ()
- void Stop ()
- unsigned int GetEstimatedDelay ()
- float * GetBufferIn ()
- float * 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 ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

### 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)

## 14.17.1 Detailed Description

Holds an instance of processing

Definition at line 900 of file sound4cl.hpp.

## 14.17.2 Constructor & Destructor Documentation

### 14.17.2.1 CInstance() [1/3]

```
sound4::CInstance::CInstance (
            CProcessor & dynlib )  [inline]
```

Creates the instance, with default parameters

**Parameters**

| | |
|---|---|
| *dynlib* | the processor dynamic library |

Definition at line 907 of file sound4cl.hpp.

### 14.17.2.2 CInstance() [2/3]

```
sound4::CInstance::CInstance (
            CProcessor & dynlib,
            void * _instance )  [inline]
```

Wrap an already created C instance.

**Note**

Stop will not stop the instance.

**Parameters**

| | |
|---|---|
| *dynlib* | the processor dynamic library |
| *_instance* | the C instance pointer (No check on the pointer is made.) |

Definition at line 922 of file sound4cl.hpp.

### 14.17.2.3 CInstance() [3/3]

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

### 14.17.2.4 ~CInstance()

```
virtual sound4::CInstance::~CInstance ( )  [inline], [virtual]
```

Destructor, stopping any running tasks

Definition at line 934 of file sound4cl.hpp.

## 14.17.3 Member Function Documentation

### 14.17.3.1 AddAudio()

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

Definition at line 1520 of file sound4cl.hpp.

### 14.17.3.2 Create()

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

Creates and initialize the instance, and starts its update thread

#### 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 empty or the folder is empty, a default configuration with a default preset will be loaded, unless using SetPresetManager.

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

#### 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

| | |
|---|---|
| *LoginKey* | SOUND4 Login key |
| *RadioName* | SOUND4 RadioName |
| *Access_Key_ID* | SOUND4 AWS access ID |
| *Access_Key_Secret* | SOUND4 AWS access Secret |
| *save_path* | Path where we can load/save the configuration |
| *json_port* | Optional json listening port |
| *frames_per_chunk* | The number of audio frames in each chunk of processing. Must be in the list returned by CProcessor::GetChunkSizeInFrames, or 0 to set default |

**Returns**

false if failed

Definition at line 1038 of file sound4cl.hpp.

#### 14.17.3.3 GetAudio()

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

Definition at line 1526 of file sound4cl.hpp.

#### 14.17.3.4 GetBufferIn()

```
float * sound4::CInstance::GetBufferIn ( )  [inline]
```

Get preallocated aligned buffer for audio input for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio input dedicated buffer.

Definition at line 1337 of file sound4cl.hpp.

#### 14.17.3.5 GetBufferOut()

```
float * sound4::CInstance::GetBufferOut ( )  [inline]
```

Get preallocated aligned buffer for audio output for ProcessAudio

The buffer is a big enough for all supported chunk sizes.

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]

**Returns**

An audio output dedicated buffer.

Definition at line 1354 of file sound4cl.hpp.

#### 14.17.3.6 GetChunkFrames()

```
unsigned int sound4::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 1084 of file sound4cl.hpp.

#### 14.17.3.7 GetEstimatedDelay()

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

Get the delay (estimated) of current processing algorithm

**Remarks**

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

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

**Returns**

Estimated delay in frames

Definition at line 1319 of file sound4cl.hpp.

#### 14.17.3.8 GetMetadataInfos()

```
std::vector< std::tuple< std::string, std::string > > sound4::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 1172 of file sound4cl.hpp.

#### 14.17.3.9 GetParam()

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

Gets a parameter value

**Parameters**

| | |
|---|---|
| *name* | the parameter name |

**Returns**

the parameter value

Definition at line 963 of file sound4cl.hpp.

#### 14.17.3.10 GetWebServerAppHealth()

```
void sound4::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 1265 of file sound4cl.hpp.

#### 14.17.3.11 IsOk()

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

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

Definition at line 942 of file sound4cl.hpp.

#### 14.17.3.12 NewClient()

```
std::shared_ptr< CClient > sound4::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 1470 of file sound4cl.hpp.

#### 14.17.3.13 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 1196 of file sound4cl.hpp.

#### 14.17.3.14 OnUpdateThreadStop()

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

Can be overloaded to call at end of UpdateThread in the thread

Definition at line 1200 of file sound4cl.hpp.

#### 14.17.3.15 operator=()

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

#### 14.17.3.16 PresetManager_InformChange()

```
void sound4::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 conjonction with SetPresetManager

**Parameters**

| | |
|---|---|
| *relative_path* | The path which changed, relative to the root storage |
| *change_kind* | What happened to this path |

Definition at line 1098 of file sound4cl.hpp.

#### 14.17.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector< T > sound4::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

**Template Parameters**

| | |
|---|---|
| *T* | the C type for one audio sample |

**Parameters**

| | |
|---|---|
| *input* | audio to process |

**Returns**

processed audio

Definition at line 1396 of file sound4cl.hpp.

#### 14.17.3.18 ProcessAudio()

```
void sound4::CInstance::ProcessAudio ( )  [inline]
```

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 1362 of file sound4cl.hpp.

#### 14.17.3.19 ProcessAudio_Planar()

```
void sound4::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...], ... ]

**Parameters**

| | |
|---|---|
| *input* | has GetAudioInputCount() * GetChannelCount() channels |
| *output* | has GetAudioOutputCount() * GetChannelCount() channels |

Definition at line 1378 of file sound4cl.hpp.

#### 14.17.3.20 SaveState()

```
bool sound4::CInstance::SaveState ( )  [inline]
```

Save the current state

Needed when CClient::ProcessJson returned with need_save to true. However it can be postponed as much as wanted.

**Remarks**

Works only when update thread is running

**Returns**

true if success

Definition at line 1486 of file sound4cl.hpp.

#### 14.17.3.21 SetBus()

```
void sound4::CInstance::SetBus (
            const CBus & bus )  [inline]
```

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**

| | |
|---|---|
| *bus* | the bus to use |

Definition at line 980 of file sound4cl.hpp.

#### 14.17.3.22 SetInstanceTracing()

```
void sound4::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid, used for creation, updates and destruction |
| *tracing_provider* | the provider as created with TRACELOGGING_DEFINE_PROVIDER |

Definition at line 1500 of file sound4cl.hpp.

#### 14.17.3.23 SetInstanceTracingProcessActivity()

```
void sound4::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

Set the instance's process activity for tracing (optional). Should be called only in process's thread.

**Parameters**

| | |
|---|---|
| *activity_guid* | the activity guid |

Definition at line 1511 of file sound4cl.hpp.

#### 14.17.3.24 SetMetadata()

```
void sound4::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**

| | |
|---|---|
| *key* | 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. |
| *value* | the metadata value or nullptr to erase it. |

Definition at line 1116 of file sound4cl.hpp.

#### 14.17.3.25 SetMetadataMulti()

```
void sound4::CInstance::SetMetadataMulti (
            const std::unordered_map< std::string, const char * > & list )  [inline]
```

Adds multiple metadata to the process.

This works like SetMetadata but with multiple metadata at once.

**See also**

CInstance::SetMetadata

**Parameters**

| | |
|---|---|
| *list* | a map of key/value pair. |

Definition at line 1136 of file sound4cl.hpp.

#### 14.17.3.26 SetParam()

```
void sound4::CInstance::SetParam (
            const std::string & name,
            const std::string & value )  [inline]
```

Set a parameter value

Must be done before Create See also Environment variables and parameters

**Parameters**

| | |
|---|---|
| *name* | the parameter name |
| *value* | the parameter value |

Definition at line 952 of file sound4cl.hpp.

#### 14.17.3.27 SetPresetManager()

```
void sound4::CInstance::SetPresetManager (
            CPresetLoader * preset_manager )  [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in Create

To dynamically update when storage change, see PresetManager_InformChange

Must be done before Create

**Parameters**

| | |
|---|---|
| *preset_manager* | The custom preset manager |

Definition at line 997 of file sound4cl.hpp.

#### 14.17.3.28 SetWebServerAppHealth()

```
void sound4::CInstance::SetWebServerAppHealth (
            int httpcode,
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 1256 of file sound4cl.hpp.

#### 14.17.3.29 StartWebServer()

```
bool sound4::CInstance::StartWebServer (
            int http_port,
            int https_port = 0 )  [inline]
```

Starts a Web server on requested ports

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**

| | |
|---|---|
| *http_port* | The port to listen for HTTP connections (0 to disable) |
| *https_port* | The port to listen for HTTPS connections (0 to disable) |

**Returns**

true if ports were opened successfully.

Definition at line 1216 of file sound4cl.hpp.

#### 14.17.3.30 Stop()

```
void sound4::CInstance::Stop ( )  [inline]
```

Stops the instance, its web server and its update thread.

**Note**

Stop will not stop a wrapped C instance

Definition at line 1296 of file sound4cl.hpp.

#### 14.17.3.31 StopWebServer()

```
void sound4::CInstance::StopWebServer (
            int timeout_ms = 1000 )  [inline]
```

Stop a web server started with StartWebServer

**Warning**

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

**Parameters**

| | |
|---|---|
| *timeout_ms* | The timeout to wait gracefull shutdown |

Definition at line 1244 of file sound4cl.hpp.

#### 14.17.3.32 Terminate()

```
int sound4::CInstance::Terminate ( )  [inline]
```

Terminate the processing.

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

**Returns**

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

Definition at line 1285 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.18 sound4::bigvoice::CPresetLoader Class Reference

Custom preset handler helper.

`#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

### 14.18.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 458 of file sound4.bigvoice.cl.hpp.

### 14.18.2 Constructor & Destructor Documentation

#### 14.18.2.1 CPresetLoader()

```
sound4::bigvoice::CPresetLoader::CPresetLoader ( )  [default]
```

#### 14.18.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.18.3 Member Function Documentation

#### 14.18.3.1 Exists()

```
virtual bool sound4::bigvoice::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

#### 14.18.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::bigvoice::CPresetLoader::GetAll ( )
[pure virtual]
```

#### 14.18.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

#### 14.18.3.4 Read()

```
virtual std::string sound4::bigvoice::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

#### 14.18.3.5 Remove()

```
virtual bool sound4::bigvoice::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

#### 14.18.3.6 Rename()

```
virtual bool sound4::bigvoice::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

#### 14.18.3.7 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:

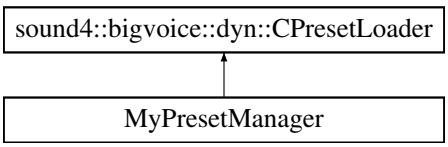
- sound4.bigvoice.cl.hpp

## 14.19 sound4::bigvoice::dyn::CPresetLoader Class Reference

Custom preset handler helper.

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

Inheritance diagram for sound4::bigvoice::dyn::CPresetLoader:




### 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

### 14.19.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 924 of file sound4.bigvoice.cl_dyn.hpp.

### 14.19.2 Constructor & Destructor Documentation

#### 14.19.2.1 CPresetLoader()

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

#### 14.19.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::dyn::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.19.3 Member Function Documentation

#### 14.19.3.1 Exists()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::bigvoice::dyn::CPresetLoader::GetAll ( )
[pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.4 Read()

```
virtual std::string sound4::bigvoice::dyn::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.5 Remove()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.6 Rename()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.19.3.7 Write()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

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

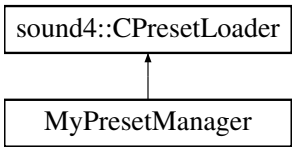
- sound4.bigvoice.cl_dyn.hpp

## 14.20 sound4::CPresetLoader Class Reference

Custom preset handler helper.

`#include <sound4cl.hpp>`

Inheritance diagram for sound4::CPresetLoader:




### 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

### 14.20.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 877 of file sound4cl.hpp.

### 14.20.2 Constructor & Destructor Documentation

#### 14.20.2.1 CPresetLoader()

```
sound4::CPresetLoader::CPresetLoader ( )  [default]
```

#### 14.20.2.2 ~CPresetLoader()

```
virtual sound4::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.20.3 Member Function Documentation

#### 14.20.3.1 Exists()

```
virtual bool sound4::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::CPresetLoader::GetAll ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.3 IsReadOnly()

```
virtual bool sound4::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.4 Read()

```
virtual std::string sound4::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.5 Remove()

```
virtual bool sound4::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.6 Rename()

```
virtual bool sound4::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.20.3.7 Write()

```
virtual bool sound4::CPresetLoader::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

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

- sound4cl.hpp

## 14.21 sound4::CProcessor Class Reference

Dynamic library interface.

`#include <sound4cl.hpp>`

### Public Member Functions

- CProcessor ()=default
- bool Load (const std::filesystem::path &filepath)

  *Loads the library.*
- bool IsOk () const

  *Check if the library was loaded correctly.*
- void AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
- void AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
- void StereoToMono (const float *input, float *output, size_t nFrame)
- void MonoToStereo (const float *input, float *output, size_t nFrame)
- void StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- void AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- std::string GetVersion ()
- unsigned int GetChunkSizeInFrames ()
- std::vector< unsigned int > GetPossibleChunkSizeInFrames ()
- unsigned int GetChannelCount ()
- unsigned int GetAudioInputCount ()
- unsigned int GetAudioOutputCount ()
- unsigned int GetSampleRate ()
- std::string GetFormatName (const SampleFormat fmt)
- SampleFormat GetFormatFromName (const std::string &name)
- unsigned int GetBytesFromFormat (const SampleFormat fmt)
- void SetLogSeverity (LogSeverity severity)
- void SetLoggerCallback (log_cb_t cb)

### Protected Member Functions

- template<typename T >
  helper::DynFuncHolder< T > GetPrefixSymbol (const std::string &name)

### Friends

- class CBus
- class CInstance

### 14.21.1 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 309 of file sound4cl.hpp.

## 14.21.2 Constructor & Destructor Documentation

### 14.21.2.1 CProcessor()

```
sound4::CProcessor::CProcessor ( )  [default]
```

## 14.21.3 Member Function Documentation

### 14.21.3.1 GetAudioInputCount()

```
unsigned int sound4::CProcessor::GetAudioInputCount ( )  [inline]
```

Get the number of inputs for this process

**Remarks**

All inputs have GetChannelCount() channels

**Returns**

The process input count

Definition at line 780 of file sound4cl.hpp.

### 14.21.3.2 GetAudioOutputCount()

```
unsigned int sound4::CProcessor::GetAudioOutputCount ( )  [inline]
```

Get the number of outputs for this process

**Remarks**

All outputs have GetChannelCount() channels

**Returns**

The process outputs count

Definition at line 790 of file sound4cl.hpp.

### 14.21.3.3 GetBytesFromFormat()

```
unsigned int sound4::CProcessor::GetBytesFromFormat (
            const SampleFormat fmt )  [inline]
```

Get the format byte size

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

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

Definition at line 827 of file sound4cl.hpp.

#### 14.21.3.4 GetChannelCount()

```
unsigned int sound4::CProcessor::GetChannelCount ( )  [inline]
```

Get the number of channels needed by this process

**Returns**

The process channels count

Definition at line 770 of file sound4cl.hpp.

#### 14.21.3.5 GetChunkSizeInFrames()

```
unsigned int sound4::CProcessor::GetChunkSizeInFrames ( )  [inline]
```

Get the default 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 GetChannelCount samples

**Returns**

The process chunk size in frame

**Deprecated** Use GetPossibleChunkSizeInFrames

Definition at line 741 of file sound4cl.hpp.

#### 14.21.3.6 GetFormatFromName()

```
SampleFormat sound4::CProcessor::GetFormatFromName (
            const std::string & name )  [inline]
```

Convert string to format

**Parameters**

| | |
|---|---|
| *name* | the format string |

**Returns**

the matching format, or INVALID_FORMAT if invalid.

Definition at line 818 of file sound4cl.hpp.

#### 14.21.3.7 GetFormatName()

```
std::string sound4::CProcessor::GetFormatName (
            const SampleFormat fmt )  [inline]
```

Convert format to string

**Parameters**

| | |
|---|---|
| *fmt* | the format |

**Returns**

the string name of the format

Definition at line 809 of file sound4cl.hpp.

#### 14.21.3.8 GetPossibleChunkSizeInFrames()

```
std::vector< unsigned int > sound4::CProcessor::GetPossibleChunkSizeInFrames ( )  [inline]
```

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 GetChannelCount samples

**Returns**

An vector of possible process chunk size in frame

Definition at line 752 of file sound4cl.hpp.

#### 14.21.3.9 GetPrefixSymbol()

```
template<typename T >
helper::DynFuncHolder< T > sound4::CProcessor::GetPrefixSymbol (
            const std::string & name )  [inline], [protected]
```

Definition at line 409 of file sound4cl.hpp.

#### 14.21.3.10 GetSampleRate()

```
unsigned int sound4::CProcessor::GetSampleRate ( )  [inline]
```

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

Definition at line 800 of file sound4cl.hpp.

#### 14.21.3.11 GetVersion()

```
std::string sound4::CProcessor::GetVersion ( )  [inline]
```

Get the process version string

**Returns**

version string

Definition at line 728 of file sound4cl.hpp.

#### 14.21.3.12 IsOk()

```
bool sound4::CProcessor::IsOk ( ) const  [inline]
```

**Returns**

true if OK.

false if not OK.

Definition at line 615 of file sound4cl.hpp.

#### 14.21.3.13 Load()

```
bool sound4::CProcessor::Load (
            const std::filesystem::path & filepath )  [inline]
```

**Parameters**

| | |
|---|---|
| *filepath* | 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 424 of file sound4cl.hpp.

#### 14.21.3.14 SetLoggerCallback()

```
void sound4::CProcessor::SetLoggerCallback (
            log_cb_t cb )  [inline]
```

Sets the log callback. If not set, console output will be used.

Definition at line 839 of file sound4cl.hpp.

#### 14.21.3.15 SetLogSeverity()

```
void sound4::CProcessor::SetLogSeverity (
            LogSeverity severity )  [inline]
```

Sets the log severity (global)

Definition at line 833 of file sound4cl.hpp.

### 14.21.4 Friends And Related Symbol Documentation

#### 14.21.4.1 CBus

```
friend class CBus  [friend]
```

Definition at line 315 of file sound4cl.hpp.

#### 14.21.4.2 CInstance

```
friend class CInstance  [friend]
```

Definition at line 316 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.22 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

### 14.22.1 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 87 of file sound4.bigvoice.cl_dyn.hpp.

### 14.22.2 Constructor & Destructor Documentation

#### 14.22.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::DynFuncHolder ( )  [default]
```

#### 14.22.2.2 DynFuncHolder() [2/2]

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::DynFuncHolder (
            DynFunc_t a_ptr )  [inline]
```

Definition at line 98 of file sound4.bigvoice.cl_dyn.hpp.

### 14.22.3 Member Function Documentation

#### 14.22.3.1 IsOk()

```
template<typename T >
bool sound4::bigvoice::dyn::helper::DynFuncHolder< T >::IsOk ( ) const  [inline]
```

Definition at line 100 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.22.3.2 operator bool()

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::operator bool ( ) const  [inline]
```

Definition at line 99 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.22.3.3 operator T*()

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::operator T* ( ) const  [inline]
```

Definition at line 101 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.23 sound4::helper::DynFuncHolder< T > Class Template Reference

Helper to wrap a function pointer.

```
#include <sound4cl.hpp>
```

### Public Member Functions

- DynFuncHolder ()=default
- DynFuncHolder (DynFunc_t a_ptr)
- operator bool () const
- bool IsOk () const
- operator T* () const

### 14.23.1 Detailed Description

**template<typename T>**
**class sound4::helper::DynFuncHolder< T >**

**Parameters**

| | |
|---|---|
| *T* | is the function type, you may use decltype(fn) |

Definition at line 134 of file sound4cl.hpp.

## 14.23.2 Constructor & Destructor Documentation

### 14.23.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::helper::DynFuncHolder< T >::DynFuncHolder ( )  [default]
```

### 14.23.2.2 DynFuncHolder() [2/2]

```
template<typename T >
sound4::helper::DynFuncHolder< T >::DynFuncHolder (
            DynFunc_t a_ptr )  [inline]
```

Definition at line 145 of file sound4cl.hpp.

## 14.23.3 Member Function Documentation

### 14.23.3.1 IsOk()

```
template<typename T >
bool sound4::helper::DynFuncHolder< T >::IsOk ( ) const  [inline]
```

Definition at line 147 of file sound4cl.hpp.

### 14.23.3.2 operator bool()

```
template<typename T >
sound4::helper::DynFuncHolder< T >::operator bool ( ) const  [inline]
```

Definition at line 146 of file sound4cl.hpp.

### 14.23.3.3 operator T*()

```
template<typename T >
sound4::helper::DynFuncHolder< T >::operator T* ( ) const  [inline]
```

Definition at line 148 of file sound4cl.hpp.

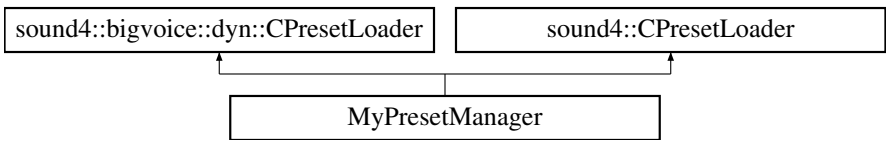
The documentation for this class was generated from the following file:

- sound4cl.hpp

## 14.24 MyPresetManager Class Reference

[Custom preset manager]

Inheritance diagram for MyPresetManager:




### 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
- 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

### Public Member Functions inherited from sound4::bigvoice::dyn::CPresetLoader

- CPresetLoader ()=default
- virtual ~CPresetLoader ()=default

### Public Member Functions inherited from sound4::CPresetLoader

- CPresetLoader ()=default
- virtual ~CPresetLoader ()=default

### 14.24.1 Detailed Description

Definition at line 120 of file sound4.bigvoice.cl-pipe-dyn.cpp.

## 14.24.2 Constructor & Destructor Documentation

### 14.24.2.1 MyPresetManager() [1/2]

```
MyPresetManager::MyPresetManager ( )  [default]
```

### 14.24.2.2 ~MyPresetManager() [1/2]

```
virtual MyPresetManager::~MyPresetManager ( )  [virtual], [default]
```

### 14.24.2.3 MyPresetManager() [2/2]

```
MyPresetManager::MyPresetManager ( )  [default]
```

### 14.24.2.4 ~MyPresetManager() [2/2]

```
virtual MyPresetManager::~MyPresetManager ( )  [virtual], [default]
```

## 14.24.3 Member Function Documentation

### 14.24.3.1 Exists() [1/2]

```
virtual bool MyPresetManager::Exists (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 138 of file sound4.bigvoice.cl-pipe-dyn.cpp.

### 14.24.3.2 Exists() [2/2]

```
virtual bool MyPresetManager::Exists (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 122 of file sound4cl-pipe.cpp.

### 14.24.3.3 GetAll() [1/2]

```
virtual std::vector< std::filesystem::path > MyPresetManager::GetAll ( )  [inline], [override],
[virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 154 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.4 GetAll() [2/2]

```
virtual std::vector< std::filesystem::path > MyPresetManager::GetAll ( )  [inline], [override],
[virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 138 of file sound4cl-pipe.cpp.

#### 14.24.3.5 IsReadOnly() [1/2]

```
virtual bool MyPresetManager::IsReadOnly ( )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 137 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.6 IsReadOnly() [2/2]

```
virtual bool MyPresetManager::IsReadOnly ( )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 121 of file sound4cl-pipe.cpp.

#### 14.24.3.7 Read() [1/2]

```
virtual std::string MyPresetManager::Read (
            const std::filesystem::path & filename )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 168 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.8 Read() [2/2]

```
virtual std::string MyPresetManager::Read (
            const std::filesystem::path & filename )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 152 of file sound4cl-pipe.cpp.

#### 14.24.3.9 Remove() [1/2]

```
virtual bool MyPresetManager::Remove (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 143 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.10 Remove() [2/2]

```
virtual bool MyPresetManager::Remove (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 127 of file sound4cl-pipe.cpp.

#### 14.24.3.11 Rename() [1/2]

```
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 148 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.12 Rename() [2/2]

```
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 132 of file sound4cl-pipe.cpp.

#### 14.24.3.13 SetupPath() [1/2]

```
bool MyPresetManager::SetupPath (
            const std::filesystem::path & a_preset_path )  [inline]
```

Definition at line 125 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.14 SetupPath() [2/2]

```
bool MyPresetManager::SetupPath (
            const std::filesystem::path & a_preset_path )  [inline]
```

Definition at line 109 of file sound4cl-pipe.cpp.

#### 14.24.3.15 Write() [1/2]

```
virtual bool MyPresetManager::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 178 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 14.24.3.16 Write() [2/2]

```
virtual bool MyPresetManager::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 162 of file sound4cl-pipe.cpp.

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

- sound4.bigvoice.cl-pipe-dyn.cpp
- sound4cl-pipe.cpp

## 14.25 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

### 14.25.1 Detailed Description

**template<typename T>**
**struct sound4::bigvoice::dyn::helper::SampleFormat< T >**

**Template Parameters**

| | |
|---|---|
| *T* | the C type for one audio sample |

Definition at line 566 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.26 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

### 14.26.1 Detailed Description

**template<typename T>**
**struct sound4::bigvoice::helper::SampleFormat< T >**

**Template Parameters**

| | |
|---|---|
| *T* | the C type for one audio sample |

Definition at line 132 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.27 sound4::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

`#include <sound4cl.hpp>`

### 14.27.1 Detailed Description

**template<typename T>**
**struct sound4::helper::SampleFormat< T >**

**Template Parameters**

| | |
|---|---|
| *T* | the C type for one audio sample |

Definition at line 113 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.28 SOUND4_AuxiliaryOutputInfos Struct Reference

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

**Public Attributes**

- char * id
  *The ID of the output (fixed across versions)*
- char * description
  *The description of the output.*
- unsigned int sample_rate
  *The sampling rate of this output.*
- unsigned int channel_count
  *The number of channels of this output.*
- const float * buffer
  *The output buffer of the output. See bigvoice_ProcessAudio for format.*

### 14.28.1 Detailed Description

Structure holding information on an auxiliary output

Definition at line 810 of file sound4.bigvoice.cl.h.

### 14.28.2 Member Data Documentation

#### 14.28.2.1 buffer

```
const float* SOUND4_AuxiliaryOutputInfos::buffer
```

Definition at line 815 of file sound4.bigvoice.cl.h.

#### 14.28.2.2 channel_count

```
unsigned int SOUND4_AuxiliaryOutputInfos::channel_count
```

Definition at line 814 of file sound4.bigvoice.cl.h.

#### 14.28.2.3 description

```
char* SOUND4_AuxiliaryOutputInfos::description
```

Definition at line 812 of file sound4.bigvoice.cl.h.

#### 14.28.2.4 id

```
char* SOUND4_AuxiliaryOutputInfos::id
```

Definition at line 811 of file sound4.bigvoice.cl.h.

#### 14.28.2.5 sample_rate

```
unsigned int SOUND4_AuxiliaryOutputInfos::sample_rate
```

Definition at line 813 of file sound4.bigvoice.cl.h.

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

- sound4.bigvoice.cl.h

## 14.29 SOUND4_ProcessInfo Struct Reference

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

**Public Attributes**

- size_t struct_size
  *The size of total this structure, for compatibility checking.*
- const char * name
  *The name of the processor.*
- const char * prefix
  *The prefix of this processor functions.*
- const char * shortname
  *A short name of the processor, single word.*
- const char * version
  *Library version.*

## 14.29.1 Detailed Description

Information on this processor.

**See also**

SOUND4_GetProcessInfo

Definition at line 77 of file sound4.bigvoice.cl.h.

## 14.29.2 Member Data Documentation

### 14.29.2.1 name

```
const char* SOUND4_ProcessInfo::name
```

Definition at line 79 of file sound4.bigvoice.cl.h.

### 14.29.2.2 prefix

```
const char* SOUND4_ProcessInfo::prefix
```

Definition at line 80 of file sound4.bigvoice.cl.h.

### 14.29.2.3 shortname

```
const char* SOUND4_ProcessInfo::shortname
```

Definition at line 81 of file sound4.bigvoice.cl.h.

### 14.29.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

Definition at line 78 of file sound4.bigvoice.cl.h.

### 14.29.2.5 version

```
const char* SOUND4_ProcessInfo::version
```

Definition at line 82 of file sound4.bigvoice.cl.h.

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

- sound4.bigvoice.cl.h

# Chapter 15

# File Documentation

## 15.1 doc_examples.h File Reference

## 15.2 doc_examples.h

Go to the documentation of this file.

```
00001
00022 // ------------------------------------------------------------
```

## 15.3 doc_gencpp.h File Reference

### Namespaces

- namespace sound4

## 15.4 doc_gencpp.h

Go to the documentation of this file.

```
00001 namespace sound4 {
00002
00059 }; // namespace sound4
```

## 15.5 doc_json.h File Reference

## 15.6 doc_json.h

Go to the documentation of this file.

```
00001
```

## 15.7 doc_main.h File Reference

## 15.8 doc_main.h

Go to the documentation of this file.
```
00001
```

## 15.9 doc_params.h File Reference

## 15.10 doc_params.h

Go to the documentation of this file.
```
00001
00002 // ------------------------------------------------------------
```

## 15.11 doc_only/documentation.h File Reference

## 15.12 doc_only/documentation.h

Go to the documentation of this file.
```
00001
```

## 15.13 documentation.h File Reference

## 15.14 documentation.h

Go to the documentation of this file.
```
00001
```

## 15.15 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 *progname)
- void MyLogger (sound4::bigvoice::LogSeverity severity, const std::string &msg)
- int main (int argc, const char **argv)

## 15.15.1 Function Documentation

### 15.15.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 41 of file sound4.bigvoice.cl-pipe-cxx.cpp.

### 15.15.1.2 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 106 of file sound4.bigvoice.cl-pipe-cxx.cpp.

### 15.15.1.3 MyLogger()

```
void MyLogger (
            sound4::bigvoice::LogSeverity severity,
            const std::string & msg )
```

Definition at line 101 of file sound4.bigvoice.cl-pipe-cxx.cpp.

### 15.15.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::bigvoice::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 50 of file sound4.bigvoice.cl-pipe-cxx.cpp.

#### 15.15.1.5 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 81 of file sound4.bigvoice.cl-pipe-cxx.cpp.

## 15.16 sound4.bigvoice.cl-pipe-cxx.cpp

Go to the documentation of this file.

```
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 // ------------------------------------------------------------
00041 std::string GetEnv(const std::string& name)
00042 {
00043     char *data=getenv(name.c_str());
00044     if (!data) return {};
00045     return std::string(data);
00046 }
00047
00048
00049 // Reads from stdin, write to stdout
00050 void ProcessLoop(sound4::bigvoice::CInstance &instance)
00051 {
00052 bool stopthread=false;
00053
00054     #if defined(_WIN32) || defined(WIN32)
00055         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00056         _setmode(_fileno(stdin), _O_BINARY);
00057         _setmode(_fileno(stdout), _O_BINARY);
00058     #endif
00059
00061     while (!stopthread)
00062     {
00063         // Read a chunk from stdin
00064         if (std::fread(instance.GetBufferIn().data(), sizeof(float),
      sound4::bigvoice::InputSampleSize, stdin) != sound4::bigvoice::InputSampleSize) {
00065             std::cerr « "ProcessLoop: input stopped"« std::endl;
00066             stopthread=true;
00067             break;
00068         }
00069         if (stopthread) break;
00070         // Process it
00071         instance.ProcessAudio();
00072         // Write it to stdout
00073         if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::bigvoice::OutputSampleSize,
      stdout) != sound4::bigvoice::OutputSampleSize) {
00074             std::cerr « "Failed to write audio output" « std::endl;
00075             stopthread=true;
00076         }
00077     }
00079 }
00080
00081 void Usage(const char *progname)
00082 {
00083     std::cerr « "Usage: " « progname « std::endl;;
00084     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00085     std::cerr « "Environment:"« std::endl;;
00086     std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
      std::endl;;
00087     std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00088     std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00089     std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00090     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00091     std::cerr « std::endl;
00092     std::cerr « "Example:"« std::endl;
```

```
00093     std::cerr « "ffmpeg -re -i file.wav -ar "«sound4::bigvoice::GetSampleRate()«" -ac
      "«sound4::bigvoice::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::bigvoice::GetSampleRate()«" -ac "«sound4::bigvoice::GetChannelCount()«" -i -" « std::endl;;
00094     #if defined(_WIN32) || defined(WIN32)
00095         std::cerr « "ffmpeg -f dshow -i audio=\"Microphone\" -ar "«sound4::bigvoice::GetSampleRate()«"
      -ac "«sound4::bigvoice::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::bigvoice::GetSampleRate()«" -ac "«sound4::bigvoice::GetChannelCount()«" -i -" « std::endl;;
00096     #else
00097         std::cerr « "ffmpeg -f pulse -i default -ar "«sound4::bigvoice::GetSampleRate()«" -ac
      "«sound4::bigvoice::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::bigvoice::GetSampleRate()«" -ac "«sound4::bigvoice::GetChannelCount()«" -i -" « std::endl;;
00098     #endif
00099 }
00100
00101 void MyLogger(sound4::bigvoice::LogSeverity severity,const std::string& msg)
00102 {
00103     std::cerr « msg « std::endl;
00104 }
00105
00106 int main(int argc, const char **argv)
00107 {
00108 #if (BIGVOICE_HAS_WEBSERVER)
00109     int listen_port = 8080;
00110 #endif // (BIGVOICE_HAS_WEBSERVER)
00111     auto store_path_str=GetEnv("S4STORE_PATH");
00112
00113     // Get credentials from environment
00114     auto LoginKey=GetEnv("S4LOGINKEY");
00115     if (LoginKey.empty()) {
00116         LoginKey=GetEnv("S4USERID");
00117     }
00118     auto RadioName=GetEnv("RADIO_NAME");
00119     auto KeyId=GetEnv("ACCESS_KEY_ID");
00120     auto AccessKey=GetEnv("ACCESS_KEY_SECRET");
00121
00122     const char *progname=argv[0];
00123
00124     if (argc>1) {
00125         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00126             Usage(progname);
00127             return 0;
00128         }
00129     }
00130     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00131         std::cerr « "Missing credentials in environment" « std::endl;
00132         Usage(progname);
00133         return 1;
00134     }
00135     std::filesystem::path store_path;
00136     if (!store_path_str.empty()) {
00137         std::error_code ec;
00138         store_path = std::filesystem::path(store_path_str);
00139         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
      {
00140             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
      std::endl;
00141             Usage(progname);
00142             return 1;
00143         }
00144     }
00145
00147     sound4::bigvoice::SetLogSeverity(verbose5);
00148     sound4::bigvoice::SetLoggerCallback(MyLogger);
00149
00150     sound4::bigvoice::CInstance instance;
00151     instance.SetParam("ADMIN_USER","admin");
00152     instance.SetParam("ADMIN_SECRET","admin");
00154
00155     std::cerr « "Creating" « std::endl;
00157     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, store_path)) {
00158         std::cerr « "Failed to create the processing instance" « std::endl;
00159         return 1;
00160     }
00162
00163 #if (BIGVOICE_HAS_WEBSERVER)
00164     std::cerr « "Starting web server" « std::endl;
00166     if (!instance.StartWebServer(listen_port,0)) {
00167         std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
00168         return 1;
00169     }
00171 #endif // (BIGVOICE_HAS_WEBSERVER)
00172
00173     {
00175         auto client = instance.NewClient();
00176         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00177         std::cerr « "Request for processor name returned "«answer«std::endl;
00179     }
```

```
00180
00181     ProcessLoop(instance);
00182
00183     std::cerr « "Stopping now" « std::endl;
00184     instance.Stop();
00185
00186     std::cerr « "End" « std::endl;
00187     return 0;
00188 }
```

## 15.17 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]*

### 15.17.1 Function Documentation

#### 15.17.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 52 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 15.17.1.2 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 199 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 15.17.1.3 MyLogger()

```
void MyLogger (
            sound4::bigvoice::dyn::LogSeverity severity,
            const std::string & msg )
```

Definition at line 114 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 15.17.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::bigvoice::dyn::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 61 of file sound4.bigvoice.cl-pipe-dyn.cpp.

#### 15.17.1.5 Usage()

```
void Usage (
            const char * progname,
            sound4::bigvoice::dyn::CDynLib & dynlib )
```

Definition at line 92 of file sound4.bigvoice.cl-pipe-dyn.cpp.

# 15.18 sound4.bigvoice.cl-pipe-dyn.cpp

Go to the documentation of this file.

```
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 // ------------------------------------------------------------
00052 std::string GetEnv(const std::string& name)
00053 {
00054     char *data=getenv(name.c_str());
00055     if (!data) return {};
00056     return std::string(data);
00057 }
00058
00059
00060 // Reads from stdin, write to stdout
00061 void ProcessLoop(sound4::bigvoice::dyn::CInstance &instance)
00062 {
00063 bool stopthread=false;
00064
00065     #if defined(_WIN32) || defined(WIN32)
00066         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00067         _setmode(_fileno(stdin), _O_BINARY);
00068         _setmode(_fileno(stdout), _O_BINARY);
00069     #endif
00070
00072     while (!stopthread)
00073     {
00074         // Read a chunk from stdin
00075         if (std::fread(instance.GetBufferIn().data(), sizeof(float),
      sound4::bigvoice::dyn::InputSampleSize, stdin) != sound4::bigvoice::dyn::InputSampleSize) {
00076             std::cerr « "ProcessLoop: input stopped"« std::endl;
00077             stopthread=true;
00078             break;
00079         }
00080         if (stopthread) break;
00081         // Process it
00082         instance.ProcessAudio();
00083         // Write it to stdout
00084         if (fwrite(instance.GetBufferOut().data(), sizeof(float),
      sound4::bigvoice::dyn::OutputSampleSize, stdout) != sound4::bigvoice::dyn::OutputSampleSize) {
00085             std::cerr « "Failed to write audio output" « std::endl;
00086             stopthread=true;
00087         }
00088     }
00090 }
00091
00092 void Usage(const char *progname, sound4::bigvoice::dyn::CDynLib &dynlib)
00093 {
00094     std::cerr « "Usage: " « progname « std::endl;;
00095     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00096     std::cerr « "Environment:"« std::endl;;
00097     std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
      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 « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00101     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00102     if (dynlib.IsOk()) {
00103         std::cerr « std::endl;
00104         std::cerr « "Example:"« std::endl;
00105         std::cerr « "ffmpeg -re -i file.wav -ar "«dynlib.GetSampleRate()«" -ac
      "«dynlib.GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«dynlib.GetSampleRate()«" -ac "«dynlib.GetChannelCount()«" -i -" « std::endl;;
00106         #if defined(_WIN32) || defined(WIN32)
00107             std::cerr « "ffmpeg -f dshow -i audio=\"Microphone\" -ar "«dynlib.GetSampleRate()«" -ac
      "«dynlib.GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«dynlib.GetSampleRate()«" -ac "«dynlib.GetChannelCount()«" -i -" « std::endl;;
00108         #else
00109             std::cerr « "ffmpeg -f pulse -i default -ar "«dynlib.GetSampleRate()«" -ac
      "«dynlib.GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«dynlib.GetSampleRate()«" -ac "«dynlib.GetChannelCount()«" -i -" « std::endl;;
00110         #endif
```

```
00111     }
00112 }
00113
00114 void MyLogger(sound4::bigvoice::dyn::LogSeverity severity,const std::string& msg)
00115 {
00116     std::cerr « msg « std::endl;
00117 }
00118
00120 class MyPresetManager : public sound4::bigvoice::dyn::CPresetLoader {
00121 public:
00122     MyPresetManager() = default;
00123     virtual ~MyPresetManager() = default;
00124
00125     bool SetupPath(const std::filesystem::path& a_preset_path)
00126         {
00127             m_preset_path=a_preset_path;
00128             std::error_code ec;
00129             if ( !std::filesystem::exists(m_preset_path,ec ) ) {
00130                 return false;
00131             }
00132             if ( !std::filesystem::is_directory(m_preset_path, ec) ) {
00133                 return false;
00134             }
00135             return true;
00136         }
00137     virtual bool IsReadOnly() override { return false; };
00138     virtual bool Exists(const std::filesystem::path &name) override
00139         {
00140             std::error_code ec;
00141             return std::filesystem::exists(GetRealPath(name), ec);
00142         }
00143     virtual bool Remove(const std::filesystem::path &name) override
00144         {
00145             std::error_code ec;
00146             return std::filesystem::remove(GetRealPath(name), ec);
00147         }
00148     virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) override
00149         {
00150             std::error_code ec;
00151             std::filesystem::rename(GetRealPath(from), GetRealPath(to), ec);
00152             return (!ec);
00153         }
00154     virtual std::vector<std::filesystem::path> GetAll() override
00155         {
00156             std::vector<std::filesystem::path> list;
00157             std::error_code ec;
00158             for (auto it = std::filesystem::directory_iterator(m_preset_path.c_str(),
      std::filesystem::directory_options::skip_permission_denied, ec);
00159                  !ec && it != std::filesystem::directory_iterator{};
00160                   it.increment(ec) )
00161             {
00162                 if (it->is_regular_file(ec)) {
00163                     list.push_back(it->path().lexically_relative(m_preset_path));
00164                 }
00165             }
00166             return list;
00167         }
00168     virtual std::string Read(const std::filesystem::path &filename) override
00169         {
00170             std::ifstream f(GetRealPath(filename));
00171             if ( f.fail() ) {
00172                 return std::string();
00173             }
00174             std::ostringstream ss;
00175             ss « f.rdbuf();
00176             return ss.str();
00177         }
00178     virtual bool Write(const std::filesystem::path &filename, const std::string &content) override
00179         {
00180             std::ofstream f(GetRealPath(filename));
00181             if ( f.fail() ) {
00182                 return false;
00183             }
00184             f « content;
00185             if ( f.fail() ) {
00186                 return false;
00187             }
00188             return true;
00189         }
00190 private:
00191     std::filesystem::path m_preset_path;
00192     std::filesystem::path GetRealPath(const std::filesystem::path &name)
00193         {
00194             return m_preset_path / name;
00195         }
00196 };
00198
```

```
00199 int main(int argc, const char **argv)
00200 {
00201 #if (BIGVOICE_HAS_WEBSERVER)
00202     int listen_port = 8080;
00203 #endif // (BIGVOICE_HAS_WEBSERVER)
00204     auto store_path_str=GetEnv("S4STORE_PATH");
00205 
00206     // Get credentials from environment
00207     auto LoginKey=GetEnv("S4LOGINKEY");
00208     if (LoginKey.empty()) {
00209         LoginKey=GetEnv("S4USERID");
00210     }
00211     auto RadioName=GetEnv("RADIO_NAME");
00212     auto KeyId=GetEnv("ACCESS_KEY_ID");
00213     auto AccessKey=GetEnv("ACCESS_KEY_SECRET");
00214 
00215     const char *progname=argv[0];
00216 
00217 
00219     sound4::bigvoice::dyn::CDynLib dynlib;
00220     if (!dynlib.Load()) {
00221         std::cerr « "Failed to load processing library" « std::endl;
00222         return 1;
00223     }
00225 
00226     if (argc>1) {
00227         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00228             Usage(progname,dynlib);
00229             return 0;
00230         }
00231     }
00232     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00233         std::cerr « "Missing credentials in environment" « std::endl;
00234         Usage(progname,dynlib);
00235         return 1;
00236     }
00237     std::filesystem::path store_path;
00238     if (!store_path_str.empty()) {
00239         std::error_code ec;
00240         store_path = std::filesystem::path(store_path_str);
00241         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
    {
00242             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
    std::endl;
00243             Usage(progname,dynlib);
00244             return 1;
00245         }
00246     }
00247 
00249     sound4::bigvoice::dyn::SetLogSeverity(dynlib, verbose5);
00250     sound4::bigvoice::dyn::SetLoggerCallback(dynlib, MyLogger);
00251 
00252     sound4::bigvoice::dyn::CInstance instance(dynlib);
00253     instance.SetParam("ADMIN_USER","admin");
00254     instance.SetParam("ADMIN_SECRET","admin");
00256 
00258     MyPresetManager preset_manager;
00259     if (!store_path.empty()) {
00260         if (!preset_manager.SetupPath(store_path)) {
00261             std::cerr « "Failed to setup the custom preset manager on path " « store_path.c_str() «
    std::endl;
00262             return 1;
00263         }
00264         instance.SetPresetManager(&preset_manager);
00265     }
00267 
00268     std::cerr « "Creating" « std::endl;
00270     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00271         std::cerr « "Failed to create the processing instance" « std::endl;
00272         return 1;
00273     }
00275 
00276 #if (BIGVOICE_HAS_WEBSERVER)
00277     std::cerr « "Starting web server" « std::endl;
00279     if (!instance.StartWebServer(listen_port,0)) {
00280         std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
00281         return 1;
00282     }
00284 #endif // (BIGVOICE_HAS_WEBSERVER)
00285 
00286     {
00288         auto client = instance.NewClient();
00289         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00290         std::cerr « "Request for processor name returned "«answer«std::endl;
00292     }
00293 
00294     ProcessLoop(instance);
```

```
00295
00296     std::cerr « "Stopping now" « std::endl;
00297     instance.Stop();
00298
00299     std::cerr « "End" « std::endl;
00300     return 0;
00301 }
```

## 15.19 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 <filesystem>
#include <io.h>
#include <fcntl.h>
#include <stringapiset.h>
#include <sound4.bigvoice.cl.h>
```

### 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, const char ∗str)
- int main (int argc, char ∗const argv[ ])

### 15.19.1 Macro Definition Documentation

#### 15.19.1.1 CHANNEL_COUNT

```
#define CHANNEL_COUNT 2
```

Definition at line 22 of file sound4.bigvoice.cl-pipe.cpp.

#### 15.19.1.2 CHUNK_SIZE

```
#define CHUNK_SIZE 64
```

Definition at line 21 of file sound4.bigvoice.cl-pipe.cpp.

#### 15.19.1.3 SAMPLE_RATE

```
#define SAMPLE_RATE 48000
```

Definition at line 20 of file sound4.bigvoice.cl-pipe.cpp.

### 15.19.2 Function Documentation

#### 15.19.2.1 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 84 of file sound4.bigvoice.cl-pipe.cpp.

#### 15.19.2.2 MyLogger()

```
void MyLogger (
            bigvoice_LogSeverity severe,
            const char * str )
```

Definition at line 78 of file sound4.bigvoice.cl-pipe.cpp.

#### 15.19.2.3 ProcessLoop()

```
void ProcessLoop (
            struct bigvoice_CInstance * pInstance )
```

[Processing Loop]

[Processing Loop]

Definition at line 25 of file sound4.bigvoice.cl-pipe.cpp.

#### 15.19.2.4 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 58 of file sound4.bigvoice.cl-pipe.cpp.

# 15.20 sound4.bigvoice.cl-pipe.cpp

Go to the documentation of this file.

```
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 #if (__cplusplus<201703L)
00010     #error "Need C++17 for this example"
00011 #endif
00012 #include <filesystem>
00013 #if defined(_WIN32) || defined(WIN32)
00014     #include <io.h>
00015     #include <fcntl.h>
00016     #include <stringapiset.h>   // UTF-8 conversion
00017 #endif
00018 #include <sound4.bigvoice.cl.h>
00019
00020 #define SAMPLE_RATE       48000
00021 #define CHUNK_SIZE        64
00022 #define CHANNEL_COUNT     2
00023
00024 // Reads from stdin, write to stdout
00025 void ProcessLoop(struct bigvoice_CInstance *pInstance)
00026 {
00028 bool stopthread=false;
00029 alignas(64) float audio_in[CHUNK_SIZE*CHANNEL_COUNT];
00030 alignas(64) float audio_out[CHUNK_SIZE*CHANNEL_COUNT];
00031
00032     #if defined(_WIN32) || defined(WIN32)
00033         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00034         _setmode(_fileno(stdin), _O_BINARY);
00035         _setmode(_fileno(stdout), _O_BINARY);
00036     #endif
00037
00038     while (!stopthread)
00039     {
00040         // Read a chunk from stdin
00041         if (std::fread(&audio_in[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdin) !=
    CHUNK_SIZE*CHANNEL_COUNT) {
00042             std::cerr « "ProcessLoop: input stopped"« std::endl;
00043             stopthread=true;
00044             break;
00045         }
00046         if (stopthread) break;
00047         // Process it
00048         bigvoice_ProcessAudio(pInstance, audio_in, audio_out);
00049         // Write it to stdout
00050         if (fwrite(&audio_out[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdout) !=
    CHUNK_SIZE*CHANNEL_COUNT) {
00051             std::cerr « "Failed to write audio output" « std::endl;
00052             stopthread=true;
00053         }
00054     }
00056 }
00057
00058 void Usage(const char *progname)
00059 {
00060     std::cerr « "Usage: " « progname « std::endl;;
00061     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00062     std::cerr « "Environment:"« std::endl;;
00063     std::cerr « " - S4STORE_PATH            : path to store config [default: ./state]"« std::endl;;
00064     std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00065     std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00066     std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00067     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00068     std::cerr « std::endl;
00069     std::cerr « "Example:"« std::endl;
00070     std::cerr « "ffmpeg -re -i file.wav -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«" -f f32le - |
    "«progname«" | ffplay -f f32le -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«" -i -" « std::endl;;
00071     #if defined(_WIN32) || defined(WIN32)
00072         std::cerr « "ffmpeg -f dshow -i audio=\"Microphone\" -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«"
    -f f32le - | "«progname«" | ffplay -f f32le -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«" -i -" «
    std::endl;;
00073     #else
00074         std::cerr « "ffmpeg -f pulse -i default -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«" -f f32le - |
    "«progname«" | ffplay -f f32le -ar "«SAMPLE_RATE«" -ac "«CHANNEL_COUNT«" -i -" « std::endl;;
00075     #endif
00076 }
00077
00078 void MyLogger(bigvoice_LogSeverity severe [[maybe_unused]], const char *str)
00079 {
00080     std::cerr « str « std::endl;
```

```
00081 }
00082
00083
00084 int main(int argc, char *const argv[])
00085 {
00086     struct bigvoice_CInstance *pInstance = NULL;
00087     const char *store_path_str=getenv("S4STORE_PATH");
00088     // Get credentials from environment
00089     char *pLoginKey=getenv("S4LOGINKEY");
00090     if (!pLoginKey) {
00091         pLoginKey=getenv("S4USERID");
00092     }
00093     char *pRadioName=getenv("RADIO_NAME");
00094     char *pKeyId=getenv("ACCESS_KEY_ID");
00095     char *pAccessKey=getenv("ACCESS_KEY_SECRET");
00096
00097     const char *progname=argv[0];
00098
00099     if (argc>1) {
00100         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00101             Usage(progname);
00102             return 0;
00103         }
00104     }
00105     if (!pLoginKey || !pRadioName || !pKeyId || !pAccessKey) {
00106         std::cerr « "Missing credentials in environment" « std::endl;
00107         Usage(progname);
00108         return 1;
00109     }
00110     std::error_code ec;
00111     std::filesystem::path fs_store_path;
00112     if (!store_path_str || store_path_str[0]==0) {
00113         fs_store_path=std::filesystem::current_path(ec) / "state";
00114     } else {
00115         fs_store_path = std::filesystem::path(store_path_str);
00116     }
00117     if (!std::filesystem::exists(fs_store_path,ec) ||
    !std::filesystem::is_directory(fs_store_path,ec)) {
00118         std::cerr « "Storage folder '"«fs_store_path.c_str()«"' is not a valid directory" « std::endl;
00119         Usage(progname);
00120         return 1;
00121     }
00122     std::string store_path;
00123     #ifdef _WIN32
00124     {
00125         // Convert path to UTF-8
00126         std::wstring wstr(fs_store_path.wstring());
00127         if (wstr.empty()) return {};
00128         int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
00129         std::string str(size, 0);
00130         WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00131         store_path=str;
00132     }
00133     #else // !_WIN32
00134         store_path=fs_store_path;
00135     #endif // !_WIN32
00136
00138     // Optional: Use our own logging, instead of default color output to stderr.
00139     //bigvoice_SetLoggerCallback(&MyLogger);
00140
00141     //bigvoice_SetLogSeverity(bigvoice_LogSeverity::info);
00142     bigvoice_SetLogSeverity(bigvoice_LogSeverity::verbose5);
00143
00144     std::cerr « "Process version: " « bigvoice_GetVersion() « std::endl;
00145     // Do some sanity checks
00146     if (bigvoice_GetSampleRate()!=SAMPLE_RATE) {
00147       std::cerr « "Bad library sampling rate, expecting '"«SAMPLE_RATE«"' but has "«
    bigvoice_GetSampleRate() « std::endl;
00148       return 1;
00149     }
00150     if (bigvoice_GetChunkSizeInFrames()!=CHUNK_SIZE) {
00151       std::cerr « "Bad library frame size, expecting '"«CHUNK_SIZE«"' but has "«
    bigvoice_GetChunkSizeInFrames() « std::endl;
00152       return 1;
00153     }
00154     if (bigvoice_GetChannelCount()!=CHANNEL_COUNT) {
00155       std::cerr « "Bad library channel count, expecting '"«CHANNEL_COUNT«"' but has "«
    bigvoice_GetChannelCount() « std::endl;
00156       return 1;
00157     }
00159
00161     // Setup parameters
00162     struct bigvoice_CParameters *params = bigvoice_NewParameters();
00163     // Optional: activate web authentication, set admin password
00164     // This creates the authentication only for a new storage path
00165     bigvoice_SetParameter(params,"ADMIN_USER", "admin");
00166     bigvoice_SetParameter(params,"ADMIN_SECRET", "admin");
```

```
00167
00168     // Create the processing instance
00169     pInstance=bigvoice_InitProcess2(pLoginKey,pRadioName,pKeyId,pAccessKey,store_path.c_str(),
      params);
00170     bigvoice_FreeParameters(params);
00171     if (!pInstance) {
00172       std::cerr « "Failed to create the process instance"« std::endl;
00173       return 1;
00174     }
00175
00176     // Start the update thread, without JSON server (low priority)
00177     std::thread updater([pInstance](){
00178        // will returned when stopped or destroyed
00179        std::cerr « "bigvoice_StartUpdateThread (no json port)" « std::endl;
00180        bigvoice_StartUpdateThread(pInstance, 0);
00181        std::cerr « "bigvoice_StartUpdateThread END" « std::endl;
00182      });
00183
00184     // This is not mandatory, but better be sure we are ready right now
00185     std::cerr « "Waiting update thread" « std::endl;
00186     if (bigvoice_WaitUpdateThreadReady(pInstance, 5000)<0) {
00187         std::cerr « "Timeout to create the process instance"« std::endl;
00188     }
00189     std::cerr « "Waiting update thread DONE" « std::endl;
00190
00191 #if (BIGVOICE_HAS_WEBSERVER)
00193     // Starts a web server, listening on port http://localhost:8080, https://localhost:8443
00194     std::cerr « "Starting web server" « std::endl;
00195     int listenport= 8080;
00196     int listenport_secure= 8443;
00197     uint64_t webserver= SOUND4_INVALID_WEBSERVER_ID;
00198     if (listenport || listenport_secure) {
00199         webserver = bigvoice_Webserver(listenport,listenport_secure,pInstance);
00200         if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00201             std::cerr « "Starting web server FAILED" « std::endl;
00202         } else {
00203             int web_status=bigvoice_Webserver_Status(webserver);
00204             if (listenport) {
00205                 if (web_status & SOUND4_WEBSERVER_HTTP_OK) {
00206                     std::cerr « "Web server listening on port " « listenport;
00207                 } else {
00208                     std::cerr « "Web server failed to listen on port " « listenport;
00209                 }
00210             }
00211             if (listenport_secure) {
00212                 if (web_status & SOUND4_WEBSERVER_HTTPS_OK) {
00213                     std::cerr « "Web server listening on secure port " « listenport_secure;
00214                 } else {
00215                     std::cerr « "Web server failed to listen on secure port " « listenport_secure;
00216                 }
00217             }
00218             std::cerr « "Starting web server DONE" « std::endl;
00219         }
00220     }
00222 #endif // (BIGVOICE_HAS_WEBSERVER)
00223
00224      // Force load a factory preset now
00225      // Could be used instead of setting a store_path
00226      /*{
00227          bigvoice_CClientInstance *pClient=bigvoice_NewClient(pInstance);
00228          const char *answer=bigvoice_ProcessJson(pClient, "{ \"factoryload\": \"All Programs\" }",
      NULL);
00229          std::cerr « "Loaded factory preset answer: " « answer« std::endl;
00230          bigvoice_FreeJsonAnswer(answer);
00231          bigvoice_DeleteClient(pClient);
00232      }*/
00234
00235     std::cerr « "Estimated delay " « ( bigvoice_GetEstimatedDelay(pInstance)*1000.0/SAMPLE_RATE ) « "
      ms" « std::endl;
00236
00237     ProcessLoop(pInstance);
00238
00240 #if (BIGVOICE_HAS_WEBSERVER)
00241     // Stopping the webserver
00242     std::cerr « "Stopping web server" « std::endl;
00243     bigvoice_Webserver_Stop(webserver, 1000);
00244     std::cerr « "Stopping web server DONE" « std::endl;
00245 #endif // (BIGVOICE_HAS_WEBSERVER)
00246
00247     // Stopping the processing instance
00248     std::cerr « "Stopping Update thread" « std::endl;
00249     bigvoice_StopUpdateThread(pInstance); // this will exit the update thread
00250     std::cerr « "Stopping Update thread DONE" « std::endl;
00251
00252     updater.join();
00253
00254     bigvoice_ExitProcess(pInstance);
```

```
00256
00257     return 0;
00258 }
```

## 15.21 sound4cl-pipe.cpp File Reference

C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided.

```
#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 "sound4cl.hpp"
```

### Classes

- class MyPresetManager

  *[Custom preset manager]*

### Functions

- std::string GetEnv (const std::string &name)
- void ProcessLoop (sound4::CInstance &instance, sound4::CProcessor &processor)
- int main (int argc, const char **argv)

  *[Custom preset manager]*

### 15.21.1 Function Documentation

#### 15.21.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 61 of file sound4cl-pipe.cpp.

#### 15.21.1.2 main()

```
int main (
             int argc,
             const char ** argv )
```

[Dynamic loading config]

[Dynamic loading config]

[Dynamic loading]

[Dynamic loading]

[Setup and configure]

[Setup and configure]

[Custom preset manager loading]

[Custom preset manager loading]

[Create the processing instance]

[Create the processing instance]

[Starting a web-server]

[Starting a web-server]

[Communicating with the instance]

[Communicating with the instance]

Definition at line 183 of file sound4cl-pipe.cpp.

#### 15.21.1.3 ProcessLoop()

```
void ProcessLoop (
             sound4::CInstance & instance,
             sound4::CProcessor & processor )
```

[Processing Loop]

[Processing Loop]

Definition at line 70 of file sound4cl-pipe.cpp.

# 15.22 sound4cl-pipe.cpp

Go to the documentation of this file.

```
00001
00006 #include <unistd.h>
00007 #include <iostream>
00008 #include <strings.h>
00009 #include <sys/types.h>
00010 #include <filesystem>
00011 #include <fstream>
00012 #include <sstream>
00013 #if defined(_WIN32) || defined(WIN32)
00014     #include <io.h>
00015     #include <fcntl.h>
00016 #endif
00017
00018 #include "sound4cl.hpp"
00019
00020 // ------------------------------------------------------------
00061 std::string GetEnv(const std::string& name)
00062 {
00063     char *data=getenv(name.c_str());
00064     if (!data) return {};
00065     return std::string(data);
00066 }
00067
00068
00069 // Reads from stdin, write to stdout
00070 void ProcessLoop(sound4::CInstance &instance, sound4::CProcessor &processor)
00071 {
00072 bool stopthread=false;
00073
00074     #if defined(_WIN32) || defined(WIN32)
00075         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00076         _setmode(_fileno(stdin), _O_BINARY);
00077         _setmode(_fileno(stdout), _O_BINARY);
00078     #endif
00079
00081     size_t input_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
00082     size_t output_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
00083     while (!stopthread)
00084     {
00085         // Read a chunk from stdin
00086         if (std::fread(instance.GetBufferIn(), sizeof(float), input_sample_count, stdin) !=
      input_sample_count) {
00087             std::cerr « "ProcessLoop: input stopped"« std::endl;
00088             stopthread=true;
00089             break;
00090         }
00091         if (stopthread) break;
00092         // Process it
00093         instance.ProcessAudio();
00094         // Write it to stdout
00095         if (fwrite(instance.GetBufferOut(), sizeof(float), output_sample_count, stdout) !=
      output_sample_count) {
00096             std::cerr « "Failed to write audio output" « std::endl;
00097             stopthread=true;
00098         }
00099     }
00101 }
00102
00104 class MyPresetManager : public sound4::CPresetLoader {
00105 public:
00106     MyPresetManager() = default;
00107     virtual ~MyPresetManager() = default;
00108
00109     bool SetupPath(const std::filesystem::path& a_preset_path)
00110         {
00111             m_preset_path=a_preset_path;
00112             std::error_code ec;
00113             if ( !std::filesystem::exists(m_preset_path,ec ) ) {
00114                 return false;
00115             }
00116             if ( !std::filesystem::is_directory(m_preset_path, ec) ) {
00117                 return false;
00118             }
00119             return true;
00120         }
00121     virtual bool IsReadOnly() override { return false; };
00122     virtual bool Exists(const std::filesystem::path &name) override
00123         {
00124             std::error_code ec;
00125             return std::filesystem::exists(GetRealPath(name), ec);
00126         }
00127     virtual bool Remove(const std::filesystem::path &name) override
```

```
00128         {
00129             std::error_code ec;
00130             return std::filesystem::remove(GetRealPath(name), ec);
00131         }
00132     virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) override
00133         {
00134             std::error_code ec;
00135             std::filesystem::rename(GetRealPath(from), GetRealPath(to), ec);
00136             return (!ec);
00137         }
00138     virtual std::vector<std::filesystem::path> GetAll() override
00139         {
00140             std::vector<std::filesystem::path> list;
00141             std::error_code ec;
00142             for (auto it = std::filesystem::directory_iterator(m_preset_path.c_str(),
      std::filesystem::directory_options::skip_permission_denied, ec);
00143                 !ec && it != std::filesystem::directory_iterator{};
00144                  it.increment(ec) )
00145             {
00146                 if (it->is_regular_file(ec)) {
00147                     list.push_back(it->path().lexically_relative(m_preset_path));
00148                 }
00149             }
00150             return list;
00151         }
00152     virtual std::string Read(const std::filesystem::path &filename) override
00153         {
00154             std::ifstream f(GetRealPath(filename));
00155             if ( f.fail() ) {
00156                 return std::string();
00157             }
00158             std::ostringstream ss;
00159             ss « f.rdbuf();
00160             return ss.str();
00161         }
00162     virtual bool Write(const std::filesystem::path &filename, const std::string &content) override
00163         {
00164             std::ofstream f(GetRealPath(filename));
00165             if ( f.fail() ) {
00166                 return false;
00167             }
00168             f « content;
00169             if ( f.fail() ) {
00170                 return false;
00171             }
00172             return true;
00173         }
00174 private:
00175     std::filesystem::path m_preset_path;
00176     std::filesystem::path GetRealPath(const std::filesystem::path &name)
00177         {
00178             return m_preset_path / name;
00179         }
00180 };
00182
00183 int main(int argc, const char **argv)
00184 {
00185     int listen_port = 8080;
00186     auto store_path_str=GetEnv("S4STORE_PATH");
00188     std::string libname =  "libsound4.bigvoice.cl.so";
00190
00191
00192     // Get credentials from environment
00193     auto LoginKey=GetEnv("S4LOGINKEY");
00194     if (LoginKey.empty()) {
00195         LoginKey=GetEnv("S4USERID");
00196     }
00197     auto RadioName=GetEnv("RADIO_NAME");
00198     auto KeyId=GetEnv("ACCESS_KEY_ID");
00199     auto AccessKey=GetEnv("ACCESS_KEY_SECRET");
00200
00201     const char *progname=argv[0];
00202     for (int n=1;n<argc;n++) {
00203         std::string arg = argv[n];
00204         if (arg=="-p" || arg=="--port") {
00205             if (argc<n+1) {
00206                 std::cerr « "--port needs a number"« std::endl;
00207                 return 1;
00208             }
00209             n++;
00210             listen_port=atoi(argv[n]);
00211         } else if (arg=="-l" || arg=="--lib") {
00212             if (argc<n+1) {
00213                 std::cerr « "--port needs a number"« std::endl;
00214                 return 1;
00215             }
00216             n++;
```

```
00217             libname=argv[n];
00218         } else if (arg=="-h" || arg=="--help") {
00219             std::cerr « "Usage: " « progname « " [-p port] [-l libname.so]"«std::endl;;
00220             std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00221             std::cerr « "   -l, --lib   : library to load ["«libname«"]" « std::endl;
00222             std::cerr « "   -p, --port  : port for web server ["«listen_port«"]" « std::endl;
00223             std::cerr « std::endl;
00224             std::cerr « "Environment:"« std::endl;;
00225             std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
      std::endl;;
00226             std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00227             std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00228             std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00229             std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;
00230             std::cerr « std::endl;
00231             std::cerr « "Example:"« std::endl;
00232             std::cerr « "  ffmpeg -re -i file.wav -ar 48000 -ac 1 -f f32le - | "«progname«" | ffplay
      -f f32le -ar 48000 -ac 1 -i -" « std::endl;;
00233             #if defined(_WIN32) || defined(WIN32)
00234             std::cerr « "  ffmpeg -f dshow -i audio=\"Microphone\" -ar 48000 -ac 1 -f f32le - |
      "«progname«" | ffplay -f f32le -ar 48000 -ac 1 -i -" « std::endl;;
00235             #else
00236             std::cerr « "  ffmpeg -f pulse -i default -ar 48000 -ac 1 -f f32le - | "«progname«" |
      ffplay -f f32le -ar 48000 -ac 1 -i -" « std::endl;;
00237             #endif
00238             return 0;
00239         } else {
00240             std::cerr « "unknown argument '"«arg«"'"« std::endl;
00241             return 1;
00242         }
00243     }
00244
00245
00247     sound4::CProcessor processor;
00248     if (!processor.Load(libname)) {
00249         std::cerr « "Failed to load processing library" « std::endl;
00250         return 1;
00251     }
00253
00254     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00255         std::cerr « "Missing credentials in environment" « std::endl;
00256         return 1;
00257     }
00258     std::filesystem::path store_path;
00259     if (!store_path_str.empty()) {
00260         store_path = std::filesystem::path(store_path_str);
00261         std::error_code ec;
00262         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
      {
00263             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
      std::endl;
00264             return 1;
00265         }
00266     }
00267
00269     processor.SetLogSeverity(sound4::verbose5);
00270     processor.SetLoggerCallback([](sound4::LogSeverity severity,const std::string& msg) {
00271             std::cerr « msg « std::endl;
00272         });
00273
00274     sound4::CInstance instance(processor);
00275     instance.SetParam("ADMIN_USER","admin");
00276     instance.SetParam("ADMIN_SECRET","admin");
00278
00280     MyPresetManager preset_manager;
00281     if (!store_path.empty()) {
00282         if (!preset_manager.SetupPath(store_path)) {
00283             std::cerr « "Failed to setup the custom preset manager on path " « store_path.c_str() «
      std::endl;
00284             return 1;
00285         }
00286         instance.SetPresetManager(&preset_manager);
00287     }
00289
00290     std::cerr « "Creating" « std::endl;
00292     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00293         std::cerr « "Failed to create the processing instance" « std::endl;
00294         return 1;
00295     }
00297
00298     std::cerr « "Starting web server" « std::endl;
00300     if (!instance.StartWebServer(listen_port,0)) {
00301         std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
00302         return 1;
00303     }
00305
00306     {
```

```
00308         auto client = instance.NewClient();
00309         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00310         std::cerr « "Request for processor name returned "«answer«std::endl;
00312     }
00313
00314     ProcessLoop(instance, processor);
00315
00316     std::cerr « "Stopping now" « std::endl;
00317     instance.Stop();
00318
00319     std::cerr « "End" « std::endl;
00320     return 0;
00321 }
```

## 15.23 sound4.bigvoice.cl.h File Reference

C interface for library.

```
#include <windows.h>
#include <stdint.h>
```

### Classes

- struct SOUND4_ProcessInfo
- struct SOUND4_AuxiliaryOutputInfos

### 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 struct _tlgProvider_t const * TraceLoggingHProvider
- 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 struct SOUND4_ProcessInfo * SOUND4_GetProcessInfo ()
- 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 *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct bigvoice_CInstance * bigvoice_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct bigvoice_CParameters *parameters)
- struct bigvoice_CInstance * bigvoice_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_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_SetInstanceTracing (struct bigvoice_CParameters *params, TraceLoggingHProvider tracing_provider, GUID activity_guid)
- void bigvoice_SetInstanceTracingProcessActivity (struct bigvoice_CInstance *instance, GUID activity_guid)
- 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)

- void bigvoice_SetMetadataMulti (struct bigvoice_CInstance *instance, const char **keyvalue)
- 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)
- struct SOUND4_AuxiliaryOutputInfos ** bigvoice_GetAuxiliaryAudioOutputInfos (struct bigvoice_CInstance *instance)
- void bigvoice_FreeAuxiliaryAudioInfos (struct SOUND4_AuxiliaryOutputInfos **infos)
- int bigvoice_ActivateAuxiliaryAudioID (struct bigvoice_CInstance *instance, const char *id)
- unsigned int bigvoice_GetAuxiliaryAudioEstimatedDelay (struct bigvoice_CInstance *instance, const char *id)
- 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)
- void bigvoice_Webserver_SetAppHealth (struct bigvoice_CInstance *instance, int httpcode, const char *contenttype, const char *content)
- void bigvoice_Webserver_GetAppHealth (struct bigvoice_CInstance *instance, int *httpcode, char **contenttype, char **content)
- void bigvoice_Webserver_FreeString (char *str)

### 15.23.1 Macro Definition Documentation

#### 15.23.1.1 BIGVOICE_DECL

```
#define BIGVOICE_DECL __declspec(dllimport)
```

Definition at line 14 of file sound4.bigvoice.cl.h.

#### 15.23.1.2 BIGVOICE_HAS_CLOUDBUS

```
#define BIGVOICE_HAS_CLOUDBUS 1
```

Definition at line 26 of file sound4.bigvoice.cl.h.

#### 15.23.1.3 BIGVOICE_HAS_WEBSERVER

```
#define BIGVOICE_HAS_WEBSERVER 1
```

Definition at line 25 of file sound4.bigvoice.cl.h.

### 15.23.2 Typedef Documentation

#### 15.23.2.1 TraceLoggingHProvider

```
typedef struct _tlgProvider_t const* TraceLoggingHProvider
```

Definition at line 418 of file sound4.bigvoice.cl.h.

## 15.24 sound4.bigvoice.cl.h

Go to the documentation of this file.

```
00001
00004 #pragma once
00005
00006 #ifdef __unix__
00007     #include <sys/types.h>
00008     #define BIGVOICE_DECL       __attribute__ ((visibility ("default")))
00009 #elif defined(_WIN32) || defined(WIN32)
00010     #include <windows.h>
00011     #ifdef BUILDING_BIGVOICE_DLL
00012         #define BIGVOICE_DECL   __declspec(dllexport)
00013     #else
00014         #define BIGVOICE_DECL   __declspec(dllimport)
00015     #endif
00016     #ifdef _MSC_VER
00017         #pragma warning(disable : 4996)
00018     #endif
00019 #elif defined(__APPLE__)
00020     #include <sys/types.h>
00021     #define BIGVOICE_DECL       __attribute__ ((visibility ("default")))
00022 #endif
00023 #include <stdint.h>
00024
00025 #define BIGVOICE_HAS_WEBSERVER 1
00026 #define BIGVOICE_HAS_CLOUDBUS 1
00027
00028 #ifdef __cplusplus
00029 extern "C" {
```

```
00030 #endif // __cplusplus
00031
00032 // ------------------------------------------------------------
00046 #define BIGVOICE_AUDIOFRAME_COUNT      12
00052 #define BIGVOICE_CHANNEL_COUNT          1
00058 #define BIGVOICE_AUDIO_INPUT_COUNT      1
00064 #define BIGVOICE_AUDIO_OUTPUT_COUNT     1
00070 #define BIGVOICE_SAMPLE_RATE            48000
00071
00077 struct SOUND4_ProcessInfo {
00078     size_t struct_size;
00079     const char *name;
00080     const char *prefix;
00081     const char *shortname;
00082     const char *version;
00083 };
00084
00092 BIGVOICE_DECL const struct SOUND4_ProcessInfo* SOUND4_GetProcessInfo();
00093
00099 BIGVOICE_DECL const char *bigvoice_GetVersion();
00100
00113 BIGVOICE_DECL unsigned int bigvoice_GetChunkSizeInFrames();
00114
00123 BIGVOICE_DECL unsigned int* bigvoice_GetPossibleChunkSizeInFrames();
00124
00132 BIGVOICE_DECL unsigned int bigvoice_GetChannelCount();
00133
00142 BIGVOICE_DECL unsigned int bigvoice_GetAudioInputCount();
00143
00152 BIGVOICE_DECL unsigned int bigvoice_GetAudioOutputCount();
00153
00161 BIGVOICE_DECL unsigned int bigvoice_GetSampleRate();
00162
00166 enum bigvoice_LogSeverity {
00167     none     = 0,
00168     fatal    = 1,
00169     error    = 2,
00170     warning  = 3,
00171     info     = 4,
00172     verbose  = 5,
00173     verbose2 = 6,
00174     verbose3 = 7,
00175     verbose4 = 8,
00176     verbose5 = 9,
00177 };
00178
00185 typedef void (*bigvoice_loggerfn) (enum bigvoice_LogSeverity, const char *);
00186
00196 BIGVOICE_DECL void bigvoice_SetLoggerCallback( bigvoice_loggerfn logger );
00197
00205 BIGVOICE_DECL void bigvoice_SetLogSeverity( enum bigvoice_LogSeverity severity );
00206
00209 // ------------------------------------------------------------
00217 // ------------------------------------------------------------
00221 struct bigvoice_CInstance;
00222
00226 struct bigvoice_CParameters;
00227
00235 BIGVOICE_DECL struct bigvoice_CParameters *bigvoice_NewParameters();
00236
00240 BIGVOICE_DECL void bigvoice_FreeParameters(struct bigvoice_CParameters *params);
00241
00253 BIGVOICE_DECL void bigvoice_SetParameter(struct bigvoice_CParameters *params, const char *name, const
      char *value);
00254
00260 BIGVOICE_DECL const char *bigvoice_GetParameter(struct bigvoice_CParameters *params, const char
      *name);
00261
00267 BIGVOICE_DECL void bigvoice_FreeParameterValue(const char *value);
00268
00276 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess(
00277         const char *LoginKey,
00278         const char *RadioName,
00279         const char *Access_Key_ID,
00280         const char *Access_Key_Secret,
00281         const char *save_path
00282     );
00283
00291 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess2(
00292         const char *LoginKey,
00293         const char *RadioName,
00294         const char *Access_Key_ID,
00295         const char *Access_Key_Secret,
00296         const char *save_path,
00297         const struct bigvoice_CParameters *parameters
00298     );
00299
```

```
00327 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess3(
00328         const char *LoginKey,
00329         const char *RadioName,
00330         const char *Access_Key_ID,
00331         const char *Access_Key_Secret,
00332         const char *save_path,
00333         const struct bigvoice_CParameters *parameters,
00334         unsigned int frames_per_chunk
00335     );
00336
00348 BIGVOICE_DECL unsigned int bigvoice_GetProcessChunkFrames(struct bigvoice_CInstance *instance);
00349
00359 BIGVOICE_DECL int bigvoice_TerminateProcess(struct bigvoice_CInstance *instance);
00360
00366 BIGVOICE_DECL void bigvoice_ExitProcess(struct bigvoice_CInstance *instance);
00367
00370 #if BIGVOICE_HAS_CLOUDBUS
00371 // ---------------------------------------------------------------
00381 struct bigvoice_CBus;
00382
00388 BIGVOICE_DECL struct bigvoice_CBus* bigvoice_NewBus();
00389
00395 BIGVOICE_DECL void bigvoice_FreeBus(struct bigvoice_CBus *bus);
00396
00406 BIGVOICE_DECL void bigvoice_SetInstanceBus(struct bigvoice_CParameters *params, struct bigvoice_CBus
      *bus);
00407
00408
00410 #endif // BIGVOICE_HAS_CLOUDBUS
00411
00412 // ---------------------------------------------------------------
00413 //   Tracing (Windows only)
00414 // ---------------------------------------------------------------
00415 #ifdef _WIN32
00416 #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00417     struct _tlgProvider_t;
00418     typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00419 #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00420 // ---------------------------------------------------------------
00435 BIGVOICE_DECL void bigvoice_SetInstanceTracing(struct bigvoice_CParameters *params,
      TraceLoggingHProvider tracing_provider, GUID activity_guid);
00436
00444 BIGVOICE_DECL void bigvoice_SetInstanceTracingProcessActivity(struct bigvoice_CInstance *instance,
      GUID activity_guid);
00445
00447 #endif // _WIN32
00448
00449 // ---------------------------------------------------------------
00472 // ---------------------------------------------------------------
00473
00474 // OS dependent filesystem path char type
00475 #ifdef _WIN32
00476     typedef wchar_t fs_char;
00477 #else // !_WIN32
00478     typedef char fs_char;
00479 #endif // !_WIN32
00480
00488 typedef char * (*bigvoice_storage_reader) (const fs_char *filename, void* userdata);
00489
00496 typedef void (*bigvoice_storage_reader_free) (char *content, void* userdata);
00497
00506 typedef int (*bigvoice_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00507
00515 typedef int (*bigvoice_storage_exists) (const fs_char *filename, void* userdata);
00516
00524 typedef fs_char** (*bigvoice_storage_getall) (void* userdata);
00525
00532 typedef void (*bigvoice_storage_getall_free) (fs_char**all, void* userdata);
00533
00541 typedef int (*bigvoice_storage_remove) (const fs_char *filename, void* userdata);
00542
00551 typedef int (*bigvoice_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00552
00570 BIGVOICE_DECL void bigvoice_SetPresetManager(struct bigvoice_CParameters *params
00571     , bigvoice_storage_reader reader, bigvoice_storage_reader_free
00572     , bigvoice_storage_writer writer
00573     , bigvoice_storage_exists exists
00574     , bigvoice_storage_getall getall, bigvoice_storage_getall_free getall_free
00575     , bigvoice_storage_remove remove
00576     , bigvoice_storage_rename rename
00577     , int IsReadOnly
00578     , void* userdata
00579     );
00580
00584 enum bigvoice_PresetChange_Kind {
00585     change_kind_created = 1,
00586     change_kind_modified   ,
```

```
00587     change_kind_deleted    ,
00588 };
00589
00598 BIGVOICE_DECL void bigvoice_PresetManager_InformChange(struct bigvoice_CInstance *instance, const
      fs_char *relative_path, enum bigvoice_PresetChange_Kind change_kind);
00599
00602 // ------------------------------------------------------------
00610 // ------------------------------------------------------------
00611
00628 BIGVOICE_DECL void bigvoice_SetMetadata(struct bigvoice_CInstance *instance, const char* key, const
      char* value);
00629
00640 BIGVOICE_DECL void bigvoice_SetMetadataMulti(struct bigvoice_CInstance *instance, const char**
      keyvalue);
00641
00649 BIGVOICE_DECL const char** bigvoice_GetMetadataInfos(struct bigvoice_CInstance *instance);
00650
00657 BIGVOICE_DECL void bigvoice_FreeMetadataInfos(struct bigvoice_CInstance *instance, const char**
      infos);
00658
00661 // ------------------------------------------------------------
00668 // ------------------------------------------------------------
00669
00682 BIGVOICE_DECL void bigvoice_StartUpdateThread(struct bigvoice_CInstance *instance, unsigned int port);
00683
00689 BIGVOICE_DECL void bigvoice_StopUpdateThread(struct bigvoice_CInstance *instance);
00690
00698 BIGVOICE_DECL int bigvoice_WaitUpdateThreadReady(struct bigvoice_CInstance *instance, int
      milliseconds);
00699
00702 // ------------------------------------------------------------
00709 // ------------------------------------------------------------
00731 BIGVOICE_DECL void bigvoice_ProcessAudio(struct bigvoice_CInstance *instance, const float *input,
      float *output );
00732
00751 BIGVOICE_DECL void bigvoice_ProcessAudio_Planar(struct bigvoice_CInstance *instance
00752     , float const * const *input
00753     , float * const *output);
00754
00767 BIGVOICE_DECL float *bigvoice_GetBufferIn(struct bigvoice_CInstance *instance);
00768
00781 BIGVOICE_DECL float *bigvoice_GetBufferOut(struct bigvoice_CInstance *instance);
00782
00792 BIGVOICE_DECL unsigned int bigvoice_GetEstimatedDelay(struct bigvoice_CInstance *instance);
00793
00796 // ------------------------------------------------------------
00806 // ------------------------------------------------------------
00810 struct SOUND4_AuxiliaryOutputInfos {
00811     char *id;
00812     char *description;
00813     unsigned int sample_rate;
00814     unsigned int channel_count;
00815     const float *buffer;
00816 };
00817
00825 BIGVOICE_DECL struct SOUND4_AuxiliaryOutputInfos** bigvoice_GetAuxiliaryAudioOutputInfos(struct
      bigvoice_CInstance *instance);
00826
00832 BIGVOICE_DECL void bigvoice_FreeAuxiliaryAudioInfos(struct SOUND4_AuxiliaryOutputInfos** infos);
00833
00843 BIGVOICE_DECL int bigvoice_ActivateAuxiliaryAudioID(struct bigvoice_CInstance *instance, const char*
      id);
00844
00856 BIGVOICE_DECL unsigned int bigvoice_GetAuxiliaryAudioEstimatedDelay(struct bigvoice_CInstance
      *instance, const char* id);
00857
00860 // ------------------------------------------------------------
00869 // ------------------------------------------------------------
00870     #ifdef _MSC_VER
00871         #define __BYTE_ORDER__  REG_DWORD
00872         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00873         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00874     #endif
00875
00879 enum bigvoice_SampleFormat {
00880     INVALID_FORMAT=0,
00881     S16_LE,
00882     S16_BE,
00883     S24_LE,
00884     S24_BE,
00885     S32_LE,
00886     S32_BE,
00887     F32_LE,
00888     F32_BE,
00889
00890     // Native definitions, only links to the physical ones
00891     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
```

```
00892         S16_NATIVE=S16_LE,
00893         S24_NATIVE=S24_LE,
00894         S32_NATIVE=S32_LE,
00895         F32_NATIVE=F32_LE,
00896     #else
00897         S16_NATIVE=S16_BE,
00898         S24_NATIVE=S24_BE,
00899         S32_NATIVE=S32_BE,
00900         F32_NATIVE=F32_BE,
00901     #endif
00902 };
00903
00910 BIGVOICE_DECL const char *bigvoice_GetFormatName(const enum bigvoice_SampleFormat fmt);
00911
00918 BIGVOICE_DECL enum bigvoice_SampleFormat bigvoice_GetFormatFromName(const char *name);
00919
00926 BIGVOICE_DECL unsigned int bigvoice_GetBytesFromFormat(const enum bigvoice_SampleFormat fmt);
00927
00928
00935 BIGVOICE_DECL unsigned int bigvoice_GetMaxPacketFrame(struct bigvoice_CInstance *instance);
00936
00947 BIGVOICE_DECL unsigned int bigvoice_AddAudio(struct bigvoice_CInstance *instance, const uint8_t
      *payload, unsigned int nFrame, enum bigvoice_SampleFormat fmt );
00948
00955 BIGVOICE_DECL unsigned int bigvoice_AddPadAudio(struct bigvoice_CInstance *instance);
00956
00963 BIGVOICE_DECL unsigned int bigvoice_GetOutputCount(struct bigvoice_CInstance *instance);
00964
00974 BIGVOICE_DECL unsigned int bigvoice_GetAudio(struct bigvoice_CInstance *instance, uint8_t *payload,
      unsigned int max_nFrame, enum bigvoice_SampleFormat fmt );
00975
00979 // ------------------------------------------------------------
00986 // ------------------------------------------------------------
00997 BIGVOICE_DECL void bigvoice_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      bigvoice_SampleFormat fmt);
00998
01009 BIGVOICE_DECL void bigvoice_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      bigvoice_SampleFormat fmt);
01010
01020 BIGVOICE_DECL void bigvoice_StereoToMono(const float *input, float *output, size_t nFrame);
01021
01031 BIGVOICE_DECL void bigvoice_MonoToStereo(const float *input, float *output, size_t nFrame);
01032
01043 BIGVOICE_DECL void bigvoice_StereoToMono_Planar(const float *inputL, const float *inputR, float
      *output, size_t nFrame);
01044
01055 BIGVOICE_DECL void bigvoice_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame);
01056
01063 BIGVOICE_DECL void bigvoice_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
01064
01071 BIGVOICE_DECL void bigvoice_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
01072
01075 // ------------------------------------------------------------
01082 // ------------------------------------------------------------
01086 struct bigvoice_CClientInstance;
01087
01096 BIGVOICE_DECL struct bigvoice_CClientInstance *bigvoice_NewClient(struct bigvoice_CInstance
      *instance);
01097
01103 BIGVOICE_DECL void bigvoice_DeleteClient(struct bigvoice_CClientInstance *client);
01104
01118 BIGVOICE_DECL const char *bigvoice_ProcessJson(struct bigvoice_CClientInstance *client, const char
      *json_str, int *need_save);
01119
01125 BIGVOICE_DECL void bigvoice_FreeJsonAnswer(const char *json_str);
01126
01138 BIGVOICE_DECL int bigvoice_SaveState(struct bigvoice_CInstance *instance);
01139
01142 #if BIGVOICE_HAS_WEBSERVER
01143 // ------------------------------------------------------------
01161 // ------------------------------------------------------------
01169 BIGVOICE_DECL uint64_t bigvoice_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port);
01170
01171 // ------------------------------------------------------------
01191 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);
01192
01210 BIGVOICE_DECL uint64_t bigvoice_Webserver(unsigned int listenport, unsigned int listenport_secure,
      struct bigvoice_CInstance *instance);
01211
01221 BIGVOICE_DECL int bigvoice_Webserver_Stop(uint64_t id, int timeout_ms);
01222
01231 BIGVOICE_DECL int bigvoice_Webserver_Status(uint64_t id);
01232
```

```
01243 BIGVOICE_DECL void bigvoice_Webserver_SetAppHealth(struct bigvoice_CInstance *instance, int httpcode,
      const char *contenttype, const char *content);
01244
01253 BIGVOICE_DECL void bigvoice_Webserver_GetAppHealth(struct bigvoice_CInstance *instance, int* httpcode,
      char** contenttype, char** content);
01254
01260 BIGVOICE_DECL void bigvoice_Webserver_FreeString(char *str);
01261
01266 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
01267
01271 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
01275 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
01276
01278 #endif // BIGVOICE_HAS_WEBSERVER
01279
01280 #ifdef __cplusplus
01281 } // extern "C"
01282 #endif // __cplusplus
```

## 15.25 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::AuxiliaryOutputInfos
- struct sound4::bigvoice::helper::SampleFormat< T >

  *Helper for Sample format types.*
- class sound4::bigvoice::CBus

  *Shared Bus.*
- class sound4::bigvoice::CPresetLoader

  *Custom preset handler helper.*
- class sound4::bigvoice::CInstance

  *Instance handling class.*
- class sound4::bigvoice::CInstance::CClient

### Namespaces

- namespace sound4
- namespace sound4::bigvoice
- namespace sound4::bigvoice::helper

### 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

### Functions

- static char * bigvoice_custom_reader (const fs_char *filename, void *userdata)
- static void bigvoice_custom_reader_free (char *content, void *userdata)
- static int bigvoice_custom_writer (const fs_char *filename, const char *content, void *userdata)
- static int bigvoice_custom_exists (const fs_char *filename, void *userdata)
- static fs_char ** bigvoice_custom_getall (void *userdata)
- static void bigvoice_custom_getall_free (fs_char **all, void *userdata)
- static int bigvoice_custom_remove (const fs_char *filename, void *userdata)
- static int bigvoice_custom_rename (const fs_char *from, const fs_char *to, void *userdata)
- static std::string sound4::bigvoice::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::bigvoice::helper::UTF8ToWString (const std::string &str)
- static void sound4::bigvoice::helper::AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, bigvoice_SampleFormat fmt)
- static void sound4::bigvoice::helper::AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, bigvoice_SampleFormat fmt)
- static void sound4::bigvoice::helper::StereoToMono (const float *input, float *output, size_t nFrame)
- static void sound4::bigvoice::helper::MonoToStereo (const float *input, float *output, size_t nFrame)
- static void sound4::bigvoice::helper::StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- static void sound4::bigvoice::helper::MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- static void sound4::bigvoice::helper::AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- static void sound4::bigvoice::helper::AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- static std::string sound4::bigvoice::GetVersion ()
- static unsigned int sound4::bigvoice::GetChunkSizeInFrames ()
- static std::vector< unsigned int > sound4::bigvoice::GetPossibleChunkSizeInFrames ()
- static unsigned int sound4::bigvoice::GetChannelCount ()
- static unsigned int sound4::bigvoice::GetAudioInputCount ()
- static unsigned int sound4::bigvoice::GetAudioOutputCount ()
- static unsigned int sound4::bigvoice::GetSampleRate ()
- static void sound4::bigvoice::SanityCheck (bool a_bCheckFrames=true)
- static std::string sound4::bigvoice::GetFormatName (const SampleFormat fmt)
- static SampleFormat sound4::bigvoice::GetFormatFromName (const std::string &name)
- static unsigned int sound4::bigvoice::GetBytesFromFormat (const SampleFormat fmt)
- static void sound4::bigvoice::_log_cb_c (bigvoice_LogSeverity severity, const char *c_msg)
- static void sound4::bigvoice::SetLogSeverity (LogSeverity severity)
- static void sound4::bigvoice::SetLoggerCallback (log_cb_t cb)

### Variables

- static constexpr const char * sound4::bigvoice::process_name = "SOUND4 BIGVOICE.CL"
- static constexpr const char * sound4::bigvoice::process_shortname = "bigvoice"
- const size_t sound4::bigvoice::ChunkFrames = 12
- const size_t sound4::bigvoice::InputChannels = 1 * 1
- const size_t sound4::bigvoice::OutputChannels = 1 * 1
- const size_t sound4::bigvoice::InputSampleSize =ChunkFrames*InputChannels
- const size_t sound4::bigvoice::OutputSampleSize =ChunkFrames*OutputChannels
- const size_t sound4::bigvoice::SampleRate = 48000
- static log_cb_t sound4::bigvoice::_log_cb

## 15.25.1 Function Documentation

### 15.25.1.1 bigvoice_custom_exists()

```
static int bigvoice_custom_exists (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1117 of file sound4.bigvoice.cl.hpp.

### 15.25.1.2 bigvoice_custom_getall()

```
static fs_char ** bigvoice_custom_getall (
            void * userdata )  [static]
```

Definition at line 1122 of file sound4.bigvoice.cl.hpp.

### 15.25.1.3 bigvoice_custom_getall_free()

```
static void bigvoice_custom_getall_free (
            fs_char ** all,
            void * userdata )  [static]
```

Definition at line 1132 of file sound4.bigvoice.cl.hpp.

### 15.25.1.4 bigvoice_custom_reader()

```
static char * bigvoice_custom_reader (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1103 of file sound4.bigvoice.cl.hpp.

### 15.25.1.5 bigvoice_custom_reader_free()

```
static void bigvoice_custom_reader_free (
            char * content,
            void * userdata )  [static]
```

Definition at line 1108 of file sound4.bigvoice.cl.hpp.

### 15.25.1.6 bigvoice_custom_remove()

```
static int bigvoice_custom_remove (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1139 of file sound4.bigvoice.cl.hpp.

#### 15.25.1.7 bigvoice_custom_rename()

```
static int bigvoice_custom_rename (
            const fs_char * from,
            const fs_char * to,
            void * userdata )  [static]
```

Definition at line 1144 of file sound4.bigvoice.cl.hpp.

#### 15.25.1.8 bigvoice_custom_writer()

```
static int bigvoice_custom_writer (
            const fs_char * filename,
            const char * content,
            void * userdata )  [static]
```

Definition at line 1112 of file sound4.bigvoice.cl.hpp.

## 15.26 sound4.bigvoice.cl.hpp

Go to the documentation of this file.

```
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     #ifdef _MSC_VER
00027         #pragma warning(disable : 4996)
00028     #endif
00029 #else // !_WIN32
00031     #define fs_strdup strdup
00032 #endif // !_WIN32
00033
00034 // bridge C callbacks to CPresetLoader
00035 extern "C" {
00036     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata);
00037     static void bigvoice_custom_reader_free(char *content, void* userdata);
00038     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata);
00039     static int bigvoice_custom_exists(const fs_char *filename, void* userdata);
00040     static fs_char** bigvoice_custom_getall(void* userdata);
00041     static void bigvoice_custom_getall_free(fs_char** all, void* userdata);
00042     static int bigvoice_custom_remove(const fs_char *filename, void* userdata);
00043     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00044 };
00045
00049 namespace sound4 {
00050     struct AuxiliaryOutputInfos {
00051         std::string id;
00052         std::string description;
00053         unsigned int sample_rate;
00054         unsigned int channel_count;
00055         const float *buffer;
00056     };
```

```
00060 namespace bigvoice {
00061     static constexpr const char *process_name = "SOUND4 BIGVOICE.CL";
00062     static constexpr const char *process_shortname = "bigvoice";
00063     using SampleFormat = bigvoice_SampleFormat;
00068     const size_t ChunkFrames=BIGVOICE_AUDIOFRAME_COUNT;
00073     const size_t InputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT;
00078     const size_t OutputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT;
00083     const size_t InputSampleSize=ChunkFrames*InputChannels;
00088     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00093     const size_t SampleRate=BIGVOICE_SAMPLE_RATE;
00094
00098     namespace helper {
00099         #ifdef _WIN32
00105         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00106             if (wstr.empty()) return {};
00107             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00108             std::string str(size, 0);
00109             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00110             return str;
00111         }
00117         static inline std::wstring UTF8ToWString(const std::string& str) {
00118             if (str.empty()) return std::wstring();
00119             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00120             std::wstring wstr(size, 0);
00121             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00122             return wstr;
00123         }
00124         #endif // _WIN32
00125
00131             template<typename T>
00132         struct SampleFormat {
00133         };
00134             template<>
00135         struct SampleFormat<int16_t> {
00136             const bigvoice_SampleFormat format = S16_NATIVE;
00137         };
00138             template<>
00139         struct SampleFormat<int32_t> {
00140             const bigvoice_SampleFormat format = S32_NATIVE;
00141         };
00142             template<>
00143         struct SampleFormat<float> {
00144             const bigvoice_SampleFormat format = F32_NATIVE;
00145         };
00146
00147         // ------------------------------------------------------------
00158         static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl,
      bigvoice_SampleFormat fmt) {
00159             bigvoice_AudioConvertFrom(payload, output, nSpl, fmt);
00160         }
00161
00172         static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl,
      bigvoice_SampleFormat fmt) {
00173             bigvoice_AudioConvertTo(input, payload, nSpl, fmt);
00174         }
00175
00185         static inline void StereoToMono(const float *input, float *output, size_t nFrame) {
00186             bigvoice_StereoToMono(input, output, nFrame);
00187         }
00188
00198         static inline void MonoToStereo(const float *input, float *output, size_t nFrame) {
00199             bigvoice_MonoToStereo(input, output, nFrame);
00200         }
00201
00212         static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float
      *output, size_t nFrame) {
00213             bigvoice_StereoToMono_Planar(inputL, inputR, output, nFrame);
00214         }
00215
00226         static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame) {
00227             bigvoice_MonoToStereo_Planar(input, outputL, outputR, nFrame);
00228         }
00229
00236         static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
00237             bigvoice_AudioMonoFromLiveStereo(payload, output);
00238         }
00239
00246         static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
00247             bigvoice_AudioMonoToLiveStereo(input, payload);
00248         }
00249
00250     }; // namespace helper
00251
00257     static inline std::string GetVersion()
00258         {
```

```
00259             return std::string(bigvoice_GetVersion());
00260         }
00261
00272     static inline unsigned int GetChunkSizeInFrames()
00273         {
00274             return bigvoice_GetChunkSizeInFrames();
00275         }
00276
00285     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00286         {
00287             std::vector<unsigned int> list;
00288             for (unsigned int* src=bigvoice_GetPossibleChunkSizeInFrames();*src;src++) {
00289                 list.push_back(*src);
00290             }
00291             return list;
00292         }
00293
00299     static inline unsigned int GetChannelCount()
00300         {
00301             return bigvoice_GetChannelCount();
00302         }
00303
00311     static inline unsigned int GetAudioInputCount()
00312         {
00313             return bigvoice_GetAudioInputCount();
00314         }
00315
00323     static inline unsigned int GetAudioOutputCount()
00324         {
00325             return bigvoice_GetAudioOutputCount();
00326         }
00327
00335     static inline unsigned int GetSampleRate()
00336         {
00337             return bigvoice_GetSampleRate();
00338         }
00339
00345     static inline void SanityCheck(bool a_bCheckFrames = true)
00346     {
00347         // Do some sanity checks
00348         if (bigvoice_GetSampleRate()!=BIGVOICE_SAMPLE_RATE) {
00349             throw std::runtime_error("Bad library sampling rate");
00350         }
00351         if (a_bCheckFrames && bigvoice_GetChunkSizeInFrames()!=ChunkFrames) {
00352             throw std::runtime_error("Bad library frame size");
00353         }
00354         if (bigvoice_GetChannelCount()!=BIGVOICE_CHANNEL_COUNT) {
00355             throw std::runtime_error("Bad library channel count");
00356         }
00357         if (bigvoice_GetAudioInputCount()!=BIGVOICE_AUDIO_INPUT_COUNT) {
00358             throw std::runtime_error("Bad library input count");
00359         }
00360         if (bigvoice_GetAudioOutputCount()!=BIGVOICE_AUDIO_OUTPUT_COUNT) {
00361             throw std::runtime_error("Bad library output count");
00362         }
00363     }
00364
00371     static inline std::string GetFormatName(const SampleFormat fmt)
00372         {
00373             return std::string(bigvoice_GetFormatName(fmt));
00374         }
00375
00382     static inline SampleFormat GetFormatFromName(const std::string& name)
00383         {
00384             return bigvoice_GetFormatFromName(name.c_str());
00385         }
00386
00393     static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
00394         {
00395             return bigvoice_GetBytesFromFormat(fmt);
00396         }
00397
00401     using LogSeverity = bigvoice_LogSeverity;
00405     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00406
00410     static log_cb_t _log_cb;
00411     extern "C" {
00412         static inline void _log_cb_c(bigvoice_LogSeverity severity, const char *c_msg) {
00413             _log_cb(severity, std::string(c_msg));
00414         }
00415     }
00419     static inline void SetLogSeverity(LogSeverity severity)
00420         {
00421             bigvoice_SetLogSeverity(severity);
00422         }
00426     static inline void SetLoggerCallback(log_cb_t cb)
00427         {
```

```
00428             _log_cb=cb;
00429             bigvoice_SetLoggerCallback(&_log_cb_c);
00430         }
00431
00432 #if BIGVOICE_HAS_CLOUDBUS
00438     class CBus {
00439         public:
00440             CBus()
00441                 : m_bus(bigvoice_NewBus()) {}
00442             ~CBus()
00443                 {
00444                     bigvoice_FreeBus(m_bus);
00445                     m_bus = nullptr;
00446                 }
00447             bigvoice_CBus *Get() const { return m_bus; }
00448         private:
00449             bigvoice_CBus *m_bus;
00450     };
00451 #endif // BIGVOICE_HAS_CLOUDBUS
00452
00458     class CPresetLoader {
00459     public:
00460         CPresetLoader() = default;
00461         virtual ~CPresetLoader() = default;
00462
00463         virtual bool IsReadOnly() = 0;
00464         virtual bool Exists(const std::filesystem::path &name) = 0;
00465         virtual bool Remove(const std::filesystem::path &name) = 0;
00466         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00467         virtual std::vector<std::filesystem::path> GetAll() = 0;
00468         virtual std::string Read(const std::filesystem::path &filename) = 0;
00469         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00470     };
00471     using PresetChange_Kind = bigvoice_PresetChange_Kind;
00472
00478     class CInstance {
00479     public:
00483         CInstance()
00484             {
00485                 params = bigvoice_NewParameters();
00486                 if (!params) throw std::bad_alloc();
00487             }
00488         // non-copyable
00489         CInstance( const CInstance& ) = delete;
00490         CInstance& operator=( const CInstance& ) = delete;
00494         virtual ~CInstance()
00495             {
00496                 Stop();
00497                 if (params) bigvoice_FreeParameters(params);
00498             }
00502        bool IsOk() const { return instance!=nullptr; }
00512         void SetParam(const std::string& name, const std::string& value)
00513             {
00514                 assert(params);
00515                 bigvoice_SetParameter(params, name.c_str(), value.c_str());
00516             }
00523         std::string GetParam(const std::string& name)
00524             {
00525                 assert(params);
00526                 auto c_value=bigvoice_GetParameter(params, name.c_str());
00527                 std::string ret(c_value);
00528                 bigvoice_FreeParameterValue(c_value);
00529                 return ret;
00530             }
00531
00532 #if BIGVOICE_HAS_CLOUDBUS
00541             void SetBus(const CBus& bus) {
00542                 assert(params);
00543                 bigvoice_SetInstanceBus(params,bus.Get());
00544             }
00545 #endif // BIGVOICE_HAS_CLOUDBUS
00546
00557         void SetPresetManager(CPresetLoader *preset_manager)
00558             {
00559                 if (!preset_manager) return;
00560                 assert(params);
00561                 bigvoice_SetPresetManager(params,
00562                     bigvoice_custom_reader,
00563                     bigvoice_custom_reader_free,
00564                     bigvoice_custom_writer,
00565                     bigvoice_custom_exists,
00566                     bigvoice_custom_getall,
00567                     bigvoice_custom_getall_free,
00568                     bigvoice_custom_remove,
00569                     bigvoice_custom_rename,
00570                     preset_manager->IsReadOnly(),
00571                     preset_manager
```

```
00572                 );
00573             }
00574
00597         bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
      Access_Key_ID, const std::string& Access_Key_Secret, const std::filesystem::path& save_path, int
      json_port = 0, unsigned int frames_per_chunk=12)
00598             {
00599                 assert(!instance);
00600                 std::string l_save_path;
00601                 if (!save_path.empty()) {
00602                     l_save_path = save_path.u8string();
00603                 }
00604                 instance = bigvoice_InitProcess3(LoginKey.c_str(), RadioName.c_str(),
      Access_Key_ID.c_str(), Access_Key_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
00605                 if (!instance) return false;
00606                 if (!init_metadata.empty()) {
00607                     for (auto&& [key,value]: init_metadata) {
00608                         bigvoice_SetMetadata(instance, key.c_str(), value.c_str());
00609                     }
00610                     init_metadata.clear();
00611                 }
00612                 update_thread = std::thread([this, json_port](){
00613                     OnUpdateThreadStart();
00614                     bigvoice_StartUpdateThread(instance, json_port);
00615                     OnUpdateThreadStop();
00616                 });
00617                 int timeout=5000;
00618                 #ifdef DEBUG
00619                     timeout=60*1000; // For debugger pauses
00620                 #endif
00621                 if (bigvoice_WaitUpdateThreadReady(instance, timeout)<0) {
00622                     return false;
00623                 }
00624                 return true;
00625             }
00626
00634         unsigned int GetChunkFrames() {
00635             if (instance) {
00636                 return bigvoice_GetProcessChunkFrames(instance);
00637             }
00638             return 0;
00639         }
00640
00648         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
00649             {
00650                 bigvoice_PresetManager_InformChange(instance, relative_path.c_str(),
      (bigvoice_PresetChange_Kind) change_kind);
00651             }
00652
00664         void SetMetadata(const std::string& key, const char* value)
00665             {
00666                 if (instance) {
00667                     bigvoice_SetMetadata(instance, key.c_str(), value);
00668                 } else {
00669                     init_metadata.push_back( {key, value} );
00670                 }
00671             }
00672
00682         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
00683             {
00684                 if (instance) {
00685                     typedef const char* pchar;
00686                     pchar* keyvalue=new pchar[2*list.size()+1];
00687                     size_t n=0;
00688                     for (auto&& [key,value]: list) {
00689                         keyvalue[2*n+0]=key.c_str();
00690                         keyvalue[2*n+1]=value;
00691                         n++;
00692                     }
00693                     keyvalue[2*n+0]=nullptr;
00694
00695                     bigvoice_SetMetadataMulti(instance, keyvalue);
00696
00697                     delete[] keyvalue;
00698                 } else {
00699                     for (auto&& [key,value]: list) {
00700                         if (value) {
00701                             init_metadata.push_back( {key, value} );
00702                         }
00703                     }
00704                 }
00705             }
00706
00712         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
00713             {
00714                 std::vector< std::tuple<std::string,std::string> > values;
```

```
00715                 const char** c_values = bigvoice_GetMetadataInfos(instance);
00716                 if (c_values) {
00717                     for (const char** c_value=c_values; *c_value; ) {
00718                         std::string key(*c_value);
00719                         c_value++;
00720                         if (*c_value) {
00721                             std::string descr(*c_value);
00722                             c_value++;
00723                             values.push_back( {key,descr} );
00724                         }
00725                     }
00726                     bigvoice_FreeMetadataInfos(instance, c_values);
00727                 }
00728                 return values;
00729             }
00730
00736         std::vector<AuxiliaryOutputInfos> GetAuxiliaryAudioOutputInfos()
00737             {
00738                 std::vector<AuxiliaryOutputInfos> values;
00739                 SOUND4_AuxiliaryOutputInfos** c_values =
      bigvoice_GetAuxiliaryAudioOutputInfos(instance);
00740                 if (c_values) {
00741                     for (SOUND4_AuxiliaryOutputInfos** c_value=c_values; *c_value; c_value++) {
00742                         auto c_info=*c_value;
00743                         values.emplace_back( AuxiliaryOutputInfos{
00744                             .id           = c_info->id           ,
00745                             .description  = c_info->description  ,
00746                             .sample_rate  = c_info->sample_rate  ,
00747                             .channel_count = c_info->channel_count,
00748                             .buffer       = c_info->buffer       ,
00749                         });
00750                     }
00751                     bigvoice_FreeAuxiliaryAudioInfos(c_values);
00752                 }
00753                 return values;
00754             }
00755         void ActivateAuxiliaryAudioID(const std::string& id)
00756             {
00757                 bigvoice_ActivateAuxiliaryAudioID(instance, id.c_str());
00758             }
00762         virtual void OnUpdateThreadStart() {}
00766         virtual void OnUpdateThreadStop() {}
00767
00768 #if BIGVOICE_HAS_WEBSERVER
00782         bool StartWebServer(int http_port, int https_port=0)
00783             {
00784                 assert(instance);
00785                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00786                 webserver = bigvoice_Webserver(http_port,https_port,instance);
00787                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00788                     return false;
00789                 }
00790                 int web_status=bigvoice_Webserver_Status(webserver);
00791                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00792                     return false;
00793                 }
00794                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00795                     return false;
00796                 }
00797                 return true;
00798             }
00799
00807             void StopWebServer(int timeout_ms = 1000)
00808             {
00809                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00810                     bigvoice_Webserver_Stop(webserver, timeout_ms);
00811                     webserver = SOUND4_INVALID_WEBSERVER_ID;
00812                 }
00813             }
00814
00818         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
00819             {
00820                 assert(instance);
00821                 bigvoice_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
00822             }
00826         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
00827             {
00828                 assert(instance);
00829                 char* c_contenttype=nullptr;
00830                 char* c_content=nullptr;
00831                 bigvoice_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
00832                 contenttype=c_contenttype;
00833                 content=c_content;
00834                 bigvoice_Webserver_FreeString(c_contenttype);
00835                 bigvoice_Webserver_FreeString(c_content);
```

```
00836         }
00837 #endif // BIGVOICE_HAS_WEBSERVER
00838
00847         int Terminate()
00848             {
00849                 if (!instance) return 0;
00850                 return bigvoice_TerminateProcess(instance);
00851             }
00852
00856         void Stop()
00857             {
00858                 if (!instance) return;
00859 #if BIGVOICE_HAS_WEBSERVER
00860                 StopWebServer();
00861 #endif // BIGVOICE_HAS_WEBSERVER
00862                 if (update_thread.joinable()) {
00863                     bigvoice_StopUpdateThread(instance); // this will exit the update thread
00864                     update_thread.join();
00865                 }
00866                 bigvoice_ExitProcess(instance);
00867                 instance=nullptr;
00868             }
00869
00879         unsigned int GetEstimatedDelay()
00880             {
00881                 assert(instance);
00882                 return bigvoice_GetEstimatedDelay(instance);
00883             }
00884
00897         std::array<float,InputSampleSize>& GetBufferIn()
00898             {
00899                 assert(instance);
00900                 float *buf=bigvoice_GetBufferIn(instance);
00901                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
00902
00903             }
00915         std::array<float,OutputSampleSize>& GetBufferOut()
00916             {
00917                 assert(instance);
00918                 float *buf= bigvoice_GetBufferOut(instance);
00919                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00920             }
00924         void ProcessAudio()
00925             {
00926                 assert(instance);
00927                 bigvoice_ProcessAudio(instance, bigvoice_GetBufferIn(instance),
    bigvoice_GetBufferOut(instance));
00928             }
00940         void ProcessAudio_Planar(float const * const *input, float * const *output)
00941             {
00942                 assert(instance);
00943                 bigvoice_ProcessAudio_Planar(instance, input, output);
00944             }
00945
00957             template<typename T>
00958         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00959             {
00960                 assert(instance);
00961                 std::vector<T> output;
00962                 unsigned int out_offset=0;
00963                 unsigned int in_offset=0;
00964                 unsigned int todo = input.size();
00965                 while (todo>0) {
00966                     unsigned int left = AddAudio(&input[in_offset], todo);
00967                     unsigned int out_avail = bigvoice_GetOutputCount(instance);
00968                     output.resize(out_offset + out_avail);
00969                     GetAudio(&output[out_offset], out_avail);
00970                     out_offset+=out_avail;
00971                     in_offset += todo-left;
00972                     todo=left;
00973                 }
00974                 return output;
00975             }
00976
00982         class CClient {
00983             public:
00984                 CClient(bigvoice_CInstance* instance)
00985                     {
00986                         assert(instance);
00987                         client=bigvoice_NewClient(instance);
00988                         if (!client) throw std::bad_alloc();
00989                     }
00990                 // non-copyable
00991                 CClient( const CInstance& ) = delete;
00992                 CClient& operator=( const CInstance& ) = delete;
00993                 ~CClient()
00994                     {
```

```
00995                         if (client) {
00996                             bigvoice_DeleteClient(client);
00997                             client=nullptr;
00998                         }
00999                     }
01007                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01008                     {
01009                         assert(client);
01010                         int need_save=0;
01011                         const char *canswer = bigvoice_ProcessJson(client, request.c_str(),
      &need_save);
01012                         if (!canswer) return {};
01013                         std::string answer(canswer);
01014                         bigvoice_FreeJsonAnswer (canswer);
01015                         if (NeedSave) {
01016                             *NeedSave=(need_save!=0);
01017                         }
01018                         return answer;
01019                     }
01020             private:
01021                 bigvoice_CClientInstance *client = nullptr;
01022         };
01030         std::shared_ptr<CClient> NewClient()
01031             {
01032                 assert(instance);
01033                 return std::make_shared<CClient>(instance);
01034             }
01035
01046         bool SaveState() {
01047             assert(instance);
01048             if (bigvoice_SaveState(instance)!=0) {
01049                 return false;
01050             }
01051             return true;
01052         }
01053 #ifdef _WIN32
01060         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01061             bigvoice_SetInstanceTracing(params, tracing_provider, activity_guid);
01062         }
01069         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01070             assert(instance);
01071             bigvoice_SetInstanceTracingProcessActivity(instance, activity_guid);
01072         }
01073 #endif // _WIN32
01074     protected:
01075             template<typename T>
01076         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01077             {
01078                 assert(instance);
01079                 return bigvoice_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01080             }
01081             template<typename T>
01082         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01083             {
01084                 assert(instance);
01085                 return bigvoice_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01086             }
01087
01088     private:
01089         bigvoice_CParameters* params = nullptr;
01090         bigvoice_CInstance* instance = nullptr;
01091         std::thread update_thread;
01092         std::vector< std::pair<std::string,std::string> > init_metadata;
01093 #if BIGVOICE_HAS_WEBSERVER
01094         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01095 #endif // BIGVOICE_HAS_WEBSERVER
01096     };
01097
01098 }; // namespace bigvoice
01099 }; // namespace sound4
01100
01101 // bridge C callbacks to CPresetLoader
01102 extern "C" {
01103     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
01104         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01105         std::string content=preset_loader->Read(filename);
01106         return strdup(content.c_str());
01107     }
01108     static void bigvoice_custom_reader_free(char *content, void* userdata) {
01109         if (!content) return;
01110         free(content);
01111     }
01112     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01113         sound4::bigvoice::CPresetLoader
```

```
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01114         auto res=preset_loader->Write(filename,content);
01115         return res?0:-1;
01116     }
01117     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
01118         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01119         auto res=preset_loader->Exists(filename);
01120         return res?0:-1;
01121     }
01122     static fs_char** bigvoice_custom_getall(void* userdata) {
01123         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01124         auto res=preset_loader->GetAll();
01125         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01126         for (size_t n=0;n<res.size();n++) {
01127             all[n]=fs_strdup(res[n].c_str());
01128         }
01129         all[res.size()]=nullptr;
01130         return all;
01131     }
01132     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
01133         if (!all) return;
01134         for (fs_char** one=all;*one;one++) {
01135             free(*one);
01136         }
01137         free(all);
01138     }
01139     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
01140         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01141         auto res=preset_loader->Remove(filename);
01142         return res?0:-1;
01143     }
01144     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01145         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01146         auto res=preset_loader->Rename(from,to);
01147         return res?0:-1;
01148     }
01149
01150 };
```

## 15.27 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 <libloaderapi.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.*

- class sound4::bigvoice::dyn::CBus

  *Shared Bus.*

- class sound4::bigvoice::dyn::CPresetLoader

  *Custom preset handler helper.*

- class sound4::bigvoice::dyn::CInstance

  *Instance handling class.*

- class sound4::bigvoice::dyn::CInstance::CClient

### Namespaces

- namespace sound4
- namespace sound4::bigvoice
- namespace sound4::bigvoice::dyn
- namespace sound4::bigvoice::dyn::helper

### Macros

- #define UNIXLIKE 0

### Typedefs

- 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

### Functions

- static char * bigvoice_custom_reader (const fs_char *filename, void *userdata)
- static void bigvoice_custom_reader_free (char *content, void *userdata)
- static int bigvoice_custom_writer (const fs_char *filename, const char *content, void *userdata)
- static int bigvoice_custom_exists (const fs_char *filename, void *userdata)
- static fs_char ** bigvoice_custom_getall (void *userdata)
- static void bigvoice_custom_getall_free (fs_char **all, void *userdata)
- static int bigvoice_custom_remove (const fs_char *filename, void *userdata)
- static int bigvoice_custom_rename (const fs_char *from, const fs_char *to, void *userdata)
- static std::string sound4::bigvoice::dyn::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::bigvoice::dyn::helper::UTF8ToWString (const std::string &str)
- static void sound4::bigvoice::dyn::helper::AudioConvertFrom (CDynLib &dynlib, const uint8_t *payload, float *output, size_t nSpl, bigvoice_SampleFormat fmt)
- static void sound4::bigvoice::dyn::helper::AudioConvertTo (CDynLib &dynlib, const float *input, uint8_t *payload, size_t nSpl, bigvoice_SampleFormat fmt)
- static void sound4::bigvoice::dyn::helper::StereoToMono (CDynLib &dynlib, const float *input, float *output, size_t nFrame)
- static void sound4::bigvoice::dyn::helper::MonoToStereo (CDynLib &dynlib, const float *input, float *output, size_t nFrame)

- static void sound4::bigvoice::dyn::helper::StereoToMono_Planar (CDynLib &dynlib, const float *inputL, const float *inputR, float *output, size_t nFrame)
- static void sound4::bigvoice::dyn::helper::MonoToStereo_Planar (CDynLib &dynlib, const float *input, float *outputL, float *outputR, size_t nFrame)
- static void sound4::bigvoice::dyn::helper::AudioMonoFromLiveStereo (CDynLib &dynlib, const uint8_t *payload, float *output)
- static void sound4::bigvoice::dyn::helper::AudioMonoToLiveStereo (CDynLib &dynlib, const float *input, uint8_t *payload)
- static std::string sound4::bigvoice::dyn::GetVersion (CDynLib &dynlib)
- static unsigned int sound4::bigvoice::dyn::GetChunkSizeInFrames (CDynLib &dynlib)
- static std::vector< unsigned int > sound4::bigvoice::dyn::GetPossibleChunkSizeInFrames (CDynLib &dynlib)
- static unsigned int sound4::bigvoice::dyn::GetChannelCount (CDynLib &dynlib)
- static unsigned int sound4::bigvoice::dyn::GetAudioInputCount (CDynLib &dynlib)
- static unsigned int sound4::bigvoice::dyn::GetAudioOutputCount (CDynLib &dynlib)
- static unsigned int sound4::bigvoice::dyn::GetSampleRate (CDynLib &dynlib)
- static void sound4::bigvoice::dyn::SanityCheck (CDynLib &dynlib, bool a_bCheckFrames=true)
- static std::string sound4::bigvoice::dyn::GetFormatName (CDynLib &dynlib, const SampleFormat fmt)
- static SampleFormat sound4::bigvoice::dyn::GetFormatFromName (CDynLib &dynlib, const std::string &name)
- static unsigned int sound4::bigvoice::dyn::GetBytesFromFormat (CDynLib &dynlib, const SampleFormat fmt)
- static void sound4::bigvoice::dyn::_log_cb_c (bigvoice_LogSeverity severity, const char *c_msg)
- static void sound4::bigvoice::dyn::SetLogSeverity (CDynLib &dynlib, LogSeverity severity)
- static void sound4::bigvoice::dyn::SetLoggerCallback (CDynLib &dynlib, log_cb_t cb)

**Variables**

- static constexpr const char * sound4::bigvoice::dyn::process_name = "SOUND4 BIGVOICE.CL"
- static constexpr const char * sound4::bigvoice::dyn::process_shortname = "bigvoice"
- const size_t sound4::bigvoice::dyn::ChunkFrames = 12
- const size_t sound4::bigvoice::dyn::InputChannels = 1 * 1
- const size_t sound4::bigvoice::dyn::OutputChannels = 1 * 1
- 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 = 48000
- static log_cb_t sound4::bigvoice::dyn::_log_cb

## 15.27.1 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.

## 15.27.2 Macro Definition Documentation

### 15.27.2.1 UNIXLIKE

```
#define UNIXLIKE 0
```

Definition at line 23 of file sound4.bigvoice.cl_dyn.hpp.

## 15.27.3 Function Documentation

### 15.27.3.1 bigvoice_custom_exists()

```
static int bigvoice_custom_exists (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1587 of file sound4.bigvoice.cl_dyn.hpp.

### 15.27.3.2 bigvoice_custom_getall()

```
static fs_char ** bigvoice_custom_getall (
            void * userdata )  [static]
```

Definition at line 1592 of file sound4.bigvoice.cl_dyn.hpp.

### 15.27.3.3 bigvoice_custom_getall_free()

```
static void bigvoice_custom_getall_free (
            fs_char ** all,
            void * userdata )  [static]
```

Definition at line 1602 of file sound4.bigvoice.cl_dyn.hpp.

### 15.27.3.4 bigvoice_custom_reader()

```
static char * bigvoice_custom_reader (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1573 of file sound4.bigvoice.cl_dyn.hpp.

### 15.27.3.5 bigvoice_custom_reader_free()

```
static void bigvoice_custom_reader_free (
            char * content,
            void * userdata )  [static]
```

Definition at line 1578 of file sound4.bigvoice.cl_dyn.hpp.

### 15.27.3.6 bigvoice_custom_remove()

```
static int bigvoice_custom_remove (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1609 of file sound4.bigvoice.cl_dyn.hpp.

#### 15.27.3.7 bigvoice_custom_rename()

```
static int bigvoice_custom_rename (
            const fs_char * from,
            const fs_char * to,
            void * userdata )  [static]
```

Definition at line 1614 of file sound4.bigvoice.cl_dyn.hpp.

#### 15.27.3.8 bigvoice_custom_writer()

```
static int bigvoice_custom_writer (
            const fs_char * filename,
            const char * content,
            void * userdata )  [static]
```

Definition at line 1582 of file sound4.bigvoice.cl_dyn.hpp.

## 15.28 sound4.bigvoice.cl_dyn.hpp

Go to the documentation of this file.

```
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 #if defined(__unix__) || defined(__APPLE__)
00021     #define UNIXLIKE 1
00022 #else
00023     #define UNIXLIKE 0
00024 #endif
00025
00026 #if UNIXLIKE
00027     #include <dlfcn.h>
00028 #elif defined(_WIN32)
00029     #include <libloaderapi.h>
00030 #else
00031     #error "Unsupported OS"
00032 #endif
00033
00034 // Needed to have function declarations
00035 #include "sound4.bigvoice.cl.h"
00036
00037
00038 #ifdef _WIN32
00039     #include <stringapiset.h>
00040     #include <wchar.h>
00042     #define fs_strdup wcsdup
00043     #ifdef _MSC_VER
00044         #pragma warning(disable : 4996)
00045     #endif
00046 #else // !_WIN32
00048     #define fs_strdup strdup
00049 #endif // !_WIN32
00050
00051 // bridge C callbacks to CPresetLoader
00052 extern "C" {
00053     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata);
00054     static void bigvoice_custom_reader_free(char *content, void* userdata);
00055     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata);
00056     static int bigvoice_custom_exists(const fs_char *filename, void* userdata);
```

```
00057     static fs_char** bigvoice_custom_getall(void* userdata);
00058     static void bigvoice_custom_getall_free(fs_char** all, void* userdata);
00059     static int bigvoice_custom_remove(const fs_char *filename, void* userdata);
00060     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00061 };
00062
00066 namespace sound4 {
00070 namespace bigvoice {
00074 namespace dyn {
00075
00079     namespace helper {
00080
00086             template <typename T>
00087         class DynFuncHolder {
00088             // Storage definition, depending on OS
00089             #if UNIXLIKE
00090                 using DynFunc_t = void *;
00091             #else
00092                 using DynFunc_t = FARPROC;
00093             #endif
00094         private:
00095             DynFunc_t m_ptr = NULL;
00096         public:
00097             DynFuncHolder() = default;
00098             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00099             operator bool() const { return (m_ptr!=NULL); }
00100             bool IsOk() const { return m_ptr!=NULL; }
00101             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00102         };
00103
00108         class CDynLoader {
00109             // Storage definition, depending on OS
00110             #if UNIXLIKE
00111                 using DynLib_t = void *;
00112             #else
00113                 using DynLib_t = HMODULE;
00114             #endif
00115             private:
00116                 DynLib_t m_lib {};
00117             public:
00118                 CDynLoader() = default;
00119                 CDynLoader(CDynLoader&&) = default;
00120                 // No copy allowed
00121                 CDynLoader(const CDynLoader&) = delete;
00122                 CDynLoader& operator=(CDynLoader const&) = delete;
00123                 operator bool() const {
00124                     return (m_lib!=NULL);
00125                 }
00126                 bool IsOk() const {
00127                     return (m_lib!=NULL);
00128                 }
00129                 #if UNIXLIKE
00130                     void Close() {
00131                         if (IsOk()) {
00132                             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00133                             dlclose(m_lib);
00134                             #endif // __SANITIZE_ADDRESS__
00135                             m_lib=NULL;
00136                         }
00137                     }
00138                     bool Load(const std::filesystem::path& dynlib)
00139                     {
00140                         Close();
00141                         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00142                         return IsOk();
00143                     }
00144                         template <typename T>
00145                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00146                         auto ptr=dlsym(m_lib, name.c_str());
00147                         if (!ptr) {
00148                             throw std::runtime_error("Missing function in library");
00149                         }
00150                         return DynFuncHolder<T>(ptr);
00151                     }
00152                         template <typename T>
00153                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00154                         auto ptr=dlsym(m_lib, name.c_str());
00155                         return DynFuncHolder<T>(ptr);
00156                     }
00157                     static std::filesystem::path GetThisLibraryPath(void) {
00158                         static Dl_info info;
00159                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
00160                         auto rp=realpath(info.dli_fname, NULL);
00161                         if (!rp) {
00162                             #if defined(__unix__)
00163                                 rp=realpath("/proc/self/exe", NULL);
00164                             #elif defined(__APPLE__)
```

```
00165                                 // How to solve this for Apple ?
00166                             #endif
00167                         }
00168                         std::filesystem::path p(rp);
00169                         free(rp);
00170                         return p;
00171                     }
00172                 #else
00173                     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_S
00174                     {
00175                         Close();
00176                         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00177                         return IsOk();
00178                     }
00179                     void Close() {
00180                         if (IsOk()) {
00181                             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
      FreeLibraryAndExitThread but...
00182                             // So define SOUND4_CALL_FREELIBRARY before including this file if you
      really want to unload the library dynamically
00183                             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00184                                 FreeLibraryAndExitThread(m_lib, 0);
00185                             #elif defined(SOUND4_CALL_FREELIBRARY)
00186                                 FreeLibrary(m_lib);
00187                             #endif // SOUND4_CALL_FREELIBRARY
00188                             m_lib=NULL;
00189                         }
00190                     }
00191                         template <typename T>
00192                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00193                         auto ptr=GetProcAddress(m_lib, name.c_str());
00194                         if (!ptr) {
00195                             throw std::runtime_error("Missing function in library");
00196                         }
00197                         return DynFuncHolder<T>(ptr);
00198                     }
00199                         template <typename T>
00200                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00201                         auto ptr=GetProcAddress(m_lib, name.c_str());
00202                         return DynFuncHolder<T>(ptr);
00203                     }
00204                     static std::filesystem::path GetThisLibraryPath(void) {
00205                         static wchar_t path[MAX_PATH]={};
00206                         HMODULE hm = NULL;
00207                         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00208                                 GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00209                                 (LPCWSTR) &GetThisLibraryPath, &hm))
00210                         {
00211                             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00212                                 return std::filesystem::path(path);
00213                             }
00214                         }
00215                         return {};
00216                     }
00217                 #endif
00218                 virtual ~CDynLoader() {
00219                     Close();
00220                 }
00221         }; // class CDynLoader
00222     }; // namespace helper
00223
00230     class CDynLib {
00231     private:
00232         helper::CDynLoader m_lib;
00233         bool m_bOK = false;
00234     public:
00235         CDynLib() = default;
00236
00237         // All those wrapped functions have the same signature as there C bigvoice_XXX
00238         helper::DynFuncHolder< decltype(bigvoice_GetVersion             ) > GetVersion             ;
00239         helper::DynFuncHolder< decltype(bigvoice_GetChunkSizeInFrames   ) > GetChunkSizeInFrames   ;
00240         helper::DynFuncHolder< decltype(bigvoice_GetChannelCount        ) > GetChannelCount        ;
00241         helper::DynFuncHolder< decltype(bigvoice_GetAudioInputCount     ) > GetAudioInputCount     ;
00242         helper::DynFuncHolder< decltype(bigvoice_GetAudioOutputCount    ) > GetAudioOutputCount    ;
00243         helper::DynFuncHolder< decltype(bigvoice_GetSampleRate          ) > GetSampleRate          ;
00244         helper::DynFuncHolder< decltype(bigvoice_SetLoggerCallback      ) > SetLoggerCallback      ;
00245         helper::DynFuncHolder< decltype(bigvoice_SetLogSeverity         ) > SetLogSeverity         ;
00246         helper::DynFuncHolder< decltype(bigvoice_NewParameters          ) > NewParameters          ;
00247         helper::DynFuncHolder< decltype(bigvoice_FreeParameters         ) > FreeParameters         ;
00248         helper::DynFuncHolder< decltype(bigvoice_SetParameter           ) > SetParameter           ;
00249         helper::DynFuncHolder< decltype(bigvoice_GetParameter           ) > GetParameter           ;
00250         helper::DynFuncHolder< decltype(bigvoice_FreeParameterValue     ) > FreeParameterValue     ;
00251         helper::DynFuncHolder< decltype(bigvoice_InitProcess            ) > InitProcess            ;
00252         helper::DynFuncHolder< decltype(bigvoice_InitProcess2           ) > InitProcess2           ;
00253         helper::DynFuncHolder< decltype(bigvoice_TerminateProcess       ) > TerminateProcess       ;
00254         helper::DynFuncHolder< decltype(bigvoice_ExitProcess            ) > ExitProcess            ;
```

```
00255         helper::DynFuncHolder< decltype(bigvoice_StartUpdateThread      ) > StartUpdateThread      ;
00256         helper::DynFuncHolder< decltype(bigvoice_StopUpdateThread       ) > StopUpdateThread       ;
00257         helper::DynFuncHolder< decltype(bigvoice_WaitUpdateThreadReady  ) > WaitUpdateThreadReady  ;
00258         helper::DynFuncHolder< decltype(bigvoice_ProcessAudio           ) > ProcessAudio           ;
00259         helper::DynFuncHolder< decltype(bigvoice_ProcessAudio_Planar    ) > ProcessAudio_Planar    ;
00260         helper::DynFuncHolder< decltype(bigvoice_GetBufferIn            ) > GetBufferIn            ;
00261         helper::DynFuncHolder< decltype(bigvoice_GetBufferOut           ) > GetBufferOut           ;
00262         helper::DynFuncHolder< decltype(bigvoice_GetEstimatedDelay      ) > GetEstimatedDelay      ;
00263         helper::DynFuncHolder< decltype(bigvoice_GetFormatName          ) > GetFormatName          ;
00264         helper::DynFuncHolder< decltype(bigvoice_GetFormatFromName      ) > GetFormatFromName      ;
00265         helper::DynFuncHolder< decltype(bigvoice_GetBytesFromFormat     ) > GetBytesFromFormat     ;
00266         helper::DynFuncHolder< decltype(bigvoice_GetMaxPacketFrame      ) > GetMaxPacketFrame      ;
00267         helper::DynFuncHolder< decltype(bigvoice_AddAudio               ) > AddAudio               ;
00268         helper::DynFuncHolder< decltype(bigvoice_AddPadAudio            ) > AddPadAudio            ;
00269         helper::DynFuncHolder< decltype(bigvoice_GetOutputCount         ) > GetOutputCount         ;
00270         helper::DynFuncHolder< decltype(bigvoice_GetAudio               ) > GetAudio               ;
00271         helper::DynFuncHolder< decltype(bigvoice_AudioConvertFrom       ) > AudioConvertFrom       ;
00272         helper::DynFuncHolder< decltype(bigvoice_AudioConvertTo         ) > AudioConvertTo         ;
00273         helper::DynFuncHolder< decltype(bigvoice_StereoToMono           ) > StereoToMono           ;
00274         helper::DynFuncHolder< decltype(bigvoice_MonoToStereo           ) > MonoToStereo           ;
00275         helper::DynFuncHolder< decltype(bigvoice_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo;
00276         helper::DynFuncHolder< decltype(bigvoice_AudioMonoToLiveStereo  ) > AudioMonoToLiveStereo  ;
00277         helper::DynFuncHolder< decltype(bigvoice_NewClient              ) > NewClient              ;
00278         helper::DynFuncHolder< decltype(bigvoice_DeleteClient           ) > DeleteClient           ;
00279         helper::DynFuncHolder< decltype(bigvoice_ProcessJson            ) > ProcessJson            ;
00280         helper::DynFuncHolder< decltype(bigvoice_FreeJsonAnswer         ) > FreeJsonAnswer         ;
00281         helper::DynFuncHolder< decltype(bigvoice_SaveState              ) > SaveState              ;
00282 #if BIGVOICE_HAS_WEBSERVER
00283         helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp          ) > Webserver_tcp          ;
00284         helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp2         ) > Webserver_tcp2         ;
00285         helper::DynFuncHolder< decltype(bigvoice_Webserver              ) > Webserver              ;
00286         helper::DynFuncHolder< decltype(bigvoice_Webserver_Stop         ) > Webserver_Stop         ;
00287         helper::DynFuncHolder< decltype(bigvoice_Webserver_Status       ) > Webserver_Status       ;
00288 #endif // BIGVOICE_HAS_WEBSERVER
00289
00290         // added 2023-03-22
00291         helper::DynFuncHolder< decltype(bigvoice_StereoToMono_Planar    ) > StereoToMono_Planar    ;
00292         helper::DynFuncHolder< decltype(bigvoice_MonoToStereo_Planar    ) > MonoToStereo_Planar    ;
00293
00294         // added 2023-05-22
00295         helper::DynFuncHolder< decltype(bigvoice_SetMetadata            ) > SetMetadata            ;
00296         helper::DynFuncHolder< decltype(bigvoice_GetMetadataInfos       ) > GetMetadataInfos       ;
00297         helper::DynFuncHolder< decltype(bigvoice_FreeMetadataInfos      ) > FreeMetadataInfos      ;
00298
00299         // added 2023-06-20
00300         helper::DynFuncHolder< decltype(bigvoice_SetPresetManager               ) > SetPresetManager
    ;
00301         helper::DynFuncHolder< decltype(bigvoice_PresetManager_InformChange     ) >
    PresetManager_InformChange     ;
00302
00303         // added 2023-09-21
00304         helper::DynFuncHolder< decltype(bigvoice_GetPossibleChunkSizeInFrames   ) >
    GetPossibleChunkSizeInFrames   ;
00305         helper::DynFuncHolder< decltype(bigvoice_GetProcessChunkFrames          ) >
    GetProcessChunkFrames   ;
00306         helper::DynFuncHolder< decltype(bigvoice_InitProcess3                   ) > InitProcess3
    ;
00307
00308 #if BIGVOICE_HAS_CLOUDBUS
00309         // added 2023-06-26
00310         helper::DynFuncHolder< decltype(bigvoice_NewBus                 ) > NewBus      ;
00311         helper::DynFuncHolder< decltype(bigvoice_FreeBus                ) > FreeBus      ;
00312         helper::DynFuncHolder< decltype(bigvoice_SetInstanceBus         ) > SetInstanceBus      ;
00313 #endif // BIGVOICE_HAS_CLOUDBUS
00314
00315         // added 2024-02-19
00316         helper::DynFuncHolder< decltype(bigvoice_SetMetadataMulti       ) > SetMetadataMulti        ;
00317
00318 #if BIGVOICE_HAS_WEBSERVER
00319         // added 2024-05-23
00320         helper::DynFuncHolder< decltype(bigvoice_Webserver_SetAppHealth    ) > Webserver_SetAppHealth
    ;
00321         helper::DynFuncHolder< decltype(bigvoice_Webserver_GetAppHealth    ) > Webserver_GetAppHealth
    ;
00322         helper::DynFuncHolder< decltype(bigvoice_Webserver_FreeString      ) > Webserver_FreeString
    ;
00323 #endif // BIGVOICE_HAS_WEBSERVER
00324
00325 #ifdef _WIN32
00326         // added 2025-05-28
00327         helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracing               ) >
    SetInstanceTracing               ;
00328         helper::DynFuncHolder< decltype(bigvoice_SetInstanceTracingProcessActivity ) >
    SetInstanceTracingProcessActivity ;
00329 #endif // _WIN32
00330
00338         bool Load(const std::filesystem::path& path = {}) {
```

```
00339             std::string filename;
00340             #if defined(__APPLE__)
00341                 filename="libsound4.bigvoice.cl.dylib";
00342             #elif defined(__unix__)
00343                 filename="libsound4.bigvoice.cl.so";
00344             #else
00345                 filename="sound4.bigvoice.cl.dll";
00346             #endif
00347             m_lib.Close();
00348             // If a path is given, use it directly and do not try other path
00349             if (!path.empty() && !m_lib.Load(path / filename)) {
00350                 return false;
00351             } else if (!m_lib.IsOk()) {
00352                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00353                 #ifndef _WIN32
00354                 // Linux: if in a bin directory, try ../lib first
00355                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00356                     auto libdir = thisdir.parent_path() / "lib";
00357                     m_lib.Load(libdir / filename);
00358                 }
00359                 #endif // !_WIN32
00360                 if (!m_lib.IsOk()) {
00361                     // Search in the same directory this code is in
00362                     m_lib.Load(thisdir / filename);
00363                 }
00364                 if (!m_lib.IsOk()) {
00365                     // Try current path
00366                     std::error_code ec;
00367                     auto p = std::filesystem::current_path(ec);
00368                     if (!ec) {
00369                         m_lib.Load(p / filename);
00370                     }
00371                 }
00372                 if (!m_lib.IsOk()) {
00373                     return false;
00374                 }
00375             }
00376             // Load all C functions
00377             try {
00378                 GetVersion              = m_lib.GetSymbol< decltype(bigvoice_GetVersion              )
    >("bigvoice_GetVersion"             );
00379                 GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(bigvoice_GetChunkSizeInFrames    )
    >("bigvoice_GetChunkSizeInFrames"   );
00380                 GetChannelCount         = m_lib.GetSymbol< decltype(bigvoice_GetChannelCount         )
    >("bigvoice_GetChannelCount"        );
00381                 GetAudioInputCount      = m_lib.GetSymbol< decltype(bigvoice_GetAudioInputCount      )
    >("bigvoice_GetAudioInputCount"     );
00382                 GetAudioOutputCount     = m_lib.GetSymbol< decltype(bigvoice_GetAudioOutputCount     )
    >("bigvoice_GetAudioOutputCount"    );
00383                 GetSampleRate           = m_lib.GetSymbol< decltype(bigvoice_GetSampleRate           )
    >("bigvoice_GetSampleRate"          );
00384                 SetLoggerCallback       = m_lib.GetSymbol< decltype(bigvoice_SetLoggerCallback       )
    >("bigvoice_SetLoggerCallback"      );
00385                 SetLogSeverity          = m_lib.GetSymbol< decltype(bigvoice_SetLogSeverity          )
    >("bigvoice_SetLogSeverity"         );
00386                 NewParameters           = m_lib.GetSymbol< decltype(bigvoice_NewParameters           )
    >("bigvoice_NewParameters"          );
00387                 FreeParameters          = m_lib.GetSymbol< decltype(bigvoice_FreeParameters          )
    >("bigvoice_FreeParameters"         );
00388                 SetParameter            = m_lib.GetSymbol< decltype(bigvoice_SetParameter            )
    >("bigvoice_SetParameter"           );
00389                 GetParameter            = m_lib.GetSymbol< decltype(bigvoice_GetParameter            )
    >("bigvoice_GetParameter"           );
00390                 FreeParameterValue      = m_lib.GetSymbol< decltype(bigvoice_FreeParameterValue      )
    >("bigvoice_FreeParameterValue"     );
00391                 InitProcess             = m_lib.GetSymbol< decltype(bigvoice_InitProcess             )
    >("bigvoice_InitProcess"            );
00392                 InitProcess2            = m_lib.GetSymbol< decltype(bigvoice_InitProcess2            )
    >("bigvoice_InitProcess2"           );
00393                 TerminateProcess        = m_lib.GetSymbol< decltype(bigvoice_TerminateProcess        )
    >("bigvoice_TerminateProcess"       );
00394                 ExitProcess             = m_lib.GetSymbol< decltype(bigvoice_ExitProcess             )
    >("bigvoice_ExitProcess"            );
00395                 StartUpdateThread       = m_lib.GetSymbol< decltype(bigvoice_StartUpdateThread       )
    >("bigvoice_StartUpdateThread"      );
00396                 StopUpdateThread        = m_lib.GetSymbol< decltype(bigvoice_StopUpdateThread        )
    >("bigvoice_StopUpdateThread"       );
00397                 WaitUpdateThreadReady   = m_lib.GetSymbol< decltype(bigvoice_WaitUpdateThreadReady   )
    >("bigvoice_WaitUpdateThreadReady"  );
00398                 ProcessAudio            = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio            )
    >("bigvoice_ProcessAudio"           );
00399                 ProcessAudio_Planar     = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio_Planar     )
    >("bigvoice_ProcessAudio_Planar"    );
00400                 GetBufferIn             = m_lib.GetSymbol< decltype(bigvoice_GetBufferIn             )
    >("bigvoice_GetBufferIn"            );
00401                 GetBufferOut            = m_lib.GetSymbol< decltype(bigvoice_GetBufferOut            )
    >("bigvoice_GetBufferOut"           );
```

```
00402                 GetEstimatedDelay       = m_lib.GetSymbol< decltype(bigvoice_GetEstimatedDelay      )
      >("bigvoice_GetEstimatedDelay"      );
00403                 GetFormatName           = m_lib.GetSymbol< decltype(bigvoice_GetFormatName          )
      >("bigvoice_GetFormatName"          );
00404                 GetFormatFromName       = m_lib.GetSymbol< decltype(bigvoice_GetFormatFromName      )
      >("bigvoice_GetFormatFromName"      );
00405                 GetBytesFromFormat      = m_lib.GetSymbol< decltype(bigvoice_GetBytesFromFormat     )
      >("bigvoice_GetBytesFromFormat"     );
00406                 GetMaxPacketFrame       = m_lib.GetSymbol< decltype(bigvoice_GetMaxPacketFrame      )
      >("bigvoice_GetMaxPacketFrame"      );
00407                 AddAudio                = m_lib.GetSymbol< decltype(bigvoice_AddAudio               )
      >("bigvoice_AddAudio"               );
00408                 AddPadAudio             = m_lib.GetSymbol< decltype(bigvoice_AddPadAudio            )
      >("bigvoice_AddPadAudio"            );
00409                 GetOutputCount          = m_lib.GetSymbol< decltype(bigvoice_GetOutputCount         )
      >("bigvoice_GetOutputCount"         );
00410                 GetAudio                = m_lib.GetSymbol< decltype(bigvoice_GetAudio               )
      >("bigvoice_GetAudio"               );
00411                 AudioConvertFrom        = m_lib.GetSymbol< decltype(bigvoice_AudioConvertFrom       )
      >("bigvoice_AudioConvertFrom"       );
00412                 AudioConvertTo          = m_lib.GetSymbol< decltype(bigvoice_AudioConvertTo         )
      >("bigvoice_AudioConvertTo"         );
00413                 StereoToMono            = m_lib.GetSymbol< decltype(bigvoice_StereoToMono           )
      >("bigvoice_StereoToMono"           );
00414                 MonoToStereo            = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo           )
      >("bigvoice_MonoToStereo"           );
00415                 AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(bigvoice_AudioMonoFromLiveStereo)
      >("bigvoice_AudioMonoFromLiveStereo");
00416                 AudioMonoToLiveStereo   = m_lib.GetSymbol< decltype(bigvoice_AudioMonoToLiveStereo  )
      >("bigvoice_AudioMonoToLiveStereo"  );
00417                 NewClient               = m_lib.GetSymbol< decltype(bigvoice_NewClient              )
      >("bigvoice_NewClient"              );
00418                 DeleteClient            = m_lib.GetSymbol< decltype(bigvoice_DeleteClient           )
      >("bigvoice_DeleteClient"           );
00419                 ProcessJson             = m_lib.GetSymbol< decltype(bigvoice_ProcessJson            )
      >("bigvoice_ProcessJson"            );
00420                 FreeJsonAnswer          = m_lib.GetSymbol< decltype(bigvoice_FreeJsonAnswer         )
      >("bigvoice_FreeJsonAnswer"         );
00421                 SaveState               = m_lib.GetSymbol< decltype(bigvoice_SaveState              )
      >("bigvoice_SaveState"              );
00422 #if BIGVOICE_HAS_WEBSERVER
00423                 Webserver_tcp           = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp          )
      >("bigvoice_Webserver_tcp"          );
00424                 Webserver_tcp2          = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp2         )
      >("bigvoice_Webserver_tcp2"         );
00425                 Webserver               = m_lib.GetSymbol< decltype(bigvoice_Webserver              )
      >("bigvoice_Webserver"              );
00426                 Webserver_Stop          = m_lib.GetSymbol< decltype(bigvoice_Webserver_Stop         )
      >("bigvoice_Webserver_Stop"         );
00427                 Webserver_Status        = m_lib.GetSymbol< decltype(bigvoice_Webserver_Status       )
      >("bigvoice_Webserver_Status"       );
00428 #endif // BIGVOICE_HAS_WEBSERVER
00429
00430                 // added 2023-03-22
00431                 StereoToMono_Planar     = m_lib.GetSymbol< decltype(bigvoice_StereoToMono_Planar    )
      >("bigvoice_StereoToMono_Planar"    );
00432                 MonoToStereo_Planar     = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo_Planar    )
      >("bigvoice_MonoToStereo_Planar"    );
00433
00434             } catch (std::runtime_error& ) {
00435                 return false;
00436             }
00437             // C functions allowed to be missing
00438             try {
00439                 // added 2023-05-22
00440                 SetMetadata             = m_lib.GetSymbol< decltype(bigvoice_SetMetadata            )
      >("bigvoice_SetMetadata"            );
00441                 GetMetadataInfos        = m_lib.GetSymbol< decltype(bigvoice_GetMetadataInfos       )
      >("bigvoice_GetMetadataInfos"       );
00442                 FreeMetadataInfos       = m_lib.GetSymbol< decltype(bigvoice_FreeMetadataInfos      )
      >("bigvoice_FreeMetadataInfos"      );
00443                 // added 2023-06-20
00444                 SetPresetManager            = m_lib.GetSymbol< decltype(bigvoice_SetPresetManager
      ) >("bigvoice_SetPresetManager"             );
00445                 PresetManager_InformChange  = m_lib.GetSymbol<
      decltype(bigvoice_PresetManager_InformChange     ) >("bigvoice_PresetManager_InformChange"    );
00446
00447                 // added 2023-09-21
00448                 GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
      decltype(bigvoice_GetPossibleChunkSizeInFrames  ) >("bigvoice_GetPossibleChunkSizeInFrames"  );
00449                 GetProcessChunkFrames        = m_lib.GetSymbol<
      decltype(bigvoice_GetProcessChunkFrames         ) >("bigvoice_GetProcessChunkFrames"         );
00450                 InitProcess3                 = m_lib.GetSymbol< decltype(bigvoice_InitProcess3
      ) >("bigvoice_InitProcess3"                 );
00451
00452 #if BIGVOICE_HAS_CLOUDBUS
00453                 // added 2023-06-26
```

```
00454                 NewBus                  = m_lib.GetSymbol< decltype(bigvoice_NewBus                  )
      >("bigvoice_NewBus"                );
00455                 FreeBus                 = m_lib.GetSymbol< decltype(bigvoice_FreeBus                 )
      >("bigvoice_FreeBus"               );
00456                 SetInstanceBus          = m_lib.GetSymbol< decltype(bigvoice_SetInstanceBus          )
      >("bigvoice_SetInstanceBus"        );
00457 #endif // BIGVOICE_HAS_CLOUDBUS
00458
00459                 // added 2024-02-19
00460                 SetMetadataMulti        = m_lib.GetSymbol< decltype(bigvoice_SetMetadataMulti        )
      >("bigvoice_SetMetadataMulti"      );
00461
00462 #if BIGVOICE_HAS_WEBSERVER
00463                 // added 2024-05-23
00464                 Webserver_SetAppHealth      = m_lib.GetSymbol<
      decltype(bigvoice_Webserver_SetAppHealth      ) >("Webserver_SetAppHealth"     );
00465                 Webserver_GetAppHealth      = m_lib.GetSymbol<
      decltype(bigvoice_Webserver_GetAppHealth      ) >("Webserver_GetAppHealth"     );
00466                 Webserver_FreeString        = m_lib.GetSymbol< decltype(bigvoice_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00467                 #endif // BIGVOICE_HAS_WEBSERVER
00468
00469 #ifdef _WIN32
00470                 // added 2025-05-28
00471                 SetInstanceTracing                = m_lib.GetSymbol<
      decltype(bigvoice_SetInstanceTracing               ) >("SetInstanceTracing"               );
00472                 SetInstanceTracingProcessActivity = m_lib.GetSymbol<
      decltype(bigvoice_SetInstanceTracingProcessActivity) >("SetInstanceTracingProcessActivity");
00473 #endif // _WIN32
00474             }  catch (std::runtime_error& ) {
00475                 // Ignored, handler will take care
00476             }
00477
00478             m_bOK=true;
00479             return true;
00480         }
00487         bool IsOk() const {
00488             return m_bOK;
00489         }
00490
00491     };
00492
00493
00494
00495     static constexpr const char *process_name = "SOUND4 BIGVOICE.CL";
00496     static constexpr const char *process_shortname = "bigvoice";
00497     using SampleFormat = bigvoice_SampleFormat;
00502     const size_t ChunkFrames=BIGVOICE_AUDIOFRAME_COUNT;
00507     const size_t InputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT;
00512     const size_t OutputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT;
00517     const size_t InputSampleSize=ChunkFrames*InputChannels;
00522     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00527     const size_t SampleRate=BIGVOICE_SAMPLE_RATE;
00528
00532     namespace helper {
00533         #ifdef _WIN32
00539         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00540             if (wstr.empty()) return {};
00541             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00542             std::string str(size, 0);
00543             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00544             return str;
00545         }
00551         static inline std::wstring UTF8ToWString(const std::string& str) {
00552             if (str.empty()) return std::wstring();
00553             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00554             std::wstring wstr(size, 0);
00555             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00556             return wstr;
00557         }
00558         #endif // _WIN32
00559
00565             template<typename T>
00566         struct SampleFormat {
00567         };
00568             template<>
00569         struct SampleFormat<int16_t> {
00570             const bigvoice_SampleFormat format = S16_NATIVE;
00571         };
00572             template<>
00573         struct SampleFormat<int32_t> {
00574             const bigvoice_SampleFormat format = S32_NATIVE;
00575         };
00576             template<>
00577         struct SampleFormat<float> {
00578             const bigvoice_SampleFormat format = F32_NATIVE;
```

```
00579         };
00580
00581         // ------------------------------------------------------------
00593         static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
      size_t nSpl, bigvoice_SampleFormat fmt) {
00594             dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00595         }
00596
00608         static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
      size_t nSpl, bigvoice_SampleFormat fmt) {
00609             dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00610         }
00611
00622         static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00623             dynlib.StereoToMono(input, output, nFrame);
00624         }
00625
00636         static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00637             dynlib.MonoToStereo(input, output, nFrame);
00638         }
00639
00651         static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
      *inputR, float *output, size_t nFrame) {
00652             dynlib.StereoToMono_Planar(inputL, inputR, output, nFrame);
00653         }
00654
00666         static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
      float *outputR, size_t nFrame) {
00667             dynlib.MonoToStereo_Planar(input, outputL, outputR, nFrame);
00668         }
00669
00677         static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
      *output) {
00678             dynlib.AudioMonoFromLiveStereo(payload, output);
00679         }
00680
00688         static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
      *payload) {
00689             dynlib.AudioMonoToLiveStereo(input, payload);
00690         }
00691
00692     }; // namespace helper
00693
00700     static inline std::string GetVersion(CDynLib& dynlib)
00701         {
00702             return std::string(dynlib.GetVersion());
00703         }
00704
00716     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00717         {
00718             return dynlib.GetChunkSizeInFrames();
00719         }
00720
00730     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00731         {
00732             std::vector<unsigned int> list;
00733             if (dynlib.GetPossibleChunkSizeInFrames) {
00734                 for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00735                     list.push_back(*src);
00736                 }
00737             } else {
00738                 list.push_back(12);
00739             }
00740             return list;
00741         }
00742
00749     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00750         {
00751             return dynlib.GetChannelCount();
00752         }
00753
00762     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00763         {
00764             return dynlib.GetAudioInputCount();
00765         }
00766
00775     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00776         {
00777             return dynlib.GetAudioOutputCount();
00778         }
00779
00788     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00789         {
00790             return dynlib.GetSampleRate();
00791         }
```

```
00792
00800     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00801     {
00802         // Do some sanity checks
00803         if (dynlib.GetSampleRate()!=BIGVOICE_SAMPLE_RATE) {
00804             throw std::runtime_error("Bad library sampling rate");
00805         }
00806         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
00807             throw std::runtime_error("Bad library frame size");
00808         }
00809         if (dynlib.GetChannelCount()!=BIGVOICE_CHANNEL_COUNT) {
00810             throw std::runtime_error("Bad library channel count");
00811         }
00812         if (dynlib.GetAudioInputCount()!=BIGVOICE_AUDIO_INPUT_COUNT) {
00813             throw std::runtime_error("Bad library input count");
00814         }
00815         if (dynlib.GetAudioOutputCount()!=BIGVOICE_AUDIO_OUTPUT_COUNT) {
00816             throw std::runtime_error("Bad library output count");
00817         }
00818     }
00819
00827     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00828         {
00829             return std::string(dynlib.GetFormatName(fmt));
00830         }
00831
00839     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00840         {
00841             return dynlib.GetFormatFromName(name.c_str());
00842         }
00843
00851     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00852         {
00853             return dynlib.GetBytesFromFormat(fmt);
00854         }
00855
00859     using LogSeverity = bigvoice_LogSeverity;
00863     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00864
00868     static log_cb_t _log_cb;
00869     extern "C" {
00870         static inline void _log_cb_c(bigvoice_LogSeverity severity, const char *c_msg) {
00871             _log_cb(severity, std::string(c_msg));
00872         }
00873     }
00877     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00878         {
00879             dynlib.SetLogSeverity(severity);
00880         }
00884     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00885         {
00886             _log_cb=cb;
00887             dynlib.SetLoggerCallback(&_log_cb_c);
00888         }
00889
00890 #if BIGVOICE_HAS_CLOUDBUS
00896     class CBus {
00897         public:
00898             CBus(CDynLib& dynlib)
00899                 : m_dynlib(dynlib)
00900                 {
00901                     if (m_dynlib.NewBus) {
00902                         m_bus=m_dynlib.NewBus();
00903                     }
00904                 }
00905             ~CBus()
00906                 {
00907                     if (m_bus && m_dynlib.FreeBus) {
00908                         m_dynlib.FreeBus(m_bus);
00909                     }
00910                     m_bus = nullptr;
00911                 }
00912             bigvoice_CBus *Get() const { return m_bus; }
00913         private:
00914             CDynLib& m_dynlib;
00915             bigvoice_CBus *m_bus;
00916     };
00917 #endif // BIGVOICE_HAS_CLOUDBUS
00918
00924     class CPresetLoader {
00925     public:
00926         CPresetLoader() = default;
00927         virtual ~CPresetLoader() = default;
00928
00929         virtual bool IsReadOnly() = 0;
00930         virtual bool Exists(const std::filesystem::path &name) = 0;
00931         virtual bool Remove(const std::filesystem::path &name) = 0;
```

```
00932         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00933         virtual std::vector<std::filesystem::path> GetAll() = 0;
00934         virtual std::string Read(const std::filesystem::path &filename) = 0;
00935         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00936     };
00937     using PresetChange_Kind = bigvoice_PresetChange_Kind;
00938
00944     class CInstance {
00945     public:
00949         CInstance(CDynLib& dynlib)
00950         : m_dynlib(dynlib)
00951             {
00952                 params = m_dynlib.NewParameters();
00953                 if (!params) throw std::bad_alloc();
00954             }
00955         // non-copyable
00956         CInstance( const CInstance& ) = delete;
00957         CInstance& operator=( const CInstance& ) = delete;
00961         virtual ~CInstance()
00962             {
00963                 Stop();
00964                 if (params) m_dynlib.FreeParameters(params);
00965             }
00969        bool IsOk() const { return instance!=nullptr; }
00979         void SetParam(const std::string& name, const std::string& value)
00980             {
00981                 assert(params);
00982                 m_dynlib.SetParameter(params, name.c_str(), value.c_str());
00983             }
00990         std::string GetParam(const std::string& name)
00991             {
00992                 assert(params);
00993                 auto c_value=m_dynlib.GetParameter(params, name.c_str());
00994                 std::string ret(c_value);
00995                 m_dynlib.FreeParameterValue(c_value);
00996                 return ret;
00997             }
00998 #if BIGVOICE_HAS_CLOUDBUS
01007             void SetBus(const CBus& bus) {
01008                 assert(params);
01009                 if (m_dynlib.SetInstanceBus) {
01010                     m_dynlib.SetInstanceBus(params,bus.Get());
01011                 }
01012             }
01013 #endif // BIGVOICE_HAS_CLOUDBUS
01014
01025         void SetPresetManager(CPresetLoader *preset_manager)
01026             {
01027                 if (!preset_manager) return;
01028                 if (!m_dynlib.SetPresetManager) return;
01029                 assert(params);
01030                 m_dynlib.SetPresetManager(params,
01031                     bigvoice_custom_reader,
01032                     bigvoice_custom_reader_free,
01033                     bigvoice_custom_writer,
01034                     bigvoice_custom_exists,
01035                     bigvoice_custom_getall,
01036                     bigvoice_custom_getall_free,
01037                     bigvoice_custom_remove,
01038                     bigvoice_custom_rename,
01039                     preset_manager->IsReadOnly(),
01040                     preset_manager
01041                 );
01042             }
01043
01066         bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
      Access_Key_ID, const std::string& Access_Key_Secret, const std::filesystem::path& save_path, int
      json_port = 0, unsigned int frames_per_chunk=12)
01067             {
01068                 assert(!instance);
01069                 std::string l_save_path;
01070                 if (!save_path.empty()) {
01071                     l_save_path = save_path.u8string();
01072                 }
01073                 if (m_dynlib.InitProcess3) {
01074                     instance = m_dynlib.InitProcess3(LoginKey.c_str(), RadioName.c_str(),
      Access_Key_ID.c_str(), Access_Key_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
01075                 } else if (frames_per_chunk!=12) {
01076                     return false;
01077                 } else {
01078                     instance = m_dynlib.InitProcess2(LoginKey.c_str(), RadioName.c_str(),
      Access_Key_ID.c_str(), Access_Key_Secret.c_str(), l_save_path.c_str(), params);
01079                 }
01080                 if (!instance) return false;
01081                 if (!init_metadata.empty()) {
01082                     for (auto&& [key,value]: init_metadata) {
01083                         m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
```

```
01084                     }
01085                     init_metadata.clear();
01086                 }
01087                 update_thread = std::thread([this, json_port](){
01088                     OnUpdateThreadStart();
01089                     m_dynlib.StartUpdateThread(instance, json_port);
01090                     OnUpdateThreadStop();
01091                 });
01092                 int timeout=5000;
01093                 #ifdef DEBUG
01094                     timeout=60*1000; // For debugger pauses
01095                 #endif
01096                 if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
01097                     return false;
01098                 }
01099                 return true;
01100             }
01101
01109         unsigned int GetChunkFrames() {
01110             if (instance && m_dynlib.GetProcessChunkFrames) {
01111                 return m_dynlib.GetProcessChunkFrames(instance);
01112             }
01113             return 12;
01114         }
01115
01123         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01124             {
01125                 if (m_dynlib.PresetManager_InformChange) {
01126                     m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
      (bigvoice_PresetChange_Kind) change_kind);
01127                 }
01128             }
01129
01141         void SetMetadata(const std::string& key, const char* value)
01142             {
01143                 if (m_dynlib.SetMetadata) {
01144                     if (instance) {
01145                         m_dynlib.SetMetadata(instance, key.c_str(), value);
01146                     } else {
01147                         init_metadata.push_back( {key, value} );
01148                     }
01149                 }
01150             }
01151
01152
01162         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01163             {
01164                 if (instance) {
01165                     if (m_dynlib.SetMetadataMulti) {
01166                         typedef const char* pchar;
01167                         pchar* keyvalue=new pchar[2*list.size()+1];
01168                         size_t n=0;
01169                         for (auto&& [key,value]: list) {
01170                             keyvalue[2*n+0]=key.c_str();
01171                             keyvalue[2*n+1]=value;
01172                             n++;
01173                         }
01174                         keyvalue[2*n+0]=nullptr;
01175
01176                         bigvoice_SetMetadataMulti(instance, keyvalue);
01177
01178                         delete[] keyvalue;
01179                     } else {
01180                         for (auto&& [key,value]: list) {
01181                             SetMetadata(key, value);
01182                         }
01183                     }
01184                 } else {
01185                     for (auto&& [key,value]: list) {
01186                         if (value) {
01187                             init_metadata.push_back( {key, value} );
01188                         }
01189                     }
01190                 }
01191             }
01192
01198         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01199             {
01200                 std::vector< std::tuple<std::string,std::string> > values;
01201                 if (m_dynlib.GetMetadataInfos) {
01202                     const char** c_values = m_dynlib.GetMetadataInfos(instance);
01203                     if (c_values) {
01204                         for (const char** c_value=c_values; *c_value; ) {
01205                             std::string key(*c_value);
01206                             c_value++;
01207                             if (*c_value) {
```

```
01208                                 std::string descr(*c_value);
01209                                 c_value++;
01210                                 values.push_back( {key,descr} );
01211                             }
01212                         }
01213                         m_dynlib.FreeMetadataInfos(instance, c_values);
01214                     }
01215                 }
01216                 return values;
01217             }
01218
01222         virtual void OnUpdateThreadStart() {}
01226         virtual void OnUpdateThreadStop() {}
01227
01228 #if BIGVOICE_HAS_WEBSERVER
01242         bool StartWebServer(int http_port, int https_port=0)
01243             {
01244                 assert(instance);
01245                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01246                 webserver = m_dynlib.Webserver(http_port,https_port,instance);
01247                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01248                     return false;
01249                 }
01250                 int web_status=m_dynlib.Webserver_Status(webserver);
01251                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01252                     return false;
01253                 }
01254                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01255                     return false;
01256                 }
01257                 return true;
01258             }
01259
01267         void StopWebServer(int timeout_ms = 1000)
01268             {
01269                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01270                     m_dynlib.Webserver_Stop(webserver, timeout_ms);
01271                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01272                 }
01273             }
01274
01278         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
01279             {
01280                 assert(instance);
01281                 if (!m_dynlib.Webserver_SetAppHealth) return;
01282                 m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
01283             }
01287         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01288             {
01289                 assert(instance);
01290                 if (!m_dynlib.Webserver_GetAppHealth) return;
01291                 char* c_contenttype=nullptr;
01292                 char* c_content=nullptr;
01293                 m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01294                 contenttype=c_contenttype;
01295                 content=c_content;
01296                 m_dynlib.Webserver_FreeString(c_contenttype);
01297                 m_dynlib.Webserver_FreeString(c_content);
01298             }
01299 #endif // BIGVOICE_HAS_WEBSERVER
01300
01309         int Terminate()
01310             {
01311                 if (!instance) return 0;
01312                 return m_dynlib.TerminateProcess(instance);
01313             }
01314
01318         void Stop()
01319             {
01320                 if (!instance) return;
01321 #if BIGVOICE_HAS_WEBSERVER
01322                 StopWebServer();
01323 #endif // BIGVOICE_HAS_WEBSERVER
01324                 if (update_thread.joinable()) {
01325                     m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01326                     update_thread.join();
01327                 }
01328                 m_dynlib.ExitProcess(instance);
01329                 instance=nullptr;
01330             }
01331
01341         unsigned int GetEstimatedDelay()
01342             {
01343                 assert(instance);
01344                 return m_dynlib.GetEstimatedDelay(instance);
```

```
01345             }
01346
01359         std::array<float,InputSampleSize>& GetBufferIn()
01360             {
01361                 assert(instance);
01362                 float *buf=m_dynlib.GetBufferIn(instance);
01363                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
01364
01365             }
01377         std::array<float,OutputSampleSize>& GetBufferOut()
01378             {
01379                 assert(instance);
01380                 float *buf= m_dynlib.GetBufferOut(instance);
01381                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
01382             }
01386         void ProcessAudio()
01387             {
01388                 assert(instance);
01389                 m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
      m_dynlib.GetBufferOut(instance));
01390             }
01402         void ProcessAudio_Planar(float const * const *input, float * const *output)
01403             {
01404                 assert(instance);
01405                 m_dynlib.ProcessAudio_Planar(instance, input, output);
01406             }
01407
01419             template<typename T>
01420         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01421             {
01422                 assert(instance);
01423                 std::vector<T> output;
01424                 unsigned int out_offset=0;
01425                 unsigned int in_offset=0;
01426                 unsigned int todo = input.size();
01427                 while (todo>0) {
01428                     unsigned int left = AddAudio(&input[in_offset], todo);
01429                     unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01430                     output.resize(out_offset + out_avail);
01431                     GetAudio(&output[out_offset], out_avail);
01432                     out_offset+=out_avail;
01433                     in_offset += todo-left;
01434                     todo=left;
01435                 }
01436                 return output;
01437             }
01438
01444         class CClient {
01445             public:
01446                 CClient(bigvoice_CInstance* instance, CDynLib& dynlib)
01447                 : m_dynlib(dynlib)
01448                     {
01449                         assert(instance);
01450                         client=m_dynlib.NewClient(instance);
01451                         if (!client) throw std::bad_alloc();
01452                     }
01453                 // non-copyable
01454                 CClient( const CInstance& ) = delete;
01455                 CClient& operator=( const CInstance& ) = delete;
01456                 ~CClient()
01457                     {
01458                         if (client) {
01459                             m_dynlib.DeleteClient(client);
01460                             client=nullptr;
01461                         }
01462                     }
01470                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01471                     {
01472                         assert(client);
01473                         int need_save=0;
01474                         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
      &need_save);
01475                         if (!canswer) return {};
01476                         std::string answer(canswer);
01477                         m_dynlib.FreeJsonAnswer (canswer);
01478                         if (NeedSave) {
01479                             *NeedSave=(need_save!=0);
01480                         }
01481                         return answer;
01482                     }
01483             private:
01484                 CDynLib& m_dynlib;
01485                 bigvoice_CClientInstance *client = nullptr;
01486         };
01494         std::shared_ptr<CClient> NewClient()
01495             {
01496                 assert(instance);
```

```
01497                 return std::make_shared<CClient>(instance, m_dynlib);
01498             }
01499
01510         bool SaveState() {
01511             assert(instance);
01512             if (m_dynlib.SaveState(instance)!=0) {
01513                 return false;
01514             }
01515             return true;
01516         }
01517 #ifdef _WIN32
01524         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01525             if (m_dynlib.SetInstanceTracing) {
01526                 m_dynlib.SetInstanceTracing(params, tracing_provider, activity_guid);
01527             }
01528         }
01535         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01536             assert(instance);
01537             if (m_dynlib.SetInstanceTracingProcessActivity) {
01538                 m_dynlib.SetInstanceTracingProcessActivity(instance, activity_guid);
01539             }
01540         }
01541 #endif // _WIN32
01542     protected:
01543             template<typename T>
01544         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01545             {
01546                 assert(instance);
01547                 return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01548             }
01549             template<typename T>
01550         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01551             {
01552                 assert(instance);
01553                 return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01554             }
01555
01556     private:
01557         CDynLib& m_dynlib;
01558         bigvoice_CParameters* params = nullptr;
01559         bigvoice_CInstance* instance = nullptr;
01560         std::thread update_thread;
01561         std::vector< std::pair<std::string,std::string> > init_metadata;
01562 #if BIGVOICE_HAS_WEBSERVER
01563         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01564 #endif // BIGVOICE_HAS_WEBSERVER
01565     };
01566
01567 }; // namespace dyn
01568 }; // namespace bigvoice
01569 }; // namespace sound4
01570
01571 // bridge C callbacks to CPresetLoader
01572 extern "C" {
01573     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
01574         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01575         std::string content=preset_loader->Read(filename);
01576         return strdup(content.c_str());
01577     }
01578     static void bigvoice_custom_reader_free(char *content, void* userdata) {
01579         if (!content) return;
01580         free(content);
01581     }
01582     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01583         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01584         auto res=preset_loader->Write(filename,content);
01585         return res?0:-1;
01586     }
01587     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
01588         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01589         auto res=preset_loader->Exists(filename);
01590         return res?0:-1;
01591     }
01592     static fs_char** bigvoice_custom_getall(void* userdata) {
01593         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01594         auto res=preset_loader->GetAll();
01595         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01596         for (size_t n=0;n<res.size();n++) {
01597             all[n]=fs_strdup(res[n].c_str());
01598         }
01599         all[res.size()]=nullptr;
```

```
01600         return all;
01601     }
01602     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
01603         if (!all) return;
01604         for (fs_char** one=all;*one;one++) {
01605             free(*one);
01606         }
01607         free(all);
01608     }
01609     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
01610         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01611         auto res=preset_loader->Remove(filename);
01612         return res?0:-1;
01613     }
01614     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01615         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01616         auto res=preset_loader->Rename(from,to);
01617         return res?0:-1;
01618     }
01619
01620 };
```

## 15.29 sound4cl.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 <libloaderapi.h>
#include "sound4cl_cdef.h"
```

### Classes

- struct sound4::helper::SampleFormat< T >

  *Helper for Sample format types.*
- class sound4::helper::DynFuncHolder< T >

  *Helper to wrap a function pointer.*
- class sound4::helper::CDynLoader

  *Helper to load dynamic library.*
- class sound4::CProcessor

  *Dynamic library interface.*
- class sound4::CBus

  *Shared Bus.*
- class sound4::CPresetLoader

  *Custom preset handler helper.*
- class sound4::CInstance

  *Instance handling class.*
- class sound4::CInstance::CClient

### Namespaces

- namespace sound4
- namespace sound4::helper

### Macros

- #define UNIXLIKE

### Typedefs

- using sound4::log_cb_t = std::function< void(LogSeverity, const std::string &)>

### Enumerations

- enum sound4::SampleFormat {
  sound4::INVALID_FORMAT , sound4::S16_LE , sound4::S16_BE , sound4::S24_LE ,
  sound4::S24_BE , sound4::S32_LE , sound4::S32_BE , sound4::F32_LE ,
  sound4::F32_BE , sound4::S16_NATIVE , sound4::S24_NATIVE , sound4::S32_NATIVE ,
  sound4::F32_NATIVE }
- enum sound4::LogSeverity {
  sound4::none , sound4::fatal , sound4::error , sound4::warning ,
  sound4::info , sound4::verbose , sound4::verbose2 , sound4::verbose3 ,
  sound4::verbose4 , sound4::verbose5 }
- enum sound4::PresetChange_Kind { sound4::change_kind_created , sound4::change_kind_modified ,
  sound4::change_kind_deleted }

### Functions

- static std::string sound4::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::helper::UTF8ToWString (const std::string &str)
- static void sound4::_log_cb_c (sound4cl_LogSeverity severity, const char *c_msg)

### Variables

- static log_cb_t sound4::_log_cb

## 15.29.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file sound4cl.hpp.

## 15.29.2 Macro Definition Documentation

### 15.29.2.1 UNIXLIKE

```
#define UNIXLIKE
```

Definition at line 24 of file sound4cl.hpp.

# 15.30 sound4cl.hpp

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include <string>
00011 #include <exception>
00012 #include <thread>
00013 #include <functional>
00014 #include <memory>
00015 #include <array>
00016 #include <assert.h>
00017 #include <stdexcept>
00018 #include <filesystem>
00019 #include <string.h>
00020
00021 #if defined(__unix__) || defined(__APPLE__)
00022     #define UNIXLIKE 1
00023 #else
00024     #define UNIXLIKE 0
00025 #endif
00026
00027 #if UNIXLIKE
00028     #include <dlfcn.h>
00029 #elif defined(_WIN32)
00030     #include <libloaderapi.h>
00031     #ifdef _MSC_VER
00032         #pragma warning(disable : 4996)
00033     #endif
00034 #else
00035     #error "Unsupported OS"
00036 #endif
00037
00038 #include "sound4cl_cdef.h"
00039
00040     // bridge C callbacks to CPresetLoader
00041     extern "C" {
00042         static char *sound4cl_custom_reader(const fs_char *filename, void* userdata);
00043         static void sound4cl_custom_reader_free(char *content, void* userdata);
00044         static int sound4cl_custom_writer(const fs_char *filename, const char *content, void*
    userdata);
00045         static int sound4cl_custom_exists(const fs_char *filename, void* userdata);
00046         static fs_char** sound4cl_custom_getall(void* userdata);
00047         static void sound4cl_custom_getall_free(fs_char** all, void* userdata);
00048         static int sound4cl_custom_remove(const fs_char *filename, void* userdata);
00049         static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00050     };
00051
00055 namespace sound4 {
00056
00060     enum SampleFormat {
00061         INVALID_FORMAT=sound4cl_SampleFormat::INVALID_FORMAT,
00062         S16_LE        =sound4cl_SampleFormat::S16_LE,
00063         S16_BE        =sound4cl_SampleFormat::S16_BE,
00064         S24_LE        =sound4cl_SampleFormat::S24_LE,
00065         S24_BE        =sound4cl_SampleFormat::S24_BE,
00066         S32_LE        =sound4cl_SampleFormat::S32_LE,
00067         S32_BE        =sound4cl_SampleFormat::S32_BE,
00068         F32_LE        =sound4cl_SampleFormat::F32_LE,
00069         F32_BE        =sound4cl_SampleFormat::F32_BE,
00070         S16_NATIVE    =sound4cl_SampleFormat::S16_NATIVE,
00071         S24_NATIVE    =sound4cl_SampleFormat::S24_NATIVE,
00072         S32_NATIVE    =sound4cl_SampleFormat::S32_NATIVE,
00073         F32_NATIVE    =sound4cl_SampleFormat::F32_NATIVE,
00074     };
00075
00079     namespace helper {
00080         #ifdef _WIN32
00086         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00087             if (wstr.empty()) return {};
00088             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
    NULL);
00089             std::string str(size, 0);
00090             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00091             return str;
00092         }
00098         static inline std::wstring UTF8ToWString(const std::string& str) {
00099             if (str.empty()) return std::wstring();
00100             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00101             std::wstring wstr(size, 0);
00102             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00103             return wstr;
00104         }
00105         #endif // _WIN32
```

```
00106
00112             template<typename T>
00113         struct SampleFormat {
00114         };
00115             template<>
00116         struct SampleFormat<int16_t> {
00117             const sound4::SampleFormat format = S16_NATIVE;
00118         };
00119             template<>
00120         struct SampleFormat<int32_t> {
00121             const sound4::SampleFormat format = S32_NATIVE;
00122         };
00123             template<>
00124         struct SampleFormat<float> {
00125             const sound4::SampleFormat format = F32_NATIVE;
00126         };
00127
00133             template <typename T>
00134         class DynFuncHolder {
00135             // Storage definition, depending on OS
00136             #if UNIXLIKE
00137                 using DynFunc_t = void *;
00138             #else
00139                 using DynFunc_t = FARPROC;
00140             #endif
00141         private:
00142             DynFunc_t m_ptr = NULL;
00143         public:
00144             DynFuncHolder() = default;
00145             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00146             operator bool() const { return (m_ptr!=NULL); }
00147             bool IsOk() const { return m_ptr!=NULL; }
00148             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00149         };
00150
00155         class CDynLoader {
00156             // Storage definition, depending on OS
00157             #if UNIXLIKE
00158                 using DynLib_t = void *;
00159             #else
00160                 using DynLib_t = HMODULE;
00161             #endif
00162             private:
00163                 DynLib_t m_lib {};
00164             public:
00165                 CDynLoader() = default;
00166                 CDynLoader(CDynLoader&&) = default;
00167                 // No copy allowed
00168                 CDynLoader(const CDynLoader&) = delete;
00169                 CDynLoader& operator=(CDynLoader const&) = delete;
00170                 operator bool() const {
00171                     return (m_lib!=NULL);
00172                 }
00173                 bool IsOk() const {
00174                     return (m_lib!=NULL);
00175                 }
00176                 #if UNIXLIKE
00177                     void Close() {
00178                         if (IsOk()) {
00179                             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00180                             dlclose(m_lib);
00181                             #endif // __SANITIZE_ADDRESS__
00182                             m_lib=NULL;
00183                         }
00184                     }
00185                     bool Load(const std::filesystem::path& dynlib)
00186                     {
00187                         Close();
00188                         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00189                         return IsOk();
00190                     }
00191                         template <typename T>
00192                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00193                         auto ptr=dlsym(m_lib, name.c_str());
00194                         if (!ptr) {
00195                             throw std::runtime_error("Missing function in library");
00196                         }
00197                         return DynFuncHolder<T>(ptr);
00198                     }
00199                         template <typename T>
00200                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00201                         auto ptr=dlsym(m_lib, name.c_str());
00202                         return DynFuncHolder<T>(ptr);
00203                     }
00204                     static std::filesystem::path GetThisLibraryPath(void) {
00205                         static Dl_info info;
00206                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
```

```
00207                         auto rp=realpath(info.dli_fname, NULL);
00208                         if (!rp) {
00209                             #if defined(__unix__)
00210                                 rp=realpath("/proc/self/exe", NULL);
00211                             #elif defined(__APPLE__)
00212                                 // How to solve this for Apple ?
00213                             #endif
00214                         }
00215                         std::filesystem::path p(rp);
00216                         free(rp);
00217                         return p;
00218                     }
00219                 #else
00220                     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_S
00221                     {
00222                         Close();
00223                         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00224                         return IsOk();
00225                     }
00226                     void Close() {
00227                         if (IsOk()) {
00228                             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00229                             // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00230                             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00231                                 FreeLibraryAndExitThread(m_lib, 0);
00232                             #elif defined(SOUND4_CALL_FREELIBRARY)
00233                                 FreeLibrary(m_lib);
00234                             #endif // SOUND4_CALL_FREELIBRARY
00235                             m_lib=NULL;
00236                         }
00237                     }
00238                         template <typename T>
00239                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00240                         auto ptr=GetProcAddress(m_lib, name.c_str());
00241                         if (!ptr) {
00242                             throw std::runtime_error("Missing function in library");
00243                         }
00244                         return DynFuncHolder<T>(ptr);
00245                     }
00246                         template <typename T>
00247                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00248                         auto ptr=GetProcAddress(m_lib, name.c_str());
00249                         return DynFuncHolder<T>(ptr);
00250                     }
00251                     static std::filesystem::path GetThisLibraryPath(void) {
00252                         static wchar_t path[MAX_PATH]={};
00253                         HMODULE hm = NULL;
00254                         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00255                                 GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00256                                 (LPCWSTR) &GetThisLibraryPath, &hm))
00257                         {
00258                             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00259                                 return std::filesystem::path(path);
00260                             }
00261                         }
00262                         return {};
00263                     }
00264                 #endif
00265                 virtual ~CDynLoader() {
00266                     Close();
00267                 }
00268         }; // class CDynLoader
00269     }; // namespace helper
00270
00271
00275     enum LogSeverity {
00276        none     = sound4cl_none,
00277        fatal    = sound4cl_fatal,
00278        error    = sound4cl_error,
00279        warning  = sound4cl_warning,
00280        info     = sound4cl_info,
00281        verbose  = sound4cl_verbose,
00282        verbose2 = sound4cl_verbose2,
00283        verbose3 = sound4cl_verbose3,
00284        verbose4 = sound4cl_verbose4,
00285        verbose5 = sound4cl_verbose5,
00286     };
00287
00291     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00292
00296     static log_cb_t _log_cb;
00297     extern "C" {
00298         static inline void _log_cb_c(sound4cl_LogSeverity severity, const char *c_msg) {
00299             _log_cb((LogSeverity)severity, std::string(c_msg));
```

```
00300         }
00301     }
00302
00309     class CProcessor {
00310     private:
00311         helper::CDynLoader m_lib;
00312         bool m_bOK = false;
00313         std::string m_prefix; // The prefix of functions, depends on the library (cloud_, impact_...)
00314     protected:
00315         friend class CBus;
00316         friend class CInstance;
00317
00318         // All those wrapped functions have the same signature as there C sound4cl_XXX
00319         helper::DynFuncHolder< decltype(sound4cl_GetVersion              ) > fnGetVersion              ;
00320         helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames    ) > fnGetChunkSizeInFrames    ;
00321         helper::DynFuncHolder< decltype(sound4cl_GetChannelCount         ) > fnGetChannelCount         ;
00322         helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount      ) > fnGetAudioInputCount      ;
00323         helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount     ) > fnGetAudioOutputCount     ;
00324         helper::DynFuncHolder< decltype(sound4cl_GetSampleRate           ) > fnGetSampleRate           ;
00325         helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback       ) > fnSetLoggerCallback       ;
00326         helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity          ) > fnSetLogSeverity          ;
00327         helper::DynFuncHolder< decltype(sound4cl_NewParameters           ) > fnNewParameters           ;
00328         helper::DynFuncHolder< decltype(sound4cl_FreeParameters          ) > fnFreeParameters          ;
00329         helper::DynFuncHolder< decltype(sound4cl_SetParameter            ) > fnSetParameter            ;
00330         helper::DynFuncHolder< decltype(sound4cl_GetParameter            ) > fnGetParameter            ;
00331         helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue      ) > fnFreeParameterValue      ;
00332         helper::DynFuncHolder< decltype(sound4cl_InitProcess             ) > fnInitProcess             ;
00333         helper::DynFuncHolder< decltype(sound4cl_InitProcess2            ) > fnInitProcess2            ;
00334         helper::DynFuncHolder< decltype(sound4cl_TerminateProcess        ) > fnTerminateProcess        ;
00335         helper::DynFuncHolder< decltype(sound4cl_ExitProcess             ) > fnExitProcess             ;
00336         helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread       ) > fnStartUpdateThread       ;
00337         helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread        ) > fnStopUpdateThread        ;
00338         helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady   ) > fnWaitUpdateThreadReady   ;
00339         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio            ) > fnProcessAudio            ;
00340         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar     ) > fnProcessAudio_Planar     ;
00341         helper::DynFuncHolder< decltype(sound4cl_GetBufferIn             ) > fnGetBufferIn             ;
00342         helper::DynFuncHolder< decltype(sound4cl_GetBufferOut            ) > fnGetBufferOut            ;
00343         helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay       ) > fnGetEstimatedDelay       ;
00344         helper::DynFuncHolder< decltype(sound4cl_GetFormatName           ) > fnGetFormatName           ;
00345         helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName       ) > fnGetFormatFromName       ;
00346         helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat      ) > fnGetBytesFromFormat      ;
00347         helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame       ) > fnGetMaxPacketFrame       ;
00348         helper::DynFuncHolder< decltype(sound4cl_AddAudio                ) > fnAddAudio                ;
00349         helper::DynFuncHolder< decltype(sound4cl_AddPadAudio             ) > fnAddPadAudio             ;
00350         helper::DynFuncHolder< decltype(sound4cl_GetOutputCount          ) > fnGetOutputCount          ;
00351         helper::DynFuncHolder< decltype(sound4cl_GetAudio                ) > fnGetAudio                ;
00352         helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom        ) > fnAudioConvertFrom        ;
00353         helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo          ) > fnAudioConvertTo          ;
00354         helper::DynFuncHolder< decltype(sound4cl_StereoToMono            ) > fnStereoToMono            ;
00355         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo            ) > fnMonoToStereo            ;
00356         helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
00357         helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo   ) > fnAudioMonoToLiveStereo   ;
00358         helper::DynFuncHolder< decltype(sound4cl_NewClient               ) > fnNewClient               ;
00359         helper::DynFuncHolder< decltype(sound4cl_DeleteClient            ) > fnDeleteClient            ;
00360         helper::DynFuncHolder< decltype(sound4cl_ProcessJson             ) > fnProcessJson             ;
00361         helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer          ) > fnFreeJsonAnswer          ;
00362         helper::DynFuncHolder< decltype(sound4cl_SaveState               ) > fnSaveState               ;
00363
00364         // only for processes supporting webserver
00365         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp           ) > fnWebserver_tcp           ;
00366         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2          ) > fnWebserver_tcp2          ;
00367         helper::DynFuncHolder< decltype(sound4cl_Webserver               ) > fnWebserver               ;
00368         helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop          ) > fnWebserver_Stop          ;
00369         helper::DynFuncHolder< decltype(sound4cl_Webserver_Status        ) > fnWebserver_Status        ;
00370
00371         // added 2023-03-22
00372         helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar     ) > fnStereoToMono_Planar     ;
00373         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar     ) > fnMonoToStereo_Planar     ;
00374
00375         // added 2023-05-22
00376         helper::DynFuncHolder< decltype(sound4cl_SetMetadata             ) > fnSetMetadata             ;
00377         helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos        ) > fnGetMetadataInfos        ;
00378         helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos       ) > fnFreeMetadataInfos       ;
00379
00380         // added 2023-06-20
00381         helper::DynFuncHolder< decltype(sound4cl_SetPresetManager                ) >
      fnSetPresetManager      ;
00382         helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange      ) >
      fnPresetManager_InformChange      ;
00383
00384         // added 2023-09-21
00385         helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) >
      fnGetPossibleChunkSizeInFrames   ;
00386         helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) >
      fnGetProcessChunkFrames   ;
00387         helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3
      ;
```

```
00388
00389         // added 2023-06-26, only for processes supporting bus
00390         helper::DynFuncHolder< decltype(sound4cl_NewBus                 ) > fnNewBus      ;
00391         helper::DynFuncHolder< decltype(sound4cl_FreeBus                ) > fnFreeBus      ;
00392         helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus         ) > fnSetInstanceBus      ;
00393
00394         // added 2024-02-19
00395         helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti       ) > fnSetMetadataMulti
    ;
00396
00397         // added 2024-05-23
00398         helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) >
    fnWebserver_SetAppHealth      ;
00399         helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) >
    fnWebserver_GetAppHealth      ;
00400         helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString
    ;
00401
00402 #ifdef _WIN32
00403         // added 2025-05-28
00404         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracing                ) >
    fnSetInstanceTracing                ;
00405         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracingProcessActivity ) >
    fnSetInstanceTracingProcessActivity ;
00406 #endif // _WIN32
00407
00408             template <typename T>
00409         helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
00410             std::string prefix_name = m_prefix+name;
00411             return m_lib.GetSymbol<T>(prefix_name);
00412         }
00413
00414     public:
00415         CProcessor() = default;
00416
00424         bool Load(const std::filesystem::path& filepath) {
00425             m_lib.Close();
00426             // If a path is given, use it directly and do not try other path
00427             if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
00428                 return false;
00429             } else if (!m_lib.IsOk()) {
00430                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00431                 auto filename = filepath.filename();
00432                 #ifndef _WIN32
00433                 // Linux: if in a bin directory, try ../lib first
00434                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00435                     auto libdir = thisdir.parent_path() / "lib";
00436                     m_lib.Load(libdir / filename);
00437                 }
00438                 #endif // !_WIN32
00439                 if (!m_lib.IsOk()) {
00440                     // Search in the same directory this code is in
00441                     m_lib.Load(thisdir / filename);
00442                 }
00443                 if (!m_lib.IsOk()) {
00444                     // Try current path
00445                     std::error_code ec;
00446                     auto p = std::filesystem::current_path(ec);
00447                     if (!ec) {
00448                         m_lib.Load(p / filename);
00449                     }
00450                 }
00451                 if (!m_lib.IsOk()) {
00452                     return false;
00453                 }
00454             }
00455             // Get information from the library itself
00456             try {
00457                 auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             )
    >("SOUND4_GetProcessInfo"   );
00458                 auto info = GetInfo();
00459                 m_prefix = info->prefix;
00460             } catch (std::runtime_error& ) {
00461                 // Compatibility: search in known libraries without this function
00462                 #ifdef _WIN32
00463                     std::string filename = helper::WStringToUTF8(filepath.stem());
00464                 #else // !_WIN32
00465                     std::string filename = filepath.stem();
00466                     // remove lib prefix
00467                     filename=filename.substr(3);
00468                 #endif // ! _WIN32
00469                 if (filename=="sound4.x1.cloud") {
00470                     m_prefix="cloudx1_";
00471                 } else if (filename=="sound4.impact.cl") {
00472                     m_prefix="impact_";
00473                 } else if (filename=="sound4.bigvoice.cl") {
00474                     m_prefix="bigvoice_";
```

```
00475                 } else {
00476                     return false;
00477                 }
00478             }
00479 
00480             // Load all C functions
00481             try {
00482                 fnGetVersion              = GetPrefixSymbol< decltype(sound4cl_GetVersion
      ) >("GetVersion"              );
00483                 fnGetChunkSizeInFrames    = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
      ) >("GetChunkSizeInFrames"    );
00484                 fnGetChannelCount         = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
      ) >("GetChannelCount"         );
00485                 fnGetAudioInputCount      = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
      ) >("GetAudioInputCount"      );
00486                 fnGetAudioOutputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
      ) >("GetAudioOutputCount"     );
00487                 fnGetSampleRate           = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
      ) >("GetSampleRate"           );
00488                 fnSetLoggerCallback       = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
      ) >("SetLoggerCallback"       );
00489                 fnSetLogSeverity          = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
      ) >("SetLogSeverity"          );
00490                 fnNewParameters           = GetPrefixSymbol< decltype(sound4cl_NewParameters
      ) >("NewParameters"           );
00491                 fnFreeParameters          = GetPrefixSymbol< decltype(sound4cl_FreeParameters
      ) >("FreeParameters"          );
00492                 fnSetParameter            = GetPrefixSymbol< decltype(sound4cl_SetParameter
      ) >("SetParameter"            );
00493                 fnGetParameter            = GetPrefixSymbol< decltype(sound4cl_GetParameter
      ) >("GetParameter"            );
00494                 fnFreeParameterValue      = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
      ) >("FreeParameterValue"      );
00495                 fnInitProcess             = GetPrefixSymbol< decltype(sound4cl_InitProcess
      ) >("InitProcess"             );
00496                 fnInitProcess2            = GetPrefixSymbol< decltype(sound4cl_InitProcess2
      ) >("InitProcess2"            );
00497                 fnTerminateProcess        = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
      ) >("TerminateProcess"        );
00498                 fnExitProcess             = GetPrefixSymbol< decltype(sound4cl_ExitProcess
      ) >("ExitProcess"             );
00499                 fnStartUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
      ) >("StartUpdateThread"       );
00500                 fnStopUpdateThread        = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
      ) >("StopUpdateThread"        );
00501                 fnWaitUpdateThreadReady   = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
      ) >("WaitUpdateThreadReady"   );
00502                 fnProcessAudio            = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
      ) >("ProcessAudio"            );
00503                 fnProcessAudio_Planar     = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
      ) >("ProcessAudio_Planar"     );
00504                 fnGetBufferIn             = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
      ) >("GetBufferIn"             );
00505                 fnGetBufferOut            = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
      ) >("GetBufferOut"            );
00506                 fnGetEstimatedDelay       = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
      ) >("GetEstimatedDelay"       );
00507                 fnGetFormatName           = GetPrefixSymbol< decltype(sound4cl_GetFormatName
      ) >("GetFormatName"           );
00508                 fnGetFormatFromName       = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
      ) >("GetFormatFromName"       );
00509                 fnGetBytesFromFormat      = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
      ) >("GetBytesFromFormat"      );
00510                 fnGetMaxPacketFrame       = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
      ) >("GetMaxPacketFrame"       );
00511                 fnAddAudio                = GetPrefixSymbol< decltype(sound4cl_AddAudio
      ) >("AddAudio"                );
00512                 fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
      ) >("AddPadAudio"             );
00513                 fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
      ) >("GetOutputCount"          );
00514                 fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
      ) >("GetAudio"                );
00515                 fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
      ) >("AudioConvertFrom"        );
00516                 fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
      ) >("AudioConvertTo"          );
00517                 fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
      ) >("StereoToMono"            );
00518                 fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo
      ) >("MonoToStereo"            );
00519                 fnAudioMonoFromLiveStereo = GetPrefixSymbol<
      decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
00520                 fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo
      ) >("AudioMonoToLiveStereo"   );
00521                 fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient
      ) >("NewClient"               );
```

```
00522                 fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient
      ) >("DeleteClient"           );
00523                 fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson
      ) >("ProcessJson"            );
00524                 fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer
      ) >("FreeJsonAnswer"         );
00525                 fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState
      ) >("SaveState"              );
00526             } catch (std::runtime_error& ) {
00527                 return false;
00528             }
00529             // Webserver : not for all projects
00530             try {
00531                 fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp
      ) >("Webserver_tcp"          );
00532                 fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2
      ) >("Webserver_tcp2"         );
00533                 fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver
      ) >("Webserver"              );
00534                 fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop
      ) >("Webserver_Stop"         );
00535                 fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status
      ) >("Webserver_Status"       );
00536             } catch (std::runtime_error& ) {
00537                 // Ignored, handler will take care
00538             }
00539             // Stereo/Mono conversion : must be present
00540             try {
00541                 // added 2023-03-22
00542                 fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar
      ) >("StereoToMono_Planar"    );
00543                 fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar
      ) >("MonoToStereo_Planar"    );
00544
00545             } catch (std::runtime_error& ) {
00546                 return false;
00547             }
00548             // C functions allowed to be missing
00549             try {
00550                 // added 2023-05-22
00551                 fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata
      ) >("SetMetadata"            );
00552                 fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos
      ) >("GetMetadataInfos"       );
00553                 fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos
      ) >("FreeMetadataInfos"      );
00554                 // added 2023-06-20
00555                 fnSetPresetManager            = GetPrefixSymbol< decltype(sound4cl_SetPresetManager
      ) >("SetPresetManager"               );
00556                 fnPresetManager_InformChange  = GetPrefixSymbol<
      decltype(sound4cl_PresetManager_InformChange     ) >("PresetManager_InformChange"    );
00557
00558                 // added 2023-09-21
00559                 fnGetPossibleChunkSizeInFrames = GetPrefixSymbol<
      decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
00560                 fnGetProcessChunkFrames        = GetPrefixSymbol<
      decltype(sound4cl_GetProcessChunkFrames         ) >("GetProcessChunkFrames"         );
00561                 fnInitProcess3                 = GetPrefixSymbol< decltype(sound4cl_InitProcess3
      ) >("InitProcess3"                   );
00562             }  catch (std::runtime_error& ) {
00563                 // Ignored, handler will take care
00564             }
00565
00566             // bus: not for all projects
00567             try {
00568                 // added 2023-06-26
00569                 fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus
      ) >("NewBus"                 );
00570                 fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus
      ) >("FreeBus"                );
00571                 fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus
      ) >("SetInstanceBus"         );
00572             }  catch (std::runtime_error& ) {
00573                 // Ignored, handler will take care
00574             }
00575
00576             // Other optional
00577             try {
00578                 // added 2024-02-19
00579                 fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti
      ) >("SetMetadataMulti"       );
00580
00581             }  catch (std::runtime_error& ) {
00582                 // Ignored, handler will take care
00583             }
00584
00585             // Other optional, web server only
```

```
00586             try {
00587                 // added 2024-05-23
00588                 fnWebserver_SetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
00589                 fnWebserver_GetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
00590                 fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00591             }  catch (std::runtime_error& ) {
00592                 // Ignored, handler will take care
00593             }
00594
00595 #ifdef _WIN32
00596             // Other optional, windows tracing
00597             try {
00598                 // added 2025-05-28
00599                 fnSetInstanceTracing               = GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracing                  ) >("SetInstanceTracing"               );
00600                 fnSetInstanceTracingProcessActivity= GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracingProcessActivity    ) >("SetInstanceTracingProcessActivity");
00601             }  catch (std::runtime_error& ) {
00602                 // Ignored, handler will take care
00603             }
00604 #endif // _WIN32
00605
00606             m_bOK=true;
00607             return true;
00608         }
00615         bool IsOk() const {
00616             return m_bOK;
00617         }
00618
00619         // ------------------------------------------------------------
00626         // ------------------------------------------------------------
00637         void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
00638             { fnAudioConvertFrom(payload, output, nSpl, (sound4cl_SampleFormat)fmt); }
00639
00650         void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
00651             { fnAudioConvertTo(input, payload, nSpl, (sound4cl_SampleFormat)fmt); }
00652
00662         void StereoToMono(const float *input, float *output, size_t nFrame)
00663             { fnStereoToMono(input, output, nFrame); }
00664
00674         void MonoToStereo(const float *input, float *output, size_t nFrame)
00675             { fnMonoToStereo(input, output, nFrame); }
00676
00687         void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame)
00688             { fnStereoToMono_Planar(inputL, inputR, output, nFrame); }
00689
00700         void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame)
00701             { fnMonoToStereo_Planar(input, outputL, outputR, nFrame); }
00702
00709         void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
00710             { fnAudioMonoFromLiveStereo(payload, output); }
00711
00718         void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
00719             { fnAudioMonoToLiveStereo(input, payload); }
00720
00728         std::string GetVersion()
00729             { return std::string(fnGetVersion()); }
00730
00741         unsigned int GetChunkSizeInFrames()
00742             { return fnGetChunkSizeInFrames(); }
00743
00752         std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00753             {
00754                 std::vector<unsigned int> list;
00755                 if (fnGetPossibleChunkSizeInFrames) {
00756                     for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
00757                         list.push_back(*src);
00758                     }
00759                 } else {
00760                     list.push_back(fnGetChunkSizeInFrames());
00761                 }
00762                 return list;
00763             }
00764
00770         unsigned int GetChannelCount()
00771             { return fnGetChannelCount(); }
00772
00780         unsigned int GetAudioInputCount()
00781             { return fnGetAudioInputCount(); }
00782
00790         unsigned int GetAudioOutputCount()
00791             { return fnGetAudioOutputCount(); }
00792
```

```
00800         unsigned int GetSampleRate()
00801             { return fnGetSampleRate(); }
00802
00809         std::string GetFormatName( const SampleFormat fmt)
00810             { return std::string(fnGetFormatName((sound4cl_SampleFormat)fmt)); }
00811
00818         SampleFormat GetFormatFromName( const std::string& name)
00819             { return SampleFormat(fnGetFormatFromName(name.c_str())); }
00820
00827         unsigned int GetBytesFromFormat( const SampleFormat fmt)
00828             { return fnGetBytesFromFormat((sound4cl_SampleFormat)fmt); }
00829
00833         void SetLogSeverity(LogSeverity severity)
00834             { fnSetLogSeverity((sound4cl_LogSeverity)severity); }
00835
00839         void SetLoggerCallback(log_cb_t cb)
00840             { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
00841     };
00842
00848     class CBus {
00849         public:
00850             CBus(CProcessor& dynlib)
00851                 : m_dynlib(dynlib)
00852                 {
00853                     if (m_dynlib.fnNewBus) {
00854                         m_bus=m_dynlib.fnNewBus();
00855                     }
00856                 }
00857             ~CBus()
00858                 {
00859                     if (m_bus && m_dynlib.fnFreeBus) {
00860                         m_dynlib.fnFreeBus(m_bus);
00861                     }
00862                     m_bus = nullptr;
00863                 }
00864         protected:
00865             friend class CInstance;
00866             sound4cl_CBus *Get() const { return m_bus; }
00867         private:
00868             CProcessor& m_dynlib;
00869             sound4cl_CBus *m_bus;
00870     };
00871
00877     class CPresetLoader {
00878     public:
00879         CPresetLoader() = default;
00880         virtual ~CPresetLoader() = default;
00881
00882         virtual bool IsReadOnly() = 0;
00883         virtual bool Exists(const std::filesystem::path &name) = 0;
00884         virtual bool Remove(const std::filesystem::path &name) = 0;
00885         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00886         virtual std::vector<std::filesystem::path> GetAll() = 0;
00887         virtual std::string Read(const std::filesystem::path &filename) = 0;
00888         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00889     };
00890     enum PresetChange_Kind {
00891         change_kind_created  = sound4cl_PresetChange_Kind::change_kind_created,
00892         change_kind_modified = sound4cl_PresetChange_Kind::change_kind_modified,
00893         change_kind_deleted  = sound4cl_PresetChange_Kind::change_kind_deleted,
00894     };
00900     class CInstance {
00901     public:
00907         CInstance(CProcessor& dynlib)
00908         : m_dynlib(dynlib)
00909             {
00910                 params = m_dynlib.fnNewParameters();
00911                 if (!params) throw std::bad_alloc();
00912             }
00913
00922         CInstance(CProcessor& dynlib, void* _instance)
00923         : m_dynlib(dynlib)
00924         , instance(reinterpret_cast<sound4cl_CInstance*>(_instance))
00925         , m_bOwned(false)
00926             {
00927             }
00928         // non-copyable
00929         CInstance( const CInstance& ) = delete;
00930         CInstance& operator=( const CInstance& ) = delete;
00934         virtual ~CInstance()
00935             {
00936                 Stop();
00937                 if (params) m_dynlib.fnFreeParameters(params);
00938             }
00942        bool IsOk() const { return instance!=nullptr; }
00952         void SetParam(const std::string& name, const std::string& value)
00953             {
```

```
00954                 assert(params);
00955                 m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
00956             }
00963         std::string GetParam(const std::string& name)
00964             {
00965                 assert(params);
00966                 auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
00967                 std::string ret(c_value);
00968                 m_dynlib.fnFreeParameterValue(c_value);
00969                 return ret;
00970             }
00971
00980         void SetBus(const CBus& bus) {
00981             assert(params);
00982             if (m_dynlib.fnSetInstanceBus) {
00983                 m_dynlib.fnSetInstanceBus(params,bus.Get());
00984             }
00985         }
00986
00997         void SetPresetManager(CPresetLoader *preset_manager)
00998             {
00999                 if (!preset_manager) return;
01000                 if (!m_dynlib.fnSetPresetManager) return;
01001                 assert(params);
01002                 m_dynlib.fnSetPresetManager(params,
01003                     sound4cl_custom_reader,
01004                     sound4cl_custom_reader_free,
01005                     sound4cl_custom_writer,
01006                     sound4cl_custom_exists,
01007                     sound4cl_custom_getall,
01008                     sound4cl_custom_getall_free,
01009                     sound4cl_custom_remove,
01010                     sound4cl_custom_rename,
01011                     preset_manager->IsReadOnly(),
01012                     preset_manager
01013                 );
01014             }
01015
01038         bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
      Access_Key_ID, const std::string& Access_Key_Secret, const std::filesystem::path& save_path, int
      json_port = 0, unsigned int frames_per_chunk=0)
01039             {
01040                 assert(!instance);
01041                 std::string l_save_path;
01042                 if (!save_path.empty()) {
01043                     l_save_path = save_path.u8string();
01044                 }
01045                 if (frames_per_chunk==0) {
01046                     frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
01047                 }
01048                 if (m_dynlib.fnInitProcess3) {
01049                     instance = m_dynlib.fnInitProcess3(LoginKey.c_str(), RadioName.c_str(),
      Access_Key_ID.c_str(), Access_Key_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
01050                 } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
01051                     return false;
01052                 } else {
01053                     instance = m_dynlib.fnInitProcess2(LoginKey.c_str(), RadioName.c_str(),
      Access_Key_ID.c_str(), Access_Key_Secret.c_str(), l_save_path.c_str(), params);
01054                 }
01055                 if (!instance) return false;
01056                 if (!init_metadata.empty()) {
01057                     for (auto&& [key,value]: init_metadata) {
01058                         m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
01059                     }
01060                     init_metadata.clear();
01061                 }
01062                 update_thread = std::thread([this, json_port](){
01063                     OnUpdateThreadStart();
01064                     m_dynlib.fnStartUpdateThread(instance, json_port);
01065                     OnUpdateThreadStop();
01066                 });
01067                 int timeout=5000;
01068                 #ifdef DEBUG
01069                     timeout=60*1000; // For debugger pauses
01070                 #endif
01071                 if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
01072                     return false;
01073                 }
01074                 return true;
01075             }
01076
01084         unsigned int GetChunkFrames() {
01085             if (instance && m_dynlib.fnGetProcessChunkFrames) {
01086                 return m_dynlib.fnGetProcessChunkFrames(instance);
01087             }
01088             return m_dynlib.fnGetChunkSizeInFrames();
01089         }
```

```
01090
01098         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01099             {
01100                 if (m_dynlib.fnPresetManager_InformChange) {
01101                     m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(),
      (sound4cl_PresetChange_Kind) change_kind);
01102                 }
01103             }
01104
01116         void SetMetadata(const std::string& key, const char* value)
01117             {
01118                 if (m_dynlib.fnSetMetadata) {
01119                     if (instance) {
01120                         m_dynlib.fnSetMetadata(instance, key.c_str(), value);
01121                     } else {
01122                         init_metadata.push_back( {key, value} );
01123                     }
01124                 }
01125             }
01126
01136         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01137             {
01138                 if (instance) {
01139                     if (m_dynlib.fnSetMetadataMulti) {
01140                         typedef const char* pchar;
01141                         pchar* keyvalue=new pchar[2*list.size()+1];
01142                         size_t n=0;
01143                         for (auto&& [key,value]: list) {
01144                             keyvalue[2*n+0]=key.c_str();
01145                             keyvalue[2*n+1]=value;
01146                             n++;
01147                         }
01148                         keyvalue[2*n+0]=nullptr;
01149
01150                         sound4cl_SetMetadataMulti(instance, keyvalue);
01151
01152                         delete[] keyvalue;
01153                     } else {
01154                         for (auto&& [key,value]: list) {
01155                             SetMetadata(key, value);
01156                         }
01157                     }
01158                 } else {
01159                     for (auto&& [key,value]: list) {
01160                         if (value) {
01161                             init_metadata.push_back( {key, value} );
01162                         }
01163                     }
01164                 }
01165             }
01166
01172         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01173             {
01174                 std::vector< std::tuple<std::string,std::string> > values;
01175                 if (m_dynlib.fnGetMetadataInfos) {
01176                     const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
01177                     if (c_values) {
01178                         for (const char** c_value=c_values; *c_value; ) {
01179                             std::string key(*c_value);
01180                             c_value++;
01181                             if (*c_value) {
01182                                 std::string descr(*c_value);
01183                                 c_value++;
01184                                 values.push_back( {key,descr} );
01185                             }
01186                         }
01187                         m_dynlib.fnFreeMetadataInfos(instance, c_values);
01188                     }
01189                 }
01190                 return values;
01191             }
01192
01196         virtual void OnUpdateThreadStart() {}
01200         virtual void OnUpdateThreadStop() {}
01201
01202
01216         bool StartWebServer(int http_port, int https_port=0)
01217             {
01218                 assert(instance);
01219                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01220                 if (!m_dynlib.fnWebserver) {
01221                     return false;
01222                 }
01223                 webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
01224                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01225                     return false;
```

```
01226                 }
01227                 int web_status=m_dynlib.fnWebserver_Status(webserver);
01228                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01229                     return false;
01230                 }
01231                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01232                     return false;
01233                 }
01234                 return true;
01235             }
01236 
01244             void StopWebServer(int timeout_ms = 1000)
01245             {
01246                 if (!m_dynlib.fnWebserver_Stop) return;
01247                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01248                     m_dynlib.fnWebserver_Stop(webserver, timeout_ms);
01249                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01250                 }
01251             }
01252 
01256         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
01257             {
01258                 assert(instance);
01259                 if (!m_dynlib.fnWebserver_SetAppHealth) return;
01260                 m_dynlib.fnWebserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
    content.c_str());
01261             }
01265         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01266             {
01267                 assert(instance);
01268                 if (!m_dynlib.fnWebserver_GetAppHealth) return;
01269                 char* c_contenttype=nullptr;
01270                 char* c_content=nullptr;
01271                 m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01272                 contenttype=c_contenttype;
01273                 content=c_content;
01274                 m_dynlib.fnWebserver_FreeString(c_contenttype);
01275                 m_dynlib.fnWebserver_FreeString(c_content);
01276             }
01285         int Terminate()
01286             {
01287                 if (!instance) return 0;
01288                 return m_dynlib.fnTerminateProcess(instance);
01289             }
01290 
01296         void Stop()
01297             {
01298                 if (!instance) return;
01299                 if (m_bOwned) {
01300                     StopWebServer();
01301                     if (update_thread.joinable()) {
01302                         m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
01303                         update_thread.join();
01304                     }
01305                     m_dynlib.fnExitProcess(instance);
01306                 }
01307                 instance=nullptr;
01308             }
01309 
01319         unsigned int GetEstimatedDelay()
01320             {
01321                 assert(instance);
01322                 return m_dynlib.fnGetEstimatedDelay(instance);
01323             }
01324 
01337         float* GetBufferIn()
01338             {
01339                 assert(instance);
01340                 return m_dynlib.fnGetBufferIn(instance);
01341 
01342             }
01354         float* GetBufferOut()
01355             {
01356                 assert(instance);
01357                 return m_dynlib.fnGetBufferOut(instance);
01358             }
01362         void ProcessAudio()
01363             {
01364                 assert(instance);
01365                 m_dynlib.fnProcessAudio(instance, GetBufferIn(), GetBufferOut());
01366             }
01378         void ProcessAudio_Planar(float const * const *input, float * const *output)
01379             {
01380                 assert(instance);
01381                 m_dynlib.fnProcessAudio_Planar(instance, input, output);
01382             }
```

```
01383
01395             template<typename T>
01396         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01397             {
01398                 assert(instance);
01399                 std::vector<T> output;
01400                 unsigned int out_offset=0;
01401                 unsigned int in_offset=0;
01402                 unsigned int todo = input.size();
01403                 while (todo>0) {
01404                     unsigned int left = AddAudio(&input[in_offset], todo);
01405                     unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
01406                     output.resize(out_offset + out_avail);
01407                     GetAudio(&output[out_offset], out_avail);
01408                     out_offset+=out_avail;
01409                     in_offset += todo-left;
01410                     todo=left;
01411                 }
01412                 return output;
01413             }
01414
01420         class CClient {
01421             public:
01422                 CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
01423                 : m_dynlib(dynlib)
01424                     {
01425                         assert(instance);
01426                         client=m_dynlib.fnNewClient(instance);
01427                         if (!client) throw std::bad_alloc();
01428                     }
01429                 // non-copyable
01430                 CClient( const CInstance& ) = delete;
01431                 CClient& operator=( const CInstance& ) = delete;
01432                 ~CClient()
01433                     {
01434                         if (client) {
01435                             m_dynlib.fnDeleteClient(client);
01436                             client=nullptr;
01437                         }
01438                     }
01446                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01447                     {
01448                         assert(client);
01449                         int need_save=0;
01450                         const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(),
      &need_save);
01451                         if (!canswer) return {};
01452                         std::string answer(canswer);
01453                         m_dynlib.fnFreeJsonAnswer (canswer);
01454                         if (NeedSave) {
01455                             *NeedSave=(need_save!=0);
01456                         }
01457                         return answer;
01458                     }
01459             private:
01460                 CProcessor& m_dynlib;
01461                 sound4cl_CClientInstance *client = nullptr;
01462         };
01470         std::shared_ptr<CClient> NewClient()
01471             {
01472                 assert(instance);
01473                 return std::make_shared<CClient>(instance, m_dynlib);
01474             }
01475
01486         bool SaveState() {
01487             assert(instance);
01488             if (!m_dynlib.fnSaveState || m_dynlib.fnSaveState(instance)!=0) {
01489                 return false;
01490             }
01491             return true;
01492         }
01493 #ifdef _WIN32
01500         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01501             if (m_dynlib.fnSetInstanceTracing) {
01502                 m_dynlib.fnSetInstanceTracing(params, tracing_provider, activity_guid);
01503             }
01504         }
01511         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01512             assert(instance);
01513             if (m_dynlib.fnSetInstanceTracingProcessActivity) {
01514                 m_dynlib.fnSetInstanceTracingProcessActivity(instance, activity_guid);
01515             }
01516         }
01517 #endif // _WIN32
01518     protected:
01519             template<typename T>
01520         unsigned int AddAudio(const T* payload, unsigned int nFrame)
```

```
01521             {
01522                 assert(instance);
01523                 return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01524             }
01525             template<typename T>
01526         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01527             {
01528                 assert(instance);
01529                 return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01530             }
01531
01532     private:
01533         CProcessor& m_dynlib;
01534         sound4cl_CParameters* params = nullptr;
01535         sound4cl_CInstance* instance = nullptr;
01536         std::thread update_thread;
01537         std::vector< std::pair<std::string,std::string> > init_metadata;
01538         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01539         bool m_bOwned = true;
01540     };
01541
01542 }; // namespace sound4
01543
01544
01545 // bridge C callbacks to CPresetLoader
01546 extern "C" {
01547     static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
01548         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01549         std::string content=preset_loader->Read(filename);
01550         return strdup(content.c_str());
01551     }
01552     static void sound4cl_custom_reader_free(char *content, void* userdata) {
01553         if (!content) return;
01554         free(content);
01555     }
01556     static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01557         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01558         auto res=preset_loader->Write(filename,content);
01559         return res?0:-1;
01560     }
01561     static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
01562         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01563         auto res=preset_loader->Exists(filename);
01564         return res?0:-1;
01565     }
01566     static fs_char** sound4cl_custom_getall(void* userdata) {
01567         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01568         auto res=preset_loader->GetAll();
01569         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01570         for (size_t n=0;n<res.size();n++) {
01571             all[n]=fs_strdup(res[n].c_str());
01572         }
01573         all[res.size()]=nullptr;
01574         return all;
01575     }
01576     static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
01577         if (!all) return;
01578         for (fs_char** one=all;*one;one++) {
01579             free(*one);
01580         }
01581         free(all);
01582     }
01583     static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
01584         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01585         auto res=preset_loader->Remove(filename);
01586         return res?0:-1;
01587     }
01588     static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01589         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01590         auto res=preset_loader->Rename(from,to);
01591         return res?0:-1;
01592     }
01593
01594 };
```

## 15.31 sound4cl_cdef.h File Reference

C function signature definition for all sound4.cl libraries.

```
#include <stdint.h>
#include <stringapiset.h>
#include <wchar.h>
```

**Macros**

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

**Typedefs**

- typedef wchar_t fs_char
- typedef struct _tlgProvider_t const * TraceLoggingHProvider

## 15.31.1 Macro Definition Documentation

### 15.31.1.1 SOUND4_INVALID_WEBSERVER_ID

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

Definition at line 65 of file sound4cl_cdef.h.

### 15.31.1.2 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 66 of file sound4cl_cdef.h.

### 15.31.1.3 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 67 of file sound4cl_cdef.h.

## 15.31.2 Typedef Documentation

### 15.31.2.1 fs_char

```
typedef wchar_t fs_char
```

Definition at line 18 of file sound4cl_cdef.h.

#### 15.31.2.2 TraceLoggingHProvider

```
typedef struct _tlgProvider_t const* TraceLoggingHProvider
```

Definition at line 210 of file sound4cl_cdef.h.

## 15.32 sound4cl_cdef.h

Go to the documentation of this file.

```
00001
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #ifdef __unix__
00010     #include <sys/types.h>
00011 #endif
00012
00013 #ifdef _WIN32
00014     #include <stringapiset.h>
00015     #include <wchar.h>
00017     #define fs_strdup wcsdup
00018     typedef wchar_t fs_char;
00019     #ifdef _MSC_VER
00020         #define __BYTE_ORDER__  REG_DWORD
00021         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00022         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00023     #endif
00024     #ifdef _MSC_VER
00025         #pragma warning(disable : 4996)
00026     #endif
00027 #else // !_WIN32
00029     #define fs_strdup strdup
00030     typedef char fs_char;
00031 #endif // !_WIN32
00032
00033 #ifdef __cplusplus
00034 extern "C" {
00035 #endif // __cplusplus
00036
00037 struct sound4cl_SOUND4_ProcessInfo {
00038     size_t struct_size;
00039     const char *name;
00040     const char *prefix;
00041     const char *shortname;
00042     const char *version;
00043 };
00044 enum sound4cl_LogSeverity {
00045     sound4cl_none     = 0,
00046     sound4cl_fatal    = 1,
00047     sound4cl_error    = 2,
00048     sound4cl_warning  = 3,
00049     sound4cl_info     = 4,
00050     sound4cl_verbose  = 5,
00051     sound4cl_verbose2 = 6,
00052     sound4cl_verbose3 = 7,
00053     sound4cl_verbose4 = 8,
00054     sound4cl_verbose5 = 9,
00055 };
00056 struct sound4cl_CInstance;
00057 struct sound4cl_CParameters;
00058 struct sound4cl_CBus;
00059 struct sound4cl_CClientInstance;
00060 enum sound4cl_PresetChange_Kind {
00061     change_kind_created = 1,
00062     change_kind_modified   ,
00063     change_kind_deleted    ,
00064 };
00065 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
00066 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
00067 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
00068
00072 enum sound4cl_SampleFormat {
00073     INVALID_FORMAT=0,
00074     S16_LE,
00075     S16_BE,
00076     S24_LE,
00077     S24_BE,
00078     S32_LE,
```

```
00079     S32_BE,
00080     F32_LE,
00081     F32_BE,
00082
00083     // Native definitions, only links to the physical ones
00084     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00085         S16_NATIVE=S16_LE,
00086         S24_NATIVE=S24_LE,
00087         S32_NATIVE=S32_LE,
00088         F32_NATIVE=F32_LE,
00089     #else
00090         S16_NATIVE=S16_BE,
00091         S24_NATIVE=S24_BE,
00092         S32_NATIVE=S32_BE,
00093         F32_NATIVE=F32_BE,
00094     #endif
00095 };
00096
00097 const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
00098 const char *sound4cl_GetVersion();
00099 unsigned int sound4cl_GetChunkSizeInFrames();
00100 unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
00101 unsigned int sound4cl_GetChannelCount();
00102 unsigned int sound4cl_GetAudioInputCount();
00103 unsigned int sound4cl_GetAudioOutputCount();
00104 unsigned int sound4cl_GetSampleRate();
00105 typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
00106 void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
00107 void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
00108 struct sound4cl_CParameters *sound4cl_NewParameters();
00109 void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
00110 void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
00111 const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
00112 void sound4cl_FreeParameterValue(const char *value);
00113 struct sound4cl_CInstance *sound4cl_InitProcess(
00114         const char *LoginKey,
00115         const char *RadioName,
00116         const char *Access_Key_ID,
00117         const char *Access_Key_Secret,
00118         const char *save_path
00119     );
00120 struct sound4cl_CInstance *sound4cl_InitProcess2(
00121         const char *LoginKey,
00122         const char *RadioName,
00123         const char *Access_Key_ID,
00124         const char *Access_Key_Secret,
00125         const char *save_path,
00126         const struct sound4cl_CParameters *parameters
00127     );
00128 struct sound4cl_CInstance *sound4cl_InitProcess3(
00129         const char *LoginKey,
00130         const char *RadioName,
00131         const char *Access_Key_ID,
00132         const char *Access_Key_Secret,
00133         const char *save_path,
00134         const struct sound4cl_CParameters *parameters,
00135         unsigned int frames_per_chunk
00136     );
00137 unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
00138 int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
00139 void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
00140 struct sound4cl_CBus* sound4cl_NewBus();
00141 void sound4cl_FreeBus(struct sound4cl_CBus *bus);
00142 void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
00143 typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
00144 typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
00145 typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00146 typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
00147 typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
00148 typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
00149 typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
00150 typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00151 void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
00152     , sound4cl_storage_reader reader, sound4cl_storage_reader_free
00153     , sound4cl_storage_writer writer
00154     , sound4cl_storage_exists exists
00155     , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
00156     , sound4cl_storage_remove remove
00157     , sound4cl_storage_rename rename
00158     , int IsReadOnly
00159     , void* userdata
00160     );
00161 void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
      *relative_path, enum sound4cl_PresetChange_Kind change_kind);
00162 void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
00163 void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
00164 const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
```

```
00165 void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
00166 void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
00167 void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
00168 int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
00169 void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
00170 void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
00171     , float const * const *input
00172     , float * const *output);
00173 float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
00174 float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
00175 unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
00176 const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
00177 enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
00178 unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
00179 unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
00180 unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
      int nFrame, enum sound4cl_SampleFormat fmt );
00181 unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
00182 unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
00183 unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
      max_nFrame, enum sound4cl_SampleFormat fmt );
00184 void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00185 void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00186 void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
00187 void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
00188 void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame);
00189 void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
00190 void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00191 void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00192 struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
00193 void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
00194 const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
      *need_save);
00195 void sound4cl_FreeJsonAnswer(const char *json_str);
00196 int sound4cl_SaveState(struct sound4cl_CInstance *instance);
00197 uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port);
00198 uint64_t sound4cl_Webserver_tcp2(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port, const struct sound4cl_CParameters *parameters);
00199 uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
      sound4cl_CInstance *instance);
00200 int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
00201 int sound4cl_Webserver_Status(uint64_t id);
00202 void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
      contenttype, const char* content);
00203 void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
      contenttype, char** content);
00204 void sound4cl_Webserver_FreeString(char* str);
00205
00206 //   Tracing (Windows only)
00207 #ifdef _WIN32
00208     #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00209         struct _tlgProvider_t;
00210         typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00211     #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00212 void sound4cl_SetInstanceTracing(struct sound4cl_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00213 void sound4cl_SetInstanceTracingProcessActivity(struct sound4cl_CInstance *instance, GUID
      activity_guid);
00214 #endif // _WIN32
00215
00216
00217 #ifdef __cplusplus
00218 } // extern "C"
00219 #endif // __cplusplus
```

# Index


_log_cb
  sound4, 86
  sound4::bigvoice, 92
  sound4::bigvoice::dyn, 101
_log_cb_c
  sound4, 85
  sound4::bigvoice, 88
  sound4::bigvoice::dyn, 95
~CBus
  sound4::bigvoice::CBus, 118
  sound4::bigvoice::dyn::CBus, 119
  sound4::CBus, 120
~CClient
  sound4::bigvoice::CInstance::CClient, 121
  sound4::bigvoice::dyn::CInstance::CClient, 123
  sound4::CInstance::CClient, 124
~CDynLoader
  sound4::bigvoice::dyn::helper::CDynLoader, 138
  sound4::helper::CDynLoader, 140
~CInstance
  sound4::bigvoice::CInstance, 144
  sound4::bigvoice::dyn::CInstance, 156
  sound4::CInstance, 169
~CPresetLoader
  sound4::bigvoice::CPresetLoader, 180
  sound4::bigvoice::dyn::CPresetLoader, 182
  sound4::CPresetLoader, 184
~MyPresetManager
  MyPresetManager, 196

ActivateAuxiliaryAudioID
  sound4::bigvoice::CInstance, 144
AddAudio
  sound4::bigvoice::CInstance, 144
  sound4::bigvoice::dyn::CDynLib, 127
  sound4::bigvoice::dyn::CInstance, 156
  sound4::CInstance, 170
AddPadAudio
  sound4::bigvoice::dyn::CDynLib, 127
Audio convert helpers, 29
  bigvoice_AudioConvertFrom, 29
  bigvoice_AudioConvertTo, 30
  bigvoice_AudioMonoFromLiveStereo, 30
  bigvoice_AudioMonoToLiveStereo, 30
  bigvoice_MonoToStereo, 32
  bigvoice_MonoToStereo_Planar, 32
  bigvoice_StereoToMono, 32
  bigvoice_StereoToMono_Planar, 33
Audio convert helpers (C++), 33
  AudioConvertFrom, 34
  AudioConvertTo, 34
  AudioMonoFromLiveStereo, 35
  AudioMonoToLiveStereo, 35
  MonoToStereo, 35
  MonoToStereo_Planar, 36
  StereoToMono, 36
  StereoToMono_Planar, 37
Audio managing (any audio), 37
  bigvoice_AddAudio, 39
  bigvoice_AddPadAudio, 39
  bigvoice_GetAudio, 39
  bigvoice_GetBytesFromFormat, 40
  bigvoice_GetFormatFromName, 40
  bigvoice_GetFormatName, 40
  bigvoice_GetMaxPacketFrame, 41
  bigvoice_GetOutputCount, 41
  bigvoice_SampleFormat, 38
  F32_BE, 38
  F32_LE, 38
  F32_NATIVE, 38
  INVALID_FORMAT, 38
  S16_BE, 38
  S16_LE, 38
  S16_NATIVE, 38
  S24_BE, 38
  S24_LE, 38
  S24_NATIVE, 38
  S32_BE, 38
  S32_LE, 38
  S32_NATIVE, 38
AudioConvertFrom
  Audio convert helpers (C++), 34
  sound4::bigvoice::dyn::CDynLib, 128
  sound4::bigvoice::dyn::helper, 104
  sound4::bigvoice::helper, 109
AudioConvertTo
  Audio convert helpers (C++), 34
  sound4::bigvoice::dyn::CDynLib, 128
  sound4::bigvoice::dyn::helper, 104
  sound4::bigvoice::helper, 109
AudioMonoFromLiveStereo
  Audio convert helpers (C++), 35
  sound4::bigvoice::dyn::CDynLib, 128
  sound4::bigvoice::dyn::helper, 105
  sound4::bigvoice::helper, 109
AudioMonoToLiveStereo
  Audio convert helpers (C++), 35
  sound4::bigvoice::dyn::CDynLib, 128
  sound4::bigvoice::dyn::helper, 105

sound4::bigvoice::helper, 110
Auxiliary outputs, 41
bigvoice_ActivateAuxiliaryAudioID, 42
bigvoice_FreeAuxiliaryAudioInfos, 42
bigvoice_GetAuxiliaryAudioEstimatedDelay, 43
bigvoice_GetAuxiliaryAudioOutputInfos, 43

bigvoice_ActivateAuxiliaryAudioID
Auxiliary outputs, 42
bigvoice_AddAudio
Audio managing (any audio), 39
bigvoice_AddPadAudio
Audio managing (any audio), 39
BIGVOICE_AUDIO_INPUT_COUNT
Global setup, 56
BIGVOICE_AUDIO_OUTPUT_COUNT
Global setup, 56
bigvoice_AudioConvertFrom
Audio convert helpers, 29
bigvoice_AudioConvertTo
Audio convert helpers, 30
BIGVOICE_AUDIOFRAME_COUNT
Global setup, 56
bigvoice_AudioMonoFromLiveStereo
Audio convert helpers, 30
bigvoice_AudioMonoToLiveStereo
Audio convert helpers, 30
bigvoice_CBus, 116
bigvoice_CClientInstance, 116
BIGVOICE_CHANNEL_COUNT
Global setup, 56
bigvoice_CInstance, 117
bigvoice_CParameters, 117
bigvoice_custom_exists
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_getall
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_getall_free
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_reader
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_reader_free
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_remove
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_rename
sound4.bigvoice.cl.hpp, 235
sound4.bigvoice.cl_dyn.hpp, 247
bigvoice_custom_writer
sound4.bigvoice.cl.hpp, 236
sound4.bigvoice.cl_dyn.hpp, 248
BIGVOICE_DECL
sound4.bigvoice.cl.h, 228
bigvoice_DeleteClient
Client's JSon config interfacing, 44
bigvoice_ExitProcess
Process creation/destruction, 65
bigvoice_FreeAuxiliaryAudioInfos
Auxiliary outputs, 42
bigvoice_FreeBus
Instance's bus, 62
bigvoice_FreeJsonAnswer
Client's JSon config interfacing, 44
bigvoice_FreeMetadataInfos
Metadata Management, 63
bigvoice_FreeParameters
Process creation/destruction, 66
bigvoice_FreeParameterValue
Process creation/destruction, 66
bigvoice_GetAudio
Audio managing (any audio), 39
bigvoice_GetAudioInputCount
Global setup, 58
bigvoice_GetAudioOutputCount
Global setup, 58
bigvoice_GetAuxiliaryAudioEstimatedDelay
Auxiliary outputs, 43
bigvoice_GetAuxiliaryAudioOutputInfos
Auxiliary outputs, 43
bigvoice_GetBufferIn
Direct processing, 52
bigvoice_GetBufferOut
Direct processing, 52
bigvoice_GetBytesFromFormat
Audio managing (any audio), 40
bigvoice_GetChannelCount
Global setup, 58
bigvoice_GetChunkSizeInFrames
Global setup, 58
bigvoice_GetEstimatedDelay
Direct processing, 53
bigvoice_GetFormatFromName
Audio managing (any audio), 40
bigvoice_GetFormatName
Audio managing (any audio), 40
bigvoice_GetMaxPacketFrame
Audio managing (any audio), 41
bigvoice_GetMetadataInfos
Metadata Management, 64
bigvoice_GetOutputCount
Audio managing (any audio), 41
bigvoice_GetParameter
Process creation/destruction, 66
bigvoice_GetPossibleChunkSizeInFrames
Global setup, 59
bigvoice_GetProcessChunkFrames
Process creation/destruction, 66
bigvoice_GetSampleRate
Global setup, 59
bigvoice_GetVersion
Global setup, 59

BIGVOICE_HAS_CLOUDBUS
sound4.bigvoice.cl.h, 228
BIGVOICE_HAS_WEBSERVER
sound4.bigvoice.cl.h, 228
bigvoice_InitProcess
Process creation/destruction, 67
bigvoice_InitProcess2
Process creation/destruction, 67
bigvoice_InitProcess3
Process creation/destruction, 67
bigvoice_loggerfn
Global setup, 57
bigvoice_LogSeverity
Global setup, 57
bigvoice_MonoToStereo
Audio convert helpers, 32
bigvoice_MonoToStereo_Planar
Audio convert helpers, 32
bigvoice_NewBus
Instance's bus, 62
bigvoice_NewClient
Client's JSon config interfacing, 44
bigvoice_NewParameters
Process creation/destruction, 68
bigvoice_PresetChange_Kind
Custom preset managagement, 50
bigvoice_PresetManager_InformChange
Custom preset managagement, 50
bigvoice_ProcessAudio
Direct processing, 53
bigvoice_ProcessAudio_Planar
Direct processing, 54
bigvoice_ProcessJson
Client's JSon config interfacing, 45
BIGVOICE_SAMPLE_RATE
Global setup, 56
bigvoice_SampleFormat
Audio managing (any audio), 38
bigvoice_SaveState
Client's JSon config interfacing, 45
bigvoice_SetInstanceBus
Instance's bus, 62
bigvoice_SetInstanceTracing
Instance tracing (Windows only), 61
bigvoice_SetInstanceTracingProcessActivity
Instance tracing (Windows only), 61
bigvoice_SetLoggerCallback
Global setup, 60
bigvoice_SetLogSeverity
Global setup, 60
bigvoice_SetMetadata
Metadata Management, 64
bigvoice_SetMetadataMulti
Metadata Management, 64
bigvoice_SetParameter
Process creation/destruction, 69
bigvoice_SetPresetManager
Custom preset managagement, 51
bigvoice_StartUpdateThread
Update Management, 77
bigvoice_StereoToMono
Audio convert helpers, 32
bigvoice_StereoToMono_Planar
Audio convert helpers, 33
bigvoice_StopUpdateThread
Update Management, 77
bigvoice_storage_exists
Custom preset managagement, 47
bigvoice_storage_getall
Custom preset managagement, 47
bigvoice_storage_getall_free
Custom preset managagement, 48
bigvoice_storage_reader
Custom preset managagement, 48
bigvoice_storage_reader_free
Custom preset managagement, 48
bigvoice_storage_remove
Custom preset managagement, 49
bigvoice_storage_rename
Custom preset managagement, 49
bigvoice_storage_writer
Custom preset managagement, 49
bigvoice_TerminateProcess
Process creation/destruction, 69
bigvoice_WaitUpdateThreadReady
Update Management, 77
bigvoice_Webserver
Web server, 79
bigvoice_Webserver_FreeString
Web server, 79
bigvoice_Webserver_GetAppHealth
Web server, 80
bigvoice_Webserver_SetAppHealth
Web server, 80
bigvoice_Webserver_Status
Web server, 80
bigvoice_Webserver_Stop
Web server, 81
bigvoice_Webserver_tcp
Web server, 81
bigvoice_Webserver_tcp2
Web server, 82
buffer
sound4::AuxiliaryOutputInfos, 115
SOUND4_AuxiliaryOutputInfos, 202

CBus
sound4::bigvoice::CBus, 118
sound4::bigvoice::dyn::CBus, 119
sound4::CBus, 120
sound4::CProcessor, 191
CClient
sound4::bigvoice::CInstance::CClient, 121
sound4::bigvoice::dyn::CInstance::CClient, 122
sound4::CInstance::CClient, 124
CDynLib
sound4::bigvoice::dyn::CDynLib, 127

CDynLoader
sound4::bigvoice::dyn::helper::CDynLoader, 138
sound4::helper::CDynLoader, 140
change_kind_created
Custom preset managagement, 50
sound4, 85
change_kind_deleted
Custom preset managagement, 50
sound4, 85
change_kind_modified
Custom preset managagement, 50
sound4, 85
CHANNEL_COUNT
sound4.bigvoice.cl-pipe.cpp, 215
channel_count
sound4::AuxiliaryOutputInfos, 115
SOUND4_AuxiliaryOutputInfos, 202
CHUNK_SIZE
sound4.bigvoice.cl-pipe.cpp, 215
ChunkFrames
sound4::bigvoice, 92
sound4::bigvoice::dyn, 101
CInstance
sound4::bigvoice::CInstance, 144
sound4::bigvoice::dyn::CInstance, 156
sound4::CBus, 120
sound4::CInstance, 169
sound4::CProcessor, 191
Client's JSon config interfacing, 43
bigvoice_DeleteClient, 44
bigvoice_FreeJsonAnswer, 44
bigvoice_NewClient, 44
bigvoice_ProcessJson, 45
bigvoice_SaveState, 45
Close
sound4::bigvoice::dyn::helper::CDynLoader, 138
sound4::helper::CDynLoader, 141
CPresetLoader
sound4::bigvoice::CPresetLoader, 180
sound4::bigvoice::dyn::CPresetLoader, 182
sound4::CPresetLoader, 184
CProcessor
sound4::CProcessor, 187
Create
sound4::bigvoice::CInstance, 144
sound4::bigvoice::dyn::CInstance, 156
sound4::CInstance, 170
Custom preset managagement, 46
bigvoice_PresetChange_Kind, 50
bigvoice_PresetManager_InformChange, 50
bigvoice_SetPresetManager, 51
bigvoice_storage_exists, 47
bigvoice_storage_getall, 47
bigvoice_storage_getall_free, 48
bigvoice_storage_reader, 48
bigvoice_storage_reader_free, 48
bigvoice_storage_remove, 49
bigvoice_storage_rename, 49
bigvoice_storage_writer, 49
change_kind_created, 50
change_kind_deleted, 50
change_kind_modified, 50
fs_char, 50

DeleteClient
sound4::bigvoice::dyn::CDynLib, 128
Deprecated List, 17
description
sound4::AuxiliaryOutputInfos, 116
SOUND4_AuxiliaryOutputInfos, 202
Direct processing, 51
bigvoice_GetBufferIn, 52
bigvoice_GetBufferOut, 52
bigvoice_GetEstimatedDelay, 53
bigvoice_ProcessAudio, 53
bigvoice_ProcessAudio_Planar, 54
doc_examples.h, 205
doc_gencpp.h, 205
doc_json.h, 205
doc_main.h, 206
doc_params.h, 206
documentation.h, 206
DynFuncHolder
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 192
sound4::helper::DynFuncHolder< T >, 194

Environment variables and parameters, 13
error
Global setup, 57
sound4, 84
Examples, 54
Exists
MyPresetManager, 196
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 182
sound4::CPresetLoader, 184
ExitProcess
sound4::bigvoice::dyn::CDynLib, 128

F32_BE
Audio managing (any audio), 38
sound4, 85
F32_LE
Audio managing (any audio), 38
sound4, 85
F32_NATIVE
Audio managing (any audio), 38
sound4, 85
fatal
Global setup, 57
sound4, 84
FreeBus
sound4::bigvoice::dyn::CDynLib, 128
FreeJsonAnswer
sound4::bigvoice::dyn::CDynLib, 129
FreeMetadataInfos

sound4::bigvoice::dyn::CDynLib, 129
FreeParameters
sound4::bigvoice::dyn::CDynLib, 129
FreeParameterValue
sound4::bigvoice::dyn::CDynLib, 129
fs_char
Custom preset managagement, 50
sound4cl_cdef.h, 278

Generic C++ access to SOUND4 .CL librairies, 3
Get
sound4::bigvoice::CBus, 118
sound4::bigvoice::dyn::CBus, 119
sound4::CBus, 120
GetAll
MyPresetManager, 196
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 182
sound4::CPresetLoader, 184
GetAudio
sound4::bigvoice::CInstance, 145
sound4::bigvoice::dyn::CDynLib, 129
sound4::bigvoice::dyn::CInstance, 157
sound4::CInstance, 171
GetAudioInputCount
sound4::bigvoice, 88
sound4::bigvoice::dyn, 95
sound4::bigvoice::dyn::CDynLib, 129
sound4::CProcessor, 187
GetAudioOutputCount
sound4::bigvoice, 88
sound4::bigvoice::dyn, 96
sound4::bigvoice::dyn::CDynLib, 129
sound4::CProcessor, 187
GetAuxiliaryAudioOutputInfos
sound4::bigvoice::CInstance, 145
GetBufferIn
sound4::bigvoice::CInstance, 146
sound4::bigvoice::dyn::CDynLib, 130
sound4::bigvoice::dyn::CInstance, 157
sound4::CInstance, 171
GetBufferOut
sound4::bigvoice::CInstance, 146
sound4::bigvoice::dyn::CDynLib, 130
sound4::bigvoice::dyn::CInstance, 158
sound4::CInstance, 171
GetBytesFromFormat
sound4::bigvoice, 88
sound4::bigvoice::dyn, 96
sound4::bigvoice::dyn::CDynLib, 130
sound4::CProcessor, 187
GetChannelCount
sound4::bigvoice, 89
sound4::bigvoice::dyn, 96
sound4::bigvoice::dyn::CDynLib, 130
sound4::CProcessor, 188
GetChunkFrames
sound4::bigvoice::CInstance, 146
sound4::bigvoice::dyn::CInstance, 158
sound4::CInstance, 171
GetChunkSizeInFrames
sound4::bigvoice, 89
sound4::bigvoice::dyn, 97
sound4::bigvoice::dyn::CDynLib, 130
sound4::CProcessor, 188
GetEnv
sound4.bigvoice.cl-pipe-cxx.cpp, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 210
sound4cl-pipe.cpp, 220
GetEstimatedDelay
sound4::bigvoice::CInstance, 147
sound4::bigvoice::dyn::CDynLib, 130
sound4::bigvoice::dyn::CInstance, 158
sound4::CInstance, 172
GetFormatFromName
sound4::bigvoice, 89
sound4::bigvoice::dyn, 97
sound4::bigvoice::dyn::CDynLib, 130
sound4::CProcessor, 188
GetFormatName
sound4::bigvoice, 90
sound4::bigvoice::dyn, 98
sound4::bigvoice::dyn::CDynLib, 131
sound4::CProcessor, 189
GetMaxPacketFrame
sound4::bigvoice::dyn::CDynLib, 131
GetMetadataInfos
sound4::bigvoice::CInstance, 147
sound4::bigvoice::dyn::CDynLib, 131
sound4::bigvoice::dyn::CInstance, 159
sound4::CInstance, 172
GetOutputCount
sound4::bigvoice::dyn::CDynLib, 131
GetParam
sound4::bigvoice::CInstance, 147
sound4::bigvoice::dyn::CInstance, 159
sound4::CInstance, 172
GetParameter
sound4::bigvoice::dyn::CDynLib, 131
GetPossibleChunkSizeInFrames
sound4::bigvoice, 90
sound4::bigvoice::dyn, 98
sound4::bigvoice::dyn::CDynLib, 131
sound4::CProcessor, 189
GetPrefixSymbol
sound4::CProcessor, 189
GetProcessChunkFrames
sound4::bigvoice::dyn::CDynLib, 131
GetSampleRate
sound4::bigvoice, 90
sound4::bigvoice::dyn, 99
sound4::bigvoice::dyn::CDynLib, 132
sound4::CProcessor, 190
GetSymbol
sound4::bigvoice::dyn::helper::CDynLoader, 138
sound4::helper::CDynLoader, 141
GetSymbol_safe

sound4::bigvoice::dyn::helper::CDynLoader, 138
sound4::helper::CDynLoader, 141
GetThisLibraryPath
sound4::bigvoice::dyn::helper::CDynLoader, 139
sound4::helper::CDynLoader, 141
GetVersion
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99
sound4::bigvoice::dyn::CDynLib, 132
sound4::CProcessor, 190
GetWebServerAppHealth
sound4::bigvoice::CInstance, 148
sound4::bigvoice::dyn::CInstance, 159
sound4::CInstance, 173
Global setup, 55
BIGVOICE_AUDIO_INPUT_COUNT, 56
BIGVOICE_AUDIO_OUTPUT_COUNT, 56
BIGVOICE_AUDIOFRAME_COUNT, 56
BIGVOICE_CHANNEL_COUNT, 56
bigvoice_GetAudioInputCount, 58
bigvoice_GetAudioOutputCount, 58
bigvoice_GetChannelCount, 58
bigvoice_GetChunkSizeInFrames, 58
bigvoice_GetPossibleChunkSizeInFrames, 59
bigvoice_GetSampleRate, 59
bigvoice_GetVersion, 59
bigvoice_loggerfn, 57
bigvoice_LogSeverity, 57
BIGVOICE_SAMPLE_RATE, 56
bigvoice_SetLoggerCallback, 60
bigvoice_SetLogSeverity, 60
error, 57
fatal, 57
info, 57
none, 57
SOUND4_GetProcessInfo, 60
verbose, 57
verbose2, 57
verbose3, 57
verbose4, 57
verbose5, 57
warning, 57

id
sound4::AuxiliaryOutputInfos, 116
SOUND4_AuxiliaryOutputInfos, 202
info
Global setup, 57
sound4, 84
InitProcess
sound4::bigvoice::dyn::CDynLib, 132
InitProcess2
sound4::bigvoice::dyn::CDynLib, 132
InitProcess3
sound4::bigvoice::dyn::CDynLib, 132
InputChannels
sound4::bigvoice, 92
sound4::bigvoice::dyn, 101
InputSampleSize
sound4::bigvoice, 92
sound4::bigvoice::dyn, 102
Instance tracing (Windows only), 61
bigvoice_SetInstanceTracing, 61
bigvoice_SetInstanceTracingProcessActivity, 61
Instance's bus, 62
bigvoice_FreeBus, 62
bigvoice_NewBus, 62
bigvoice_SetInstanceBus, 62
INVALID_FORMAT
Audio managing (any audio), 38
sound4, 85
IsOk
sound4::bigvoice::CInstance, 148
sound4::bigvoice::dyn::CDynLib, 127
sound4::bigvoice::dyn::CInstance, 160
sound4::bigvoice::dyn::helper::CDynLoader, 139
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 192
sound4::CInstance, 173
sound4::CProcessor, 190
sound4::helper::CDynLoader, 141
sound4::helper::DynFuncHolder< T >, 194
IsReadOnly
MyPresetManager, 197
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 183
sound4::CPresetLoader, 185

JSON syntax, 5
JSON syntax for internal setup, 9

Load
sound4::bigvoice::dyn::CDynLib, 127
sound4::bigvoice::dyn::helper::CDynLoader, 139
sound4::CProcessor, 190
sound4::helper::CDynLoader, 141
log_cb_t
sound4, 84
sound4::bigvoice, 87
sound4::bigvoice::dyn, 95
LogSeverity
sound4, 84
sound4::bigvoice, 87
sound4::bigvoice::dyn, 95

main
sound4.bigvoice.cl-pipe-cxx.cpp, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 210
sound4.bigvoice.cl-pipe.cpp, 216
sound4cl-pipe.cpp, 220
Metadata Management, 63
bigvoice_FreeMetadataInfos, 63
bigvoice_GetMetadataInfos, 64
bigvoice_SetMetadata, 64
bigvoice_SetMetadataMulti, 64
MonoToStereo
Audio convert helpers (C++), 35
sound4::bigvoice::dyn::CDynLib, 132

sound4::bigvoice::dyn::helper, 105
sound4::bigvoice::helper, 110
MonoToStereo_Planar
Audio convert helpers (C++), 36
sound4::bigvoice::dyn::CDynLib, 132
sound4::bigvoice::dyn::helper, 106
sound4::bigvoice::helper, 111
MyLogger
sound4.bigvoice.cl-pipe-cxx.cpp, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 211
sound4.bigvoice.cl-pipe.cpp, 216
MyPresetManager, 195
~MyPresetManager, 196
Exists, 196
GetAll, 196
IsReadOnly, 197
MyPresetManager, 196
Read, 197
Remove, 197
Rename, 198
SetupPath, 198
Write, 198

name
SOUND4_ProcessInfo, 203
NewBus
sound4::bigvoice::dyn::CDynLib, 133
NewClient
sound4::bigvoice::CInstance, 148
sound4::bigvoice::dyn::CDynLib, 133
sound4::bigvoice::dyn::CInstance, 160
sound4::CInstance, 173
NewParameters
sound4::bigvoice::dyn::CDynLib, 133
none
Global setup, 57
sound4, 84

OnUpdateThreadStart
sound4::bigvoice::CInstance, 148
sound4::bigvoice::dyn::CInstance, 160
sound4::CInstance, 173
OnUpdateThreadStop
sound4::bigvoice::CInstance, 148
sound4::bigvoice::dyn::CInstance, 160
sound4::CInstance, 173
operator bool
sound4::bigvoice::dyn::helper::CDynLoader, 139
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 192
sound4::helper::CDynLoader, 142
sound4::helper::DynFuncHolder< T >, 194
operator T*
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 193
sound4::helper::DynFuncHolder< T >, 194
operator=
sound4::bigvoice::CInstance, 149
sound4::bigvoice::CInstance::CClient, 121
sound4::bigvoice::dyn::CInstance, 160
sound4::bigvoice::dyn::CInstance::CClient, 123
sound4::bigvoice::dyn::helper::CDynLoader, 139
sound4::CInstance, 174
sound4::CInstance::CClient, 124
sound4::helper::CDynLoader, 142
OutputChannels
sound4::bigvoice, 92
sound4::bigvoice::dyn, 102
OutputSampleSize
sound4::bigvoice, 93
sound4::bigvoice::dyn, 102

prefix
SOUND4_ProcessInfo, 203
PresetChange_Kind
sound4, 84
sound4::bigvoice, 87
sound4::bigvoice::dyn, 95
PresetManager_InformChange
sound4::bigvoice::CInstance, 149
sound4::bigvoice::dyn::CDynLib, 133
sound4::bigvoice::dyn::CInstance, 160
sound4::CInstance, 174
Process creation/destruction, 65
bigvoice_ExitProcess, 65
bigvoice_FreeParameters, 66
bigvoice_FreeParameterValue, 66
bigvoice_GetParameter, 66
bigvoice_GetProcessChunkFrames, 66
bigvoice_InitProcess, 67
bigvoice_InitProcess2, 67
bigvoice_InitProcess3, 67
bigvoice_NewParameters, 68
bigvoice_SetParameter, 69
bigvoice_TerminateProcess, 69
process_name
sound4::bigvoice, 93
sound4::bigvoice::dyn, 102
process_shortname
sound4::bigvoice, 93
sound4::bigvoice::dyn, 102
ProcessAnyAudio
sound4::bigvoice::CInstance, 149
sound4::bigvoice::dyn::CInstance, 161
sound4::CInstance, 174
ProcessAudio
sound4::bigvoice::CInstance, 150
sound4::bigvoice::dyn::CDynLib, 133
sound4::bigvoice::dyn::CInstance, 161
sound4::CInstance, 175
ProcessAudio_Planar
sound4::bigvoice::CInstance, 150
sound4::bigvoice::dyn::CDynLib, 133
sound4::bigvoice::dyn::CInstance, 161
sound4::CInstance, 175
ProcessJson
sound4::bigvoice::CInstance::CClient, 121
sound4::bigvoice::dyn::CDynLib, 133

sound4::bigvoice::dyn::CInstance::CClient, 123
sound4::CInstance::CClient, 124
ProcessLoop
sound4.bigvoice.cl-pipe-cxx.cpp, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 211
sound4.bigvoice.cl-pipe.cpp, 216
sound4cl-pipe.cpp, 221

Read
MyPresetManager, 197
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 183
sound4::CPresetLoader, 185
Remove
MyPresetManager, 197
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 183
sound4::CPresetLoader, 185
Rename
MyPresetManager, 198
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 183
sound4::CPresetLoader, 185

S16_BE
Audio managing (any audio), 38
sound4, 85
S16_LE
Audio managing (any audio), 38
sound4, 85
S16_NATIVE
Audio managing (any audio), 38
sound4, 85
S24_BE
Audio managing (any audio), 38
sound4, 85
S24_LE
Audio managing (any audio), 38
sound4, 85
S24_NATIVE
Audio managing (any audio), 38
sound4, 85
S32_BE
Audio managing (any audio), 38
sound4, 85
S32_LE
Audio managing (any audio), 38
sound4, 85
S32_NATIVE
Audio managing (any audio), 38
sound4, 85
Sample using C interface, 69
Sample using C++ interface wrapper, 71
Sample using C++ interface wrapper with dynamic library loading, 72
Sample using Generic C++ interface (multi-process compatible), 74
SAMPLE_RATE
sound4.bigvoice.cl-pipe.cpp, 215

sample_rate
sound4::AuxiliaryOutputInfos, 116
SOUND4_AuxiliaryOutputInfos, 202
SampleFormat
sound4, 85
sound4::bigvoice, 87
sound4::bigvoice::dyn, 95
SampleRate
sound4::bigvoice, 93
sound4::bigvoice::dyn, 103
SanityCheck
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99
SaveState
sound4::bigvoice::CInstance, 150
sound4::bigvoice::dyn::CDynLib, 134
sound4::bigvoice::dyn::CInstance, 163
sound4::CInstance, 175
SetBus
sound4::bigvoice::CInstance, 151
sound4::bigvoice::dyn::CInstance, 163
sound4::CInstance, 176
SetInstanceBus
sound4::bigvoice::dyn::CDynLib, 134
SetInstanceTracing
sound4::bigvoice::CInstance, 151
sound4::bigvoice::dyn::CDynLib, 134
sound4::bigvoice::dyn::CInstance, 163
sound4::CInstance, 176
SetInstanceTracingProcessActivity
sound4::bigvoice::CInstance, 151
sound4::bigvoice::dyn::CDynLib, 134
sound4::bigvoice::dyn::CInstance, 164
sound4::CInstance, 176
SetLoggerCallback
sound4::bigvoice, 91
sound4::bigvoice::dyn, 101
sound4::bigvoice::dyn::CDynLib, 134
sound4::CProcessor, 191
SetLogSeverity
sound4::bigvoice, 91
sound4::bigvoice::dyn, 101
sound4::bigvoice::dyn::CDynLib, 134
sound4::CProcessor, 191
SetMetadata
sound4::bigvoice::CInstance, 152
sound4::bigvoice::dyn::CDynLib, 134
sound4::bigvoice::dyn::CInstance, 164
sound4::CInstance, 177
SetMetadataMulti
sound4::bigvoice::CInstance, 152
sound4::bigvoice::dyn::CDynLib, 135
sound4::bigvoice::dyn::CInstance, 164
sound4::CInstance, 177
SetParam
sound4::bigvoice::CInstance, 152
sound4::bigvoice::dyn::CInstance, 165
sound4::CInstance, 177

SetParameter
sound4::bigvoice::dyn::CDynLib, 135
SetPresetManager
sound4::bigvoice::CInstance, 153
sound4::bigvoice::dyn::CDynLib, 135
sound4::bigvoice::dyn::CInstance, 165
sound4::CInstance, 178
SetupPath
MyPresetManager, 198
SetWebServerAppHealth
sound4::bigvoice::CInstance, 153
sound4::bigvoice::dyn::CInstance, 165
sound4::CInstance, 178
shortname
SOUND4_ProcessInfo, 203
sound4, 83
_log_cb, 86
_log_cb_c, 85
change_kind_created, 85
change_kind_deleted, 85
change_kind_modified, 85
error, 84
F32_BE, 85
F32_LE, 85
F32_NATIVE, 85
fatal, 84
info, 84
INVALID_FORMAT, 85
log_cb_t, 84
LogSeverity, 84
none, 84
PresetChange_Kind, 84
S16_BE, 85
S16_LE, 85
S16_NATIVE, 85
S24_BE, 85
S24_LE, 85
S24_NATIVE, 85
S32_BE, 85
S32_LE, 85
S32_NATIVE, 85
SampleFormat, 85
verbose, 84
verbose2, 84
verbose3, 84
verbose4, 84
verbose5, 84
warning, 84
SOUND4 BIGVOICE.CL Library, 1
sound4.bigvoice.cl-pipe-cxx.cpp, 206, 208
GetEnv, 207
main, 207
MyLogger, 207
ProcessLoop, 207
Usage, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 210, 212
GetEnv, 210
main, 210
MyLogger, 211
ProcessLoop, 211
Usage, 211
sound4.bigvoice.cl-pipe.cpp, 215, 217
CHANNEL_COUNT, 215
CHUNK_SIZE, 215
main, 216
MyLogger, 216
ProcessLoop, 216
SAMPLE_RATE, 215
Usage, 216
sound4.bigvoice.cl.h, 225
BIGVOICE_DECL, 228
BIGVOICE_HAS_CLOUDBUS, 228
BIGVOICE_HAS_WEBSERVER, 228
TraceLoggingHProvider, 228
sound4.bigvoice.cl.hpp, 233
bigvoice_custom_exists, 235
bigvoice_custom_getall, 235
bigvoice_custom_getall_free, 235
bigvoice_custom_reader, 235
bigvoice_custom_reader_free, 235
bigvoice_custom_remove, 235
bigvoice_custom_rename, 235
bigvoice_custom_writer, 236
sound4.bigvoice.cl_dyn.hpp, 244
bigvoice_custom_exists, 247
bigvoice_custom_getall, 247
bigvoice_custom_getall_free, 247
bigvoice_custom_reader, 247
bigvoice_custom_reader_free, 247
bigvoice_custom_remove, 247
bigvoice_custom_rename, 247
bigvoice_custom_writer, 248
UNIXLIKE, 246
sound4::AuxiliaryOutputInfos, 115
buffer, 115
channel_count, 115
description, 116
id, 116
sample_rate, 116
sound4::bigvoice, 86
_log_cb, 92
_log_cb_c, 88
ChunkFrames, 92
GetAudioInputCount, 88
GetAudioOutputCount, 88
GetBytesFromFormat, 88
GetChannelCount, 89
GetChunkSizeInFrames, 89
GetFormatFromName, 89
GetFormatName, 90
GetPossibleChunkSizeInFrames, 90
GetSampleRate, 90
GetVersion, 91
InputChannels, 92
InputSampleSize, 92
log_cb_t, 87

LogSeverity, 87
OutputChannels, 92
OutputSampleSize, 93
PresetChange_Kind, 87
process_name, 93
process_shortname, 93
SampleFormat, 87
SampleRate, 93
SanityCheck, 91
SetLoggerCallback, 91
SetLogSeverity, 91
sound4::bigvoice::CBus, 117
~CBus, 118
CBus, 118
Get, 118
sound4::bigvoice::CInstance, 142
~CInstance, 144
ActivateAuxiliaryAudioID, 144
AddAudio, 144
CInstance, 144
Create, 144
GetAudio, 145
GetAuxiliaryAudioOutputInfos, 145
GetBufferIn, 146
GetBufferOut, 146
GetChunkFrames, 146
GetEstimatedDelay, 147
GetMetadataInfos, 147
GetParam, 147
GetWebServerAppHealth, 148
IsOk, 148
NewClient, 148
OnUpdateThreadStart, 148
OnUpdateThreadStop, 148
operator=, 149
PresetManager_InformChange, 149
ProcessAnyAudio, 149
ProcessAudio, 150
ProcessAudio_Planar, 150
SaveState, 150
SetBus, 151
SetInstanceTracing, 151
SetInstanceTracingProcessActivity, 151
SetMetadata, 152
SetMetadataMulti, 152
SetParam, 152
SetPresetManager, 153
SetWebServerAppHealth, 153
StartWebServer, 153
Stop, 154
StopWebServer, 154
Terminate, 154
sound4::bigvoice::CInstance::CClient, 121
~CClient, 121
CClient, 121
operator=, 121
ProcessJson, 121
sound4::bigvoice::CPresetLoader, 180
~CPresetLoader, 180
CPresetLoader, 180
Exists, 181
GetAll, 181
IsReadOnly, 181
Read, 181
Remove, 181
Rename, 181
Write, 181
sound4::bigvoice::dyn, 94
_log_cb, 101
_log_cb_c, 95
ChunkFrames, 101
GetAudioInputCount, 95
GetAudioOutputCount, 96
GetBytesFromFormat, 96
GetChannelCount, 96
GetChunkSizeInFrames, 97
GetFormatFromName, 97
GetFormatName, 98
GetPossibleChunkSizeInFrames, 98
GetSampleRate, 99
GetVersion, 99
InputChannels, 101
InputSampleSize, 102
log_cb_t, 95
LogSeverity, 95
OutputChannels, 102
OutputSampleSize, 102
PresetChange_Kind, 95
process_name, 102
process_shortname, 102
SampleFormat, 95
SampleRate, 103
SanityCheck, 99
SetLoggerCallback, 101
SetLogSeverity, 101
sound4::bigvoice::dyn::CBus, 118
~CBus, 119
CBus, 119
Get, 119
sound4::bigvoice::dyn::CDynLib, 125
AddAudio, 127
AddPadAudio, 127
AudioConvertFrom, 128
AudioConvertTo, 128
AudioMonoFromLiveStereo, 128
AudioMonoToLiveStereo, 128
CDynLib, 127
DeleteClient, 128
ExitProcess, 128
FreeBus, 128
FreeJsonAnswer, 129
FreeMetadataInfos, 129
FreeParameters, 129
FreeParameterValue, 129
GetAudio, 129
GetAudioInputCount, 129

GetAudioOutputCount, 129
GetBufferIn, 130
GetBufferOut, 130
GetBytesFromFormat, 130
GetChannelCount, 130
GetChunkSizeInFrames, 130
GetEstimatedDelay, 130
GetFormatFromName, 130
GetFormatName, 131
GetMaxPacketFrame, 131
GetMetadataInfos, 131
GetOutputCount, 131
GetParameter, 131
GetPossibleChunkSizeInFrames, 131
GetProcessChunkFrames, 131
GetSampleRate, 132
GetVersion, 132
InitProcess, 132
InitProcess2, 132
InitProcess3, 132
IsOk, 127
Load, 127
MonoToStereo, 132
MonoToStereo_Planar, 132
NewBus, 133
NewClient, 133
NewParameters, 133
PresetManager_InformChange, 133
ProcessAudio, 133
ProcessAudio_Planar, 133
ProcessJson, 133
SaveState, 134
SetInstanceBus, 134
SetInstanceTracing, 134
SetInstanceTracingProcessActivity, 134
SetLoggerCallback, 134
SetLogSeverity, 134
SetMetadata, 134
SetMetadataMulti, 135
SetParameter, 135
SetPresetManager, 135
StartUpdateThread, 135
StereoToMono, 135
StereoToMono_Planar, 135
StopUpdateThread, 135
TerminateProcess, 136
WaitUpdateThreadReady, 136
Webserver, 136
Webserver_FreeString, 136
Webserver_GetAppHealth, 136
Webserver_SetAppHealth, 136
Webserver_Status, 136
Webserver_Stop, 137
Webserver_tcp, 137
Webserver_tcp2, 137
sound4::bigvoice::dyn::CInstance, 155
~CInstance, 156
AddAudio, 156
CInstance, 156
Create, 156
GetAudio, 157
GetBufferIn, 157
GetBufferOut, 158
GetChunkFrames, 158
GetEstimatedDelay, 158
GetMetadataInfos, 159
GetParam, 159
GetWebServerAppHealth, 159
IsOk, 160
NewClient, 160
OnUpdateThreadStart, 160
OnUpdateThreadStop, 160
operator=, 160
PresetManager_InformChange, 160
ProcessAnyAudio, 161
ProcessAudio, 161
ProcessAudio_Planar, 161
SaveState, 163
SetBus, 163
SetInstanceTracing, 163
SetInstanceTracingProcessActivity, 164
SetMetadata, 164
SetMetadataMulti, 164
SetParam, 165
SetPresetManager, 165
SetWebServerAppHealth, 165
StartWebServer, 166
Stop, 166
StopWebServer, 166
Terminate, 167
sound4::bigvoice::dyn::CInstance::CClient, 122
~CClient, 123
CClient, 122
operator=, 123
ProcessJson, 123
sound4::bigvoice::dyn::CPresetLoader, 182
~CPresetLoader, 182
CPresetLoader, 182
Exists, 182
GetAll, 182
IsReadOnly, 183
Read, 183
Remove, 183
Rename, 183
Write, 183
sound4::bigvoice::dyn::helper, 103
AudioConvertFrom, 104
AudioConvertTo, 104
AudioMonoFromLiveStereo, 105
AudioMonoToLiveStereo, 105
MonoToStereo, 105
MonoToStereo_Planar, 106
StereoToMono, 106
StereoToMono_Planar, 107
UTF8ToWString, 107
WStringToUTF8, 108

sound4::bigvoice::dyn::helper::CDynLoader, 137
~CDynLoader, 138
CDynLoader, 138
Close, 138
GetSymbol, 138
GetSymbol_safe, 138
GetThisLibraryPath, 139
IsOk, 139
Load, 139
operator bool, 139
operator=, 139
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 192
DynFuncHolder, 192
IsOk, 192
operator bool, 192
operator T*, 193
sound4::bigvoice::dyn::helper::SampleFormat< T >, 199
sound4::bigvoice::helper, 108
AudioConvertFrom, 109
AudioConvertTo, 109
AudioMonoFromLiveStereo, 109
AudioMonoToLiveStereo, 110
MonoToStereo, 110
MonoToStereo_Planar, 111
StereoToMono, 111
StereoToMono_Planar, 111
UTF8ToWString, 112
WStringToUTF8, 112
sound4::bigvoice::helper::SampleFormat< T >, 199
sound4::CBus, 119
~CBus, 120
CBus, 120
CInstance, 120
Get, 120
sound4::CInstance, 167
~CInstance, 169
AddAudio, 170
CInstance, 169
Create, 170
GetAudio, 171
GetBufferIn, 171
GetBufferOut, 171
GetChunkFrames, 171
GetEstimatedDelay, 172
GetMetadataInfos, 172
GetParam, 172
GetWebServerAppHealth, 173
IsOk, 173
NewClient, 173
OnUpdateThreadStart, 173
OnUpdateThreadStop, 173
operator=, 174
PresetManager_InformChange, 174
ProcessAnyAudio, 174
ProcessAudio, 175
ProcessAudio_Planar, 175
SaveState, 175
SetBus, 176
SetInstanceTracing, 176
SetInstanceTracingProcessActivity, 176
SetMetadata, 177
SetMetadataMulti, 177
SetParam, 177
SetPresetManager, 178
SetWebServerAppHealth, 178
StartWebServer, 178
Stop, 179
StopWebServer, 179
Terminate, 179
sound4::CInstance::CClient, 123
~CClient, 124
CClient, 124
operator=, 124
ProcessJson, 124
sound4::CPresetLoader, 184
~CPresetLoader, 184
CPresetLoader, 184
Exists, 184
GetAll, 184
IsReadOnly, 185
Read, 185
Remove, 185
Rename, 185
Write, 185
sound4::CProcessor, 186
CBus, 191
CInstance, 191
CProcessor, 187
GetAudioInputCount, 187
GetAudioOutputCount, 187
GetBytesFromFormat, 187
GetChannelCount, 188
GetChunkSizeInFrames, 188
GetFormatFromName, 188
GetFormatName, 189
GetPossibleChunkSizeInFrames, 189
GetPrefixSymbol, 189
GetSampleRate, 190
GetVersion, 190
IsOk, 190
Load, 190
SetLoggerCallback, 191
SetLogSeverity, 191
sound4::helper, 113
UTF8ToWString, 113
WStringToUTF8, 113
sound4::helper::CDynLoader, 140
~CDynLoader, 140
CDynLoader, 140
Close, 141
GetSymbol, 141
GetSymbol_safe, 141
GetThisLibraryPath, 141
IsOk, 141

Load, 141
operator bool, 142
operator=, 142
sound4::helper::DynFuncHolder< T >, 193
DynFuncHolder, 194
IsOk, 194
operator bool, 194
operator T*, 194
sound4::helper::SampleFormat< T >, 201
SOUND4_AuxiliaryOutputInfos, 201
buffer, 202
channel_count, 202
description, 202
id, 202
sample_rate, 202
SOUND4_GetProcessInfo
Global setup, 60
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 278
Web server, 78
SOUND4_ProcessInfo, 202
name, 203
prefix, 203
shortname, 203
struct_size, 203
version, 203
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 278
Web server, 78
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 278
Web server, 79
sound4cl-pipe.cpp, 220, 222
GetEnv, 220
main, 220
ProcessLoop, 221
sound4cl.hpp, 262
UNIXLIKE, 263
sound4cl_cdef.h, 277
fs_char, 278
SOUND4_INVALID_WEBSERVER_ID, 278
SOUND4_WEBSERVER_HTTP_OK, 278
SOUND4_WEBSERVER_HTTPS_OK, 278
TraceLoggingHProvider, 278
StartUpdateThread
sound4::bigvoice::dyn::CDynLib, 135
StartWebServer
sound4::bigvoice::CInstance, 153
sound4::bigvoice::dyn::CInstance, 166
sound4::CInstance, 178
StereoToMono
Audio convert helpers (C++), 36
sound4::bigvoice::dyn::CDynLib, 135
sound4::bigvoice::dyn::helper, 106
sound4::bigvoice::helper, 111
StereoToMono_Planar
Audio convert helpers (C++), 37
sound4::bigvoice::dyn::CDynLib, 135
sound4::bigvoice::dyn::helper, 107
sound4::bigvoice::helper, 111
Stop
sound4::bigvoice::CInstance, 154
sound4::bigvoice::dyn::CInstance, 166
sound4::CInstance, 179
StopUpdateThread
sound4::bigvoice::dyn::CDynLib, 135
StopWebServer
sound4::bigvoice::CInstance, 154
sound4::bigvoice::dyn::CInstance, 166
sound4::CInstance, 179
struct_size
SOUND4_ProcessInfo, 203

Terminate
sound4::bigvoice::CInstance, 154
sound4::bigvoice::dyn::CInstance, 167
sound4::CInstance, 179
TerminateProcess
sound4::bigvoice::dyn::CDynLib, 136
TraceLoggingHProvider
sound4.bigvoice.cl.h, 228
sound4cl_cdef.h, 278

UNIXLIKE
sound4.bigvoice.cl_dyn.hpp, 246
sound4cl.hpp, 263
Update Management, 76
bigvoice_StartUpdateThread, 77
bigvoice_StopUpdateThread, 77
bigvoice_WaitUpdateThreadReady, 77
Usage
sound4.bigvoice.cl-pipe-cxx.cpp, 207
sound4.bigvoice.cl-pipe-dyn.cpp, 211
sound4.bigvoice.cl-pipe.cpp, 216
UTF8ToWString
sound4::bigvoice::dyn::helper, 107
sound4::bigvoice::helper, 112
sound4::helper, 113

verbose
Global setup, 57
sound4, 84
verbose2
Global setup, 57
sound4, 84
verbose3
Global setup, 57
sound4, 84
verbose4
Global setup, 57
sound4, 84
verbose5
Global setup, 57
sound4, 84
version
SOUND4_ProcessInfo, 203

WaitUpdateThreadReady
sound4::bigvoice::dyn::CDynLib, 136
warning
Global setup, 57
sound4, 84
Web server, 78
bigvoice_Webserver, 79
bigvoice_Webserver_FreeString, 79
bigvoice_Webserver_GetAppHealth, 80
bigvoice_Webserver_SetAppHealth, 80
bigvoice_Webserver_Status, 80
bigvoice_Webserver_Stop, 81
bigvoice_Webserver_tcp, 81
bigvoice_Webserver_tcp2, 82
SOUND4_INVALID_WEBSERVER_ID, 78
SOUND4_WEBSERVER_HTTP_OK, 78
SOUND4_WEBSERVER_HTTPS_OK, 79
Webserver
sound4::bigvoice::dyn::CDynLib, 136
Webserver_FreeString
sound4::bigvoice::dyn::CDynLib, 136
Webserver_GetAppHealth
sound4::bigvoice::dyn::CDynLib, 136
Webserver_SetAppHealth
sound4::bigvoice::dyn::CDynLib, 136
Webserver_Status
sound4::bigvoice::dyn::CDynLib, 136
Webserver_Stop
sound4::bigvoice::dyn::CDynLib, 137
Webserver_tcp
sound4::bigvoice::dyn::CDynLib, 137
Webserver_tcp2
sound4::bigvoice::dyn::CDynLib, 137
Write
MyPresetManager, 198
sound4::bigvoice::CPresetLoader, 181
sound4::bigvoice::dyn::CPresetLoader, 183
sound4::CPresetLoader, 185
WStringToUTF8
sound4::bigvoice::dyn::helper, 108
sound4::bigvoice::helper, 112
sound4::helper, 113