# SOUND4 BIGVOICE.CL Library [1.1.13]

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 . . . 30
12.1.2 Enumeration Type Documentation . . . 30
12.1.2.1 bigvoice_ConverterTargetMode . . . 30
12.1.3 Function Documentation . . . 30
12.1.3.1 bigvoice_AudioConverter_From() . . . 30

12.1.3.2 bigvoice_AudioConverter_To() . . . 30
12.1.3.3 bigvoice_AudioConvertFrom() . . . 31
12.1.3.4 bigvoice_AudioConvertTo() . . . 31
12.1.3.5 bigvoice_AudioMonoFromLiveStereo() . . . 32
12.1.3.6 bigvoice_AudioMonoToLiveStereo() . . . 32
12.1.3.7 bigvoice_FreeAudioConverter() . . . 32
12.1.3.8 bigvoice_GetAudioConverter() . . . 32
12.1.3.9 bigvoice_MonoToStereo() . . . 33
12.1.3.10 bigvoice_MonoToStereo_Planar() . . . 34
12.1.3.11 bigvoice_StereoToMono() . . . 34
12.1.3.12 bigvoice_StereoToMono_Planar() . . . 34
12.2 Audio convert helpers (C++) . . . 35
12.2.1 Detailed Description . . . 35
12.2.2 Enumeration Type Documentation . . . 36
12.2.2.1 ConverterTargetMode . . . 36
12.2.3 Function Documentation . . . 36
12.2.3.1 AudioConvertFrom() . . . 36
12.2.3.2 AudioConvertTo() . . . 36
12.2.3.3 AudioMonoFromLiveStereo() . . . 37
12.2.3.4 AudioMonoToLiveStereo() . . . 37
12.2.3.5 MonoToStereo() . . . 38
12.2.3.6 MonoToStereo_Planar() . . . 38
12.2.3.7 StereoToMono() . . . 38
12.2.3.8 StereoToMono_Planar() . . . 39
12.3 Audio managing (any audio) . . . 39
12.3.1 Detailed Description . . . 40
12.3.2 Enumeration Type Documentation . . . 40
12.3.2.1 bigvoice_SampleFormat . . . 40
12.3.3 Function Documentation . . . 41
12.3.3.1 bigvoice_AddAudio() . . . 41
12.3.3.2 bigvoice_AddPadAudio() . . . 41
12.3.3.3 bigvoice_GetAudio() . . . 41
12.3.3.4 bigvoice_GetBytesFromFormat() . . . 43
12.3.3.5 bigvoice_GetFormatFromName() . . . 43
12.3.3.6 bigvoice_GetFormatName() . . . 43
12.3.3.7 bigvoice_GetMaxPacketFrame() . . . 44
12.3.3.8 bigvoice_GetOutputCount() . . . 44
12.4 Auxiliary outputs . . . 44
12.4.1 Detailed Description . . . 45
12.4.2 Function Documentation . . . 45
12.4.2.1 bigvoice_ActivateAuxiliaryAudioID() . . . 45
12.4.2.2 bigvoice_FreeAuxiliaryAudioInfos() . . . 45

12.4.2.3 bigvoice_GetAuxiliaryAudioEstimatedDelay() . . . 46
12.4.2.4 bigvoice_GetAuxiliaryAudioOutputInfos() . . . 46
12.5 Client's JSon config interfacing . . . 46
12.5.1 Detailed Description . . . 47
12.5.2 Function Documentation . . . 47
12.5.2.1 bigvoice_DeleteClient() . . . 47
12.5.2.2 bigvoice_FreeJsonAnswer() . . . 47
12.5.2.3 bigvoice_NewClient() . . . 47
12.5.2.4 bigvoice_ProcessJson() . . . 48
12.5.2.5 bigvoice_SaveState() . . . 48
12.6 Custom preset managagement . . . 49
12.6.1 Detailed Description . . . 50
12.6.2 Typedef Documentation . . . 50
12.6.2.1 bigvoice_storage_exists . . . 50
12.6.2.2 bigvoice_storage_getall . . . 50
12.6.2.3 bigvoice_storage_getall_free . . . 51
12.6.2.4 bigvoice_storage_reader . . . 51
12.6.2.5 bigvoice_storage_reader_free . . . 51
12.6.2.6 bigvoice_storage_remove . . . 52
12.6.2.7 bigvoice_storage_rename . . . 52
12.6.2.8 bigvoice_storage_writer . . . 52
12.6.2.9 fs_char . . . 53
12.6.3 Enumeration Type Documentation . . . 53
12.6.3.1 bigvoice_PresetChange_Kind . . . 53
12.6.4 Function Documentation . . . 53
12.6.4.1 bigvoice_PresetManager_InformChange() . . . 53
12.6.4.2 bigvoice_SetPresetManager() . . . 54
12.7 Direct processing . . . 54
12.7.1 Detailed Description . . . 55
12.7.2 Function Documentation . . . 55
12.7.2.1 bigvoice_GetBufferIn() . . . 55
12.7.2.2 bigvoice_GetBufferOut() . . . 55
12.7.2.3 bigvoice_GetEstimatedDelay() . . . 56
12.7.2.4 bigvoice_ProcessAudio() . . . 56
12.7.2.5 bigvoice_ProcessAudio_Planar() . . . 57
12.8 Examples . . . 57
12.9 Global setup . . . 58
12.9.1 Detailed Description . . . 58
12.9.2 Macro Definition Documentation . . . 59
12.9.2.1 BIGVOICE_AUDIO_INPUT_COUNT . . . 59
12.9.2.2 BIGVOICE_AUDIO_OUTPUT_COUNT . . . 59
12.9.2.3 BIGVOICE_AUDIOFRAME_COUNT . . . 59

12.9.2.4 BIGVOICE_CHANNEL_COUNT . . . 59
12.9.2.5 BIGVOICE_SAMPLE_RATE . . . 60
12.9.3 Typedef Documentation . . . 60
12.9.3.1 bigvoice_loggerfn . . . 60
12.9.4 Enumeration Type Documentation . . . 60
12.9.4.1 bigvoice_LogSeverity . . . 60
12.9.5 Function Documentation . . . 61
12.9.5.1 bigvoice_GetAudioInputCount() . . . 61
12.9.5.2 bigvoice_GetAudioOutputCount() . . . 61
12.9.5.3 bigvoice_GetChannelCount() . . . 61
12.9.5.4 bigvoice_GetChunkSizeInFrames() . . . 62
12.9.5.5 bigvoice_GetPossibleChunkSizeInFrames() . . . 62
12.9.5.6 bigvoice_GetSampleRate() . . . 62
12.9.5.7 bigvoice_GetVersion() . . . 63
12.9.5.8 bigvoice_SetLoggerCallback() . . . 63
12.9.5.9 bigvoice_SetLogSeverity() . . . 63
12.9.5.10 SOUND4_GetProcessInfo() . . . 63
12.10 Instance tracing (Windows only) . . . 64
12.10.1 Detailed Description . . . 64
12.10.2 Function Documentation . . . 64
12.10.2.1 bigvoice_SetInstanceTracing() . . . 64
12.10.2.2 bigvoice_SetInstanceTracingProcessActivity() . . . 64
12.11 Instance's bus . . . 65
12.11.1 Detailed Description . . . 65
12.11.2 Function Documentation . . . 65
12.11.2.1 bigvoice_FreeBus() . . . 65
12.11.2.2 bigvoice_NewBus() . . . 65
12.11.2.3 bigvoice_SetInstanceBus() . . . 66
12.12 Metadata Management . . . 66
12.12.1 Detailed Description . . . 66
12.12.2 Function Documentation . . . 66
12.12.2.1 bigvoice_FreeMetadataInfos() . . . 66
12.12.2.2 bigvoice_GetMetadataInfos() . . . 67
12.12.2.3 bigvoice_SetMetadata() . . . 67
12.12.2.4 bigvoice_SetMetadataMulti() . . . 67
12.13 Process creation/destruction . . . 68
12.13.1 Detailed Description . . . 68
12.13.2 Function Documentation . . . 68
12.13.2.1 bigvoice_ExitProcess() . . . 68
12.13.2.2 bigvoice_FreeParameters() . . . 69
12.13.2.3 bigvoice_FreeParameterValue() . . . 69
12.13.2.4 bigvoice_GetParameter() . . . 69

12.13.2.5 bigvoice_GetProcessChunkFrames() . . . 69
12.13.2.6 bigvoice_InitProcess() . . . 70
12.13.2.7 bigvoice_InitProcess2() . . . 70
12.13.2.8 bigvoice_InitProcess3() . . . 71
12.13.2.9 bigvoice_NewParameters() . . . 71
12.13.2.10 bigvoice_SetParameter() . . . 72
12.13.2.11 bigvoice_TerminateProcess() . . . 72
12.14 Sample using C interface . . . 72
12.14.1 Sample to create the processing instance . . . 72
12.14.2 Sample processing loop . . . 73
12.14.3 Sample to stop the processing instance . . . 74
12.15 Sample using C++ interface wrapper . . . 74
12.15.1 Setup and sanity checks . . . 74
12.15.2 Sample to create the processing instance . . . 74
12.15.3 Communicating with the instance . . . 74
12.15.4 Starting a web-server . . . 75
12.15.5 Sample processing loop . . . 75
12.16 Sample using C++ interface wrapper with dynamic library loading . . . 75
12.16.1 Dynamic loading . . . 75
12.16.2 Setup and sanity checks . . . 75
12.16.3 Sample to create the processing instance . . . 75
12.16.4 Communicating with the instance . . . 75
12.16.5 Starting a web-server . . . 76
12.16.6 Sample processing loop . . . 76
12.16.7 Custom preset manager . . . 76
12.16.8 Custom preset manager loading . . . 77
12.17 Sample using Generic C++ interface (multi-process compatible) . . . 77
12.17.1 Loading the library . . . 77
12.17.2 Setup and configure . . . 77
12.17.3 Create the processing instance . . . 78
12.17.4 Processing loop . . . 78
12.17.5 Starting a web-server . . . 78
12.17.6 Communicating with the instance . . . 78
12.17.7 Custom preset manager example . . . 78
12.17.8 Custom preset manager loading . . . 79
12.18 Update Management . . . 79
12.18.1 Detailed Description . . . 80
12.18.2 Function Documentation . . . 80
12.18.2.1 bigvoice_StartUpdateThread() . . . 80
12.18.2.2 bigvoice_StopUpdateThread() . . . 80
12.18.2.3 bigvoice_WaitUpdateThreadReady() . . . 80
12.19 Web server . . . 81

12.19.1 Detailed Description . . . 81
12.19.2 Macro Definition Documentation . . . 81
12.19.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 81
12.19.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 82
12.19.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 82
12.19.3 Function Documentation . . . 82
12.19.3.1 bigvoice_Webserver() . . . 82
12.19.3.2 bigvoice_Webserver_FreeString() . . . 83
12.19.3.3 bigvoice_Webserver_GetAppHealth() . . . 83
12.19.3.4 bigvoice_Webserver_SetAppHealth() . . . 83
12.19.3.5 bigvoice_Webserver_Status() . . . 84
12.19.3.6 bigvoice_Webserver_Stop() . . . 84
12.19.3.7 bigvoice_Webserver_tcp() . . . 84
12.19.3.8 bigvoice_Webserver_tcp2() . . . 85

**13 Namespace Documentation** **87**
13.1 sound4 Namespace Reference . . . 87
13.1.1 Detailed Description . . . 88
13.1.2 Typedef Documentation . . . 88
13.1.2.1 log_cb_t . . . 88
13.1.3 Enumeration Type Documentation . . . 88
13.1.3.1 LogSeverity . . . 88
13.1.3.2 PresetChange_Kind . . . 89
13.1.3.3 SampleFormat . . . 89
13.1.4 Function Documentation . . . 89
13.1.4.1 _log_cb_c() . . . 89
13.1.5 Variable Documentation . . . 90
13.1.5.1 _log_cb . . . 90
13.2 sound4::bigvoice Namespace Reference . . . 90
13.2.1 Detailed Description . . . 91
13.2.2 Typedef Documentation . . . 91
13.2.2.1 log_cb_t . . . 91
13.2.2.2 LogSeverity . . . 91
13.2.2.3 PresetChange_Kind . . . 91
13.2.2.4 SampleFormat . . . 91
13.2.3 Function Documentation . . . 92
13.2.3.1 _log_cb_c() . . . 92
13.2.3.2 GetAudioInputCount() . . . 92
13.2.3.3 GetAudioOutputCount() . . . 92
13.2.3.4 GetBytesFromFormat() . . . 92
13.2.3.5 GetChannelCount() . . . 93
13.2.3.6 GetChunkSizeInFrames() . . . 93

13.2.3.7 GetFormatFromName() . . . 93
13.2.3.8 GetFormatName() . . . 94
13.2.3.9 GetPossibleChunkSizeInFrames() . . . 94
13.2.3.10 GetSampleRate() . . . 95
13.2.3.11 GetVersion() . . . 95
13.2.3.12 SanityCheck() . . . 95
13.2.3.13 SetLoggerCallback() . . . 95
13.2.3.14 SetLogSeverity() . . . 96
13.2.4 Variable Documentation . . . 96
13.2.4.1 _log_cb . . . 96
13.2.4.2 ChunkFrames . . . 96
13.2.4.3 InputChannels . . . 96
13.2.4.4 InputSampleSize . . . 96
13.2.4.5 OutputChannels . . . 97
13.2.4.6 OutputSampleSize . . . 97
13.2.4.7 process_name . . . 97
13.2.4.8 process_shortname . . . 97
13.2.4.9 SampleRate . . . 97
13.3 sound4::bigvoice::dyn Namespace Reference . . . 98
13.3.1 Detailed Description . . . 99
13.3.2 Typedef Documentation . . . 99
13.3.2.1 log_cb_t . . . 99
13.3.2.2 LogSeverity . . . 99
13.3.2.3 PresetChange_Kind . . . 99
13.3.2.4 SampleFormat . . . 99
13.3.3 Function Documentation . . . 99
13.3.3.1 _log_cb_c() . . . 99
13.3.3.2 GetAudioInputCount() . . . 99
13.3.3.3 GetAudioOutputCount() . . . 100
13.3.3.4 GetBytesFromFormat() . . . 100
13.3.3.5 GetChannelCount() . . . 101
13.3.3.6 GetChunkSizeInFrames() . . . 101
13.3.3.7 GetFormatFromName() . . . 101
13.3.3.8 GetFormatName() . . . 102
13.3.3.9 GetPossibleChunkSizeInFrames() . . . 102
13.3.3.10 GetSampleRate() . . . 103
13.3.3.11 GetVersion() . . . 103
13.3.3.12 SanityCheck() . . . 103
13.3.3.13 SetLoggerCallback() . . . 105
13.3.3.14 SetLogSeverity() . . . 105
13.3.4 Variable Documentation . . . 105
13.3.4.1 _log_cb . . . 105

13.3.4.2 ChunkFrames . . . 105
13.3.4.3 InputChannels . . . 106
13.3.4.4 InputSampleSize . . . 106
13.3.4.5 OutputChannels . . . 106
13.3.4.6 OutputSampleSize . . . 106
13.3.4.7 process_name . . . 106
13.3.4.8 process_shortname . . . 107
13.3.4.9 SampleRate . . . 107
13.4 sound4::bigvoice::dyn::helper Namespace Reference . . . 107
13.4.1 Detailed Description . . . 108
13.4.2 Function Documentation . . . 108
13.4.2.1 AudioConvertFrom() . . . 108
13.4.2.2 AudioConvertTo() . . . 108
13.4.2.3 AudioMonoFromLiveStereo() . . . 109
13.4.2.4 AudioMonoToLiveStereo() . . . 109
13.4.2.5 MonoToStereo() . . . 109
13.4.2.6 MonoToStereo_Planar() . . . 110
13.4.2.7 StereoToMono() . . . 110
13.4.2.8 StereoToMono_Planar() . . . 111
13.4.2.9 UTF8ToWString() . . . 111
13.4.2.10 WStringToUTF8() . . . 112
13.5 sound4::bigvoice::helper Namespace Reference . . . 112
13.5.1 Detailed Description . . . 112
13.5.2 Function Documentation . . . 113
13.5.2.1 AudioConvertFrom() . . . 113
13.5.2.2 AudioConvertTo() . . . 113
13.5.2.3 AudioMonoFromLiveStereo() . . . 114
13.5.2.4 AudioMonoToLiveStereo() . . . 114
13.5.2.5 MonoToStereo() . . . 114
13.5.2.6 MonoToStereo_Planar() . . . 115
13.5.2.7 StereoToMono() . . . 115
13.5.2.8 StereoToMono_Planar() . . . 116
13.5.2.9 UTF8ToWString() . . . 116
13.5.2.10 WStringToUTF8() . . . 116
13.6 sound4::helper Namespace Reference . . . 117
13.6.1 Detailed Description . . . 117
13.6.2 Function Documentation . . . 117
13.6.2.1 UTF8ToWString() . . . 117
13.6.2.2 WStringToUTF8() . . . 117

**14 Class Documentation** **119**

14.1 sound4::AuxiliaryOutputInfos Struct Reference . . . 119

14.1.1 Detailed Description . . . 119
14.1.2 Member Data Documentation . . . 119
14.1.2.1 buffer . . . 119
14.1.2.2 channel_count . . . 120
14.1.2.3 description . . . 120
14.1.2.4 id . . . 120
14.1.2.5 sample_rate . . . 120
14.2 bigvoice_CAudioConverter Struct Reference . . . 120
14.2.1 Detailed Description . . . 120
14.3 bigvoice_CBus Struct Reference . . . 121
14.3.1 Detailed Description . . . 121
14.4 bigvoice_CClientInstance Struct Reference . . . 121
14.4.1 Detailed Description . . . 121
14.5 bigvoice_CInstance Struct Reference . . . 121
14.5.1 Detailed Description . . . 121
14.6 bigvoice_CParameters Struct Reference . . . 121
14.6.1 Detailed Description . . . 122
14.7 sound4::bigvoice::CAudioConverter Class Reference . . . 122
14.7.1 Detailed Description . . . 122
14.7.2 Constructor & Destructor Documentation . . . 122
14.7.2.1 CAudioConverter() . . . 122
14.7.2.2 ~CAudioConverter() . . . 122
14.7.3 Member Function Documentation . . . 123
14.7.3.1 Configure() . . . 123
14.7.3.2 ConvertFrom() . . . 123
14.7.3.3 ConvertTo() . . . 124
14.7.3.4 Delete() . . . 124
14.7.3.5 IsOK() . . . 124
14.8 sound4::bigvoice::dyn::CAudioConverter Class Reference . . . 124
14.8.1 Detailed Description . . . 125
14.8.2 Constructor & Destructor Documentation . . . 125
14.8.2.1 CAudioConverter() . . . 125
14.8.2.2 ~CAudioConverter() . . . 125
14.8.3 Member Function Documentation . . . 125
14.8.3.1 Configure() . . . 125
14.8.3.2 ConvertFrom() . . . 127
14.8.3.3 ConvertTo() . . . 127
14.8.3.4 Delete() . . . 128
14.8.3.5 IsOK() . . . 128
14.9 sound4::CAudioConverter Class Reference . . . 128
14.9.1 Detailed Description . . . 128
14.9.2 Constructor & Destructor Documentation . . . 128

14.9.2.1 CAudioConverter() . . . 128
14.9.2.2 ~CAudioConverter() . . . 129
14.9.3 Member Function Documentation . . . 129
14.9.3.1 Configure() . . . 129
14.9.3.2 ConvertFrom() . . . 130
14.9.3.3 ConvertTo() . . . 130
14.9.3.4 Delete() . . . 130
14.9.3.5 IsOK() . . . 131
14.10 sound4::bigvoice::CBus Class Reference . . . 131
14.10.1 Detailed Description . . . 131
14.10.2 Constructor & Destructor Documentation . . . 131
14.10.2.1 CBus() . . . 131
14.10.2.2 ~CBus() . . . 131
14.10.3 Member Function Documentation . . . 132
14.10.3.1 Get() . . . 132
14.11 sound4::bigvoice::dyn::CBus Class Reference . . . 132
14.11.1 Detailed Description . . . 132
14.11.2 Constructor & Destructor Documentation . . . 132
14.11.2.1 CBus() . . . 132
14.11.2.2 ~CBus() . . . 132
14.11.3 Member Function Documentation . . . 133
14.11.3.1 Get() . . . 133
14.12 sound4::CBus Class Reference . . . 133
14.12.1 Detailed Description . . . 133
14.12.2 Constructor & Destructor Documentation . . . 133
14.12.2.1 CBus() . . . 133
14.12.2.2 ~CBus() . . . 134
14.12.3 Member Function Documentation . . . 134
14.12.3.1 Get() . . . 134
14.12.4 Friends And Related Symbol Documentation . . . 134
14.12.4.1 CInstance . . . 134
14.13 sound4::bigvoice::CInstance::CClient Class Reference . . . 134
14.13.1 Detailed Description . . . 134
14.13.2 Constructor & Destructor Documentation . . . 135
14.13.2.1 CClient() [1/2] . . . 135
14.13.2.2 CClient() [2/2] . . . 135
14.13.2.3 ~CClient() . . . 135
14.13.3 Member Function Documentation . . . 135
14.13.3.1 operator=() . . . 135
14.13.3.2 ProcessJson() . . . 135
14.14 sound4::bigvoice::dyn::CInstance::CClient Class Reference . . . 136
14.14.1 Detailed Description . . . 136

14.14.2 Constructor & Destructor Documentation . . . 136
14.14.2.1 CClient() [1/2] . . . 136
14.14.2.2 CClient() [2/2] . . . 136
14.14.2.3 ~CClient() . . . 136
14.14.3 Member Function Documentation . . . 136
14.14.3.1 operator=() . . . 136
14.14.3.2 ProcessJson() . . . 136
14.15 sound4::CInstance::CClient Class Reference . . . 137
14.15.1 Detailed Description . . . 137
14.15.2 Constructor & Destructor Documentation . . . 137
14.15.2.1 CClient() [1/2] . . . 137
14.15.2.2 CClient() [2/2] . . . 138
14.15.2.3 ~CClient() . . . 138
14.15.3 Member Function Documentation . . . 138
14.15.3.1 operator=() . . . 138
14.15.3.2 ProcessJson() . . . 138
14.16 sound4::bigvoice::dyn::CDynLib Class Reference . . . 138
14.16.1 Detailed Description . . . 140
14.16.2 Constructor & Destructor Documentation . . . 140
14.16.2.1 CDynLib() . . . 140
14.16.3 Member Function Documentation . . . 140
14.16.3.1 IsOk() . . . 140
14.16.3.2 Load() . . . 141
14.16.4 Member Data Documentation . . . 141
14.16.4.1 AddAudio . . . 141
14.16.4.2 AddPadAudio . . . 141
14.16.4.3 AudioConverter_From . . . 141
14.16.4.4 AudioConverter_To . . . 141
14.16.4.5 AudioConvertFrom . . . 142
14.16.4.6 AudioConvertTo . . . 142
14.16.4.7 AudioMonoFromLiveStereo . . . 142
14.16.4.8 AudioMonoToLiveStereo . . . 142
14.16.4.9 DeleteClient . . . 142
14.16.4.10 ExitProcess . . . 142
14.16.4.11 FreeAudioConverter . . . 142
14.16.4.12 FreeBus . . . 143
14.16.4.13 FreeJsonAnswer . . . 143
14.16.4.14 FreeMetadataInfos . . . 143
14.16.4.15 FreeParameters . . . 143
14.16.4.16 FreeParameterValue . . . 143
14.16.4.17 GetAudio . . . 143
14.16.4.18 GetAudioConverter . . . 143

14.16.4.19 GetAudioInputCount . . . 144
14.16.4.20 GetAudioOutputCount . . . 144
14.16.4.21 GetBufferIn . . . 144
14.16.4.22 GetBufferOut . . . 144
14.16.4.23 GetBytesFromFormat . . . 144
14.16.4.24 GetChannelCount . . . 144
14.16.4.25 GetChunkSizeInFrames . . . 144
14.16.4.26 GetEstimatedDelay . . . 145
14.16.4.27 GetFormatFromName . . . 145
14.16.4.28 GetFormatName . . . 145
14.16.4.29 GetMaxPacketFrame . . . 145
14.16.4.30 GetMetadataInfos . . . 145
14.16.4.31 GetOutputCount . . . 145
14.16.4.32 GetParameter . . . 145
14.16.4.33 GetPossibleChunkSizeInFrames . . . 146
14.16.4.34 GetProcessChunkFrames . . . 146
14.16.4.35 GetSampleRate . . . 146
14.16.4.36 GetVersion . . . 146
14.16.4.37 InitProcess . . . 146
14.16.4.38 InitProcess2 . . . 146
14.16.4.39 InitProcess3 . . . 146
14.16.4.40 MonoToStereo . . . 147
14.16.4.41 MonoToStereo_Planar . . . 147
14.16.4.42 NewBus . . . 147
14.16.4.43 NewClient . . . 147
14.16.4.44 NewParameters . . . 147
14.16.4.45 PresetManager_InformChange . . . 147
14.16.4.46 ProcessAudio . . . 147
14.16.4.47 ProcessAudio_Planar . . . 148
14.16.4.48 ProcessJson . . . 148
14.16.4.49 SaveState . . . 148
14.16.4.50 SetInstanceBus . . . 148
14.16.4.51 SetInstanceTracing . . . 148
14.16.4.52 SetInstanceTracingProcessActivity . . . 148
14.16.4.53 SetLoggerCallback . . . 148
14.16.4.54 SetLogSeverity . . . 149
14.16.4.55 SetMetadata . . . 149
14.16.4.56 SetMetadataMulti . . . 149
14.16.4.57 SetParameter . . . 149
14.16.4.58 SetPresetManager . . . 149
14.16.4.59 StartUpdateThread . . . 149
14.16.4.60 StereoToMono . . . 149

14.16.4.61 StereoToMono_Planar . . . 150
14.16.4.62 StopUpdateThread . . . 150
14.16.4.63 TerminateProcess . . . 150
14.16.4.64 WaitUpdateThreadReady . . . 150
14.16.4.65 Webserver . . . 150
14.16.4.66 Webserver_FreeString . . . 150
14.16.4.67 Webserver_GetAppHealth . . . 150
14.16.4.68 Webserver_SetAppHealth . . . 151
14.16.4.69 Webserver_Status . . . 151
14.16.4.70 Webserver_Stop . . . 151
14.16.4.71 Webserver_tcp . . . 151
14.16.4.72 Webserver_tcp2 . . . 151
14.17 sound4::bigvoice::dyn::helper::CDynLoader Class Reference . . . 151
14.17.1 Detailed Description . . . 152
14.17.2 Constructor & Destructor Documentation . . . 152
14.17.2.1 CDynLoader() [1/3] . . . 152
14.17.2.2 CDynLoader() [2/3] . . . 152
14.17.2.3 CDynLoader() [3/3] . . . 152
14.17.2.4 ~CDynLoader() . . . 152
14.17.3 Member Function Documentation . . . 153
14.17.3.1 Close() . . . 153
14.17.3.2 GetSymbol() . . . 153
14.17.3.3 GetSymbol_safe() . . . 153
14.17.3.4 GetThisLibraryPath() . . . 153
14.17.3.5 IsOk() . . . 153
14.17.3.6 Load() . . . 153
14.17.3.7 operator bool() . . . 154
14.17.3.8 operator=() . . . 154
14.18 sound4::helper::CDynLoader Class Reference . . . 154
14.18.1 Detailed Description . . . 154
14.18.2 Constructor & Destructor Documentation . . . 155
14.18.2.1 CDynLoader() [1/3] . . . 155
14.18.2.2 CDynLoader() [2/3] . . . 155
14.18.2.3 CDynLoader() [3/3] . . . 155
14.18.2.4 ~CDynLoader() . . . 155
14.18.3 Member Function Documentation . . . 155
14.18.3.1 Close() . . . 155
14.18.3.2 GetSymbol() . . . 155
14.18.3.3 GetSymbol_safe() . . . 155
14.18.3.4 GetThisLibraryPath() . . . 156
14.18.3.5 IsOk() . . . 156
14.18.3.6 Load() . . . 156

14.18.3.7 operator bool() . . . 156
14.18.3.8 operator=() . . . 156
14.19 sound4::bigvoice::CInstance Class Reference . . . 156
14.19.1 Detailed Description . . . 157
14.19.2 Constructor & Destructor Documentation . . . 158
14.19.2.1 CInstance() [1/2] . . . 158
14.19.2.2 CInstance() [2/2] . . . 158
14.19.2.3 ~CInstance() . . . 158
14.19.3 Member Function Documentation . . . 158
14.19.3.1 ActivateAuxiliaryAudioID() . . . 158
14.19.3.2 AddAudio() . . . 158
14.19.3.3 Create() . . . 159
14.19.3.4 GetAudio() . . . 159
14.19.3.5 GetAuxiliaryAudioOutputInfos() . . . 160
14.19.3.6 GetBufferIn() . . . 160
14.19.3.7 GetBufferOut() . . . 160
14.19.3.8 GetChunkFrames() . . . 161
14.19.3.9 GetEstimatedDelay() . . . 161
14.19.3.10 GetMetadataInfos() . . . 161
14.19.3.11 GetParam() . . . 161
14.19.3.12 GetWebServerAppHealth() . . . 162
14.19.3.13 IsOk() . . . 162
14.19.3.14 NewClient() . . . 162
14.19.3.15 OnUpdateThreadStart() . . . 162
14.19.3.16 OnUpdateThreadStop() . . . 163
14.19.3.17 operator=() . . . 163
14.19.3.18 PresetManager_InformChange() . . . 163
14.19.3.19 ProcessAnyAudio() . . . 163
14.19.3.20 ProcessAudio() . . . 164
14.19.3.21 ProcessAudio_Planar() . . . 164
14.19.3.22 SaveState() . . . 164
14.19.3.23 SetBus() . . . 165
14.19.3.24 SetInstanceTracing() . . . 165
14.19.3.25 SetInstanceTracingProcessActivity() . . . 165
14.19.3.26 SetMetadata() . . . 166
14.19.3.27 SetMetadataMulti() . . . 166
14.19.3.28 SetParam() . . . 166
14.19.3.29 SetPresetManager() . . . 167
14.19.3.30 SetWebServerAppHealth() . . . 167
14.19.3.31 StartWebServer() . . . 167
14.19.3.32 Stop() . . . 168
14.19.3.33 StopWebServer() . . . 168

14.19.3.34 Terminate() . . . 168
14.20 sound4::bigvoice::dyn::CInstance Class Reference . . . 169
14.20.1 Detailed Description . . . 170
14.20.2 Constructor & Destructor Documentation . . . 170
14.20.2.1 CInstance() [1/2] . . . 170
14.20.2.2 CInstance() [2/2] . . . 170
14.20.2.3 ~CInstance() . . . 170
14.20.3 Member Function Documentation . . . 170
14.20.3.1 AddAudio() . . . 170
14.20.3.2 Create() . . . 171
14.20.3.3 GetAudio() . . . 171
14.20.3.4 GetBufferIn() . . . 172
14.20.3.5 GetBufferOut() . . . 172
14.20.3.6 GetChunkFrames() . . . 172
14.20.3.7 GetEstimatedDelay() . . . 173
14.20.3.8 GetMetadataInfos() . . . 173
14.20.3.9 GetParam() . . . 173
14.20.3.10 GetWebServerAppHealth() . . . 174
14.20.3.11 IsOk() . . . 174
14.20.3.12 NewClient() . . . 174
14.20.3.13 OnUpdateThreadStart() . . . 174
14.20.3.14 OnUpdateThreadStop() . . . 174
14.20.3.15 operator=() . . . 174
14.20.3.16 PresetManager_InformChange() . . . 174
14.20.3.17 ProcessAnyAudio() . . . 175
14.20.3.18 ProcessAudio() . . . 175
14.20.3.19 ProcessAudio_Planar() . . . 176
14.20.3.20 SaveState() . . . 177
14.20.3.21 SetBus() . . . 177
14.20.3.22 SetInstanceTracing() . . . 177
14.20.3.23 SetInstanceTracingProcessActivity() . . . 178
14.20.3.24 SetMetadata() . . . 178
14.20.3.25 SetMetadataMulti() . . . 178
14.20.3.26 SetParam() . . . 179
14.20.3.27 SetPresetManager() . . . 179
14.20.3.28 SetWebServerAppHealth() . . . 179
14.20.3.29 StartWebServer() . . . 180
14.20.3.30 Stop() . . . 180
14.20.3.31 StopWebServer() . . . 180
14.20.3.32 Terminate() . . . 181
14.21 sound4::CInstance Class Reference . . . 181
14.21.1 Detailed Description . . . 182

14.21.2 Constructor & Destructor Documentation . . . 183
14.21.2.1 CInstance() [1/3] . . . 183
14.21.2.2 CInstance() [2/3] . . . 183
14.21.2.3 CInstance() [3/3] . . . 183
14.21.2.4 ~CInstance() . . . 183
14.21.3 Member Function Documentation . . . 184
14.21.3.1 AddAudio() . . . 184
14.21.3.2 Create() . . . 184
14.21.3.3 GetAudio() . . . 185
14.21.3.4 GetBufferIn() . . . 185
14.21.3.5 GetBufferOut() . . . 185
14.21.3.6 GetChunkFrames() . . . 186
14.21.3.7 GetEstimatedDelay() . . . 186
14.21.3.8 GetMetadataInfos() . . . 186
14.21.3.9 GetParam() . . . 186
14.21.3.10 GetWebServerAppHealth() . . . 187
14.21.3.11 IsOk() . . . 187
14.21.3.12 NewClient() . . . 187
14.21.3.13 OnUpdateThreadStart() . . . 187
14.21.3.14 OnUpdateThreadStop() . . . 188
14.21.3.15 operator=() . . . 188
14.21.3.16 PresetManager_InformChange() . . . 188
14.21.3.17 ProcessAnyAudio() . . . 188
14.21.3.18 ProcessAudio() . . . 189
14.21.3.19 ProcessAudio_Planar() . . . 189
14.21.3.20 SaveState() . . . 189
14.21.3.21 SetBus() . . . 190
14.21.3.22 SetInstanceTracing() . . . 190
14.21.3.23 SetInstanceTracingProcessActivity() . . . 190
14.21.3.24 SetMetadata() . . . 191
14.21.3.25 SetMetadataMulti() . . . 191
14.21.3.26 SetParam() . . . 191
14.21.3.27 SetPresetManager() . . . 192
14.21.3.28 SetWebServerAppHealth() . . . 192
14.21.3.29 StartWebServer() . . . 192
14.21.3.30 Stop() . . . 193
14.21.3.31 StopWebServer() . . . 193
14.21.3.32 Terminate() . . . 193
14.22 sound4::bigvoice::CPresetLoader Class Reference . . . 194
14.22.1 Detailed Description . . . 194
14.22.2 Constructor & Destructor Documentation . . . 194
14.22.2.1 CPresetLoader() . . . 194

14.22.2.2 ~CPresetLoader() . . . 194
14.22.3 Member Function Documentation . . . 195
14.22.3.1 Exists() . . . 195
14.22.3.2 GetAll() . . . 195
14.22.3.3 IsReadOnly() . . . 195
14.22.3.4 Read() . . . 195
14.22.3.5 Remove() . . . 195
14.22.3.6 Rename() . . . 195
14.22.3.7 Write() . . . 195
14.23 sound4::bigvoice::dyn::CPresetLoader Class Reference . . . 196
14.23.1 Detailed Description . . . 196
14.23.2 Constructor & Destructor Documentation . . . 196
14.23.2.1 CPresetLoader() . . . 196
14.23.2.2 ~CPresetLoader() . . . 196
14.23.3 Member Function Documentation . . . 196
14.23.3.1 Exists() . . . 196
14.23.3.2 GetAll() . . . 197
14.23.3.3 IsReadOnly() . . . 197
14.23.3.4 Read() . . . 197
14.23.3.5 Remove() . . . 197
14.23.3.6 Rename() . . . 197
14.23.3.7 Write() . . . 197
14.24 sound4::CPresetLoader Class Reference . . . 198
14.24.1 Detailed Description . . . 198
14.24.2 Constructor & Destructor Documentation . . . 198
14.24.2.1 CPresetLoader() . . . 198
14.24.2.2 ~CPresetLoader() . . . 198
14.24.3 Member Function Documentation . . . 198
14.24.3.1 Exists() . . . 198
14.24.3.2 GetAll() . . . 199
14.24.3.3 IsReadOnly() . . . 199
14.24.3.4 Read() . . . 199
14.24.3.5 Remove() . . . 199
14.24.3.6 Rename() . . . 199
14.24.3.7 Write() . . . 199
14.25 sound4::CProcessor Class Reference . . . 200
14.25.1 Detailed Description . . . 200
14.25.2 Constructor & Destructor Documentation . . . 201
14.25.2.1 CProcessor() . . . 201
14.25.3 Member Function Documentation . . . 201
14.25.3.1 GetAudioInputCount() . . . 201
14.25.3.2 GetAudioOutputCount() . . . 201

14.25.3.3 GetBytesFromFormat() . . . 201
14.25.3.4 GetChannelCount() . . . 202
14.25.3.5 GetChunkSizeInFrames() . . . 202
14.25.3.6 GetFormatFromName() . . . 202
14.25.3.7 GetFormatName() . . . 203
14.25.3.8 GetPossibleChunkSizeInFrames() . . . 203
14.25.3.9 GetPrefixSymbol() . . . 204
14.25.3.10 GetSampleRate() . . . 204
14.25.3.11 GetVersion() . . . 204
14.25.3.12 IsOk() . . . 204
14.25.3.13 Load() . . . 204
14.25.3.14 SetLoggerCallback() . . . 205
14.25.3.15 SetLogSeverity() . . . 205
14.25.4 Friends And Related Symbol Documentation . . . 205
14.25.4.1 CAudioConverter . . . 205
14.25.4.2 CBus . . . 205
14.25.4.3 CInstance . . . 206
14.26 sound4::bigvoice::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 206
14.26.1 Detailed Description . . . 206
14.26.2 Constructor & Destructor Documentation . . . 206
14.26.2.1 DynFuncHolder() [1/2] . . . 206
14.26.2.2 DynFuncHolder() [2/2] . . . 207
14.26.3 Member Function Documentation . . . 207
14.26.3.1 IsOk() . . . 207
14.26.3.2 operator bool() . . . 207
14.26.3.3 operator T*() . . . 207
14.27 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 207
14.27.1 Detailed Description . . . 207
14.27.2 Constructor & Destructor Documentation . . . 208
14.27.2.1 DynFuncHolder() [1/2] . . . 208
14.27.2.2 DynFuncHolder() [2/2] . . . 208
14.27.3 Member Function Documentation . . . 208
14.27.3.1 IsOk() . . . 208
14.27.3.2 operator bool() . . . 208
14.27.3.3 operator T*() . . . 208
14.28 MyPresetManager Class Reference . . . 209
14.28.1 Detailed Description . . . 209
14.28.2 Constructor & Destructor Documentation . . . 210
14.28.2.1 MyPresetManager() [1/2] . . . 210
14.28.2.2 ~MyPresetManager() [1/2] . . . 210
14.28.2.3 MyPresetManager() [2/2] . . . 210
14.28.2.4 ~MyPresetManager() [2/2] . . . 210

14.28.3 Member Function Documentation . . . 210
14.28.3.1 Exists() [1/2] . . . 210
14.28.3.2 Exists() [2/2] . . . 210
14.28.3.3 GetAll() [1/2] . . . 210
14.28.3.4 GetAll() [2/2] . . . 211
14.28.3.5 IsReadOnly() [1/2] . . . 211
14.28.3.6 IsReadOnly() [2/2] . . . 211
14.28.3.7 Read() [1/2] . . . 211
14.28.3.8 Read() [2/2] . . . 211
14.28.3.9 Remove() [1/2] . . . 211
14.28.3.10 Remove() [2/2] . . . 212
14.28.3.11 Rename() [1/2] . . . 212
14.28.3.12 Rename() [2/2] . . . 212
14.28.3.13 SetupPath() [1/2] . . . 212
14.28.3.14 SetupPath() [2/2] . . . 212
14.28.3.15 Write() [1/2] . . . 212
14.28.3.16 Write() [2/2] . . . 213
14.29 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference . . . 213
14.29.1 Detailed Description . . . 213
14.30 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference . . . 213
14.30.1 Detailed Description . . . 214
14.31 sound4::helper::SampleFormat< T > Struct Template Reference . . . 215
14.31.1 Detailed Description . . . 215
14.32 SOUND4_AuxiliaryOutputInfos Struct Reference . . . 215
14.32.1 Detailed Description . . . 216
14.32.2 Member Data Documentation . . . 216
14.32.2.1 buffer . . . 216
14.32.2.2 channel_count . . . 216
14.32.2.3 description . . . 216
14.32.2.4 id . . . 216
14.32.2.5 sample_rate . . . 216
14.33 SOUND4_ProcessInfo Struct Reference . . . 216
14.33.1 Detailed Description . . . 217
14.33.2 Member Data Documentation . . . 217
14.33.2.1 name . . . 217
14.33.2.2 prefix . . . 217
14.33.2.3 shortname . . . 217
14.33.2.4 struct_size . . . 217
14.33.2.5 version . . . 217

**15 File Documentation** **219**

15.1 doc_examples.h File Reference . . . 219

15.2 doc_examples.h . . . 219
15.3 doc_gencpp.h File Reference . . . 219
15.4 doc_gencpp.h . . . 219
15.5 doc_json.h File Reference . . . 219
15.6 doc_json.h . . . 219
15.7 doc_main.h File Reference . . . 220
15.8 doc_main.h . . . 220
15.9 doc_params.h File Reference . . . 220
15.10 doc_params.h . . . 220
15.11 doc_only/documentation.h File Reference . . . 220
15.12 doc_only/documentation.h . . . 220
15.13 documentation.h File Reference . . . 220
15.14 documentation.h . . . 220
15.15 sound4.bigvoice.cl-pipe-cxx.cpp File Reference . . . 220
15.15.1 Function Documentation . . . 221
15.15.1.1 GetEnv() . . . 221
15.15.1.2 main() . . . 221
15.15.1.3 MyLogger() . . . 221
15.15.1.4 ProcessLoop() . . . 221
15.15.1.5 Usage() . . . 222
15.16 sound4.bigvoice.cl-pipe-cxx.cpp . . . 222
15.17 sound4.bigvoice.cl-pipe-dyn.cpp File Reference . . . 224
15.17.1 Function Documentation . . . 224
15.17.1.1 GetEnv() . . . 224
15.17.1.2 main() . . . 225
15.17.1.3 MyLogger() . . . 225
15.17.1.4 ProcessLoop() . . . 225
15.17.1.5 Usage() . . . 225
15.18 sound4.bigvoice.cl-pipe-dyn.cpp . . . 226
15.19 sound4.bigvoice.cl-pipe.cpp File Reference . . . 229
15.19.1 Macro Definition Documentation . . . 229
15.19.1.1 CHANNEL_COUNT . . . 229
15.19.1.2 CHUNK_SIZE . . . 229
15.19.1.3 SAMPLE_RATE . . . 230
15.19.2 Function Documentation . . . 230
15.19.2.1 main() . . . 230
15.19.2.2 MyLogger() . . . 230
15.19.2.3 ProcessLoop() . . . 230
15.19.2.4 Usage() . . . 230
15.20 sound4.bigvoice.cl-pipe.cpp . . . 231
15.21 sound4cl-pipe.cpp File Reference . . . 234
15.21.1 Function Documentation . . . 234

15.21.1.1 GetEnv() . . . 234
15.21.1.2 main() . . . 235
15.21.1.3 ProcessLoop() . . . 235
15.22 sound4cl-pipe.cpp . . . 236
15.23 sound4.bigvoice.cl.h File Reference . . . 239
15.23.1 Macro Definition Documentation . . . 242
15.23.1.1 BIGVOICE_DECL . . . 242
15.23.1.2 BIGVOICE_HAS_CLOUDBUS . . . 242
15.23.1.3 BIGVOICE_HAS_WEBSERVER . . . 242
15.23.2 Typedef Documentation . . . 242
15.23.2.1 TraceLoggingHProvider . . . 242
15.24 sound4.bigvoice.cl.h . . . 243
15.25 sound4.bigvoice.cl.hpp File Reference . . . 247
15.25.1 Function Documentation . . . 249
15.25.1.1 bigvoice_custom_exists() . . . 249
15.25.1.2 bigvoice_custom_getall() . . . 250
15.25.1.3 bigvoice_custom_getall_free() . . . 250
15.25.1.4 bigvoice_custom_reader() . . . 250
15.25.1.5 bigvoice_custom_reader_free() . . . 250
15.25.1.6 bigvoice_custom_remove() . . . 250
15.25.1.7 bigvoice_custom_rename() . . . 250
15.25.1.8 bigvoice_custom_writer() . . . 251
15.26 sound4.bigvoice.cl.hpp . . . 251
15.27 sound4.bigvoice.cl_dyn.hpp File Reference . . . 259
15.27.1 Detailed Description . . . 262
15.27.2 Macro Definition Documentation . . . 262
15.27.2.1 UNIXLIKE . . . 262
15.27.3 Function Documentation . . . 262
15.27.3.1 bigvoice_custom_exists() . . . 262
15.27.3.2 bigvoice_custom_getall() . . . 262
15.27.3.3 bigvoice_custom_getall_free() . . . 262
15.27.3.4 bigvoice_custom_reader() . . . 262
15.27.3.5 bigvoice_custom_reader_free() . . . 263
15.27.3.6 bigvoice_custom_remove() . . . 263
15.27.3.7 bigvoice_custom_rename() . . . 263
15.27.3.8 bigvoice_custom_writer() . . . 263
15.28 sound4.bigvoice.cl_dyn.hpp . . . 263
15.29 sound4cl.hpp File Reference . . . 278
15.29.1 Detailed Description . . . 279
15.29.2 Macro Definition Documentation . . . 280
15.29.2.1 UNIXLIKE . . . 280
15.30 sound4cl.hpp . . . 280

15.31 sound4cl_cdef.h File Reference . . . . . 295
15.31.1 Detailed Description . . . . . 295
15.31.2 Macro Definition Documentation . . . . . 295
15.31.2.1 SOUND4_INVALID_WEBSERVER_ID . . . . . 295
15.31.2.2 SOUND4_WEBSERVER_HTTP_OK . . . . . 295
15.31.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . . . 295
15.31.3 Typedef Documentation . . . . . 296
15.31.3.1 fs_char . . . . . 296
15.31.3.2 TraceLoggingHProvider . . . . . 296
15.32 sound4cl_cdef.h . . . . . 296

**Index** **301**

# 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++) . . . 35
Audio managing (any audio) . . . 39
Auxiliary outputs . . . 44
Client's JSon config interfacing . . . 46
Custom preset managagement . . . 49
Direct processing . . . 54
Examples . . . 57
Global setup . . . 58
Instance tracing (Windows only) . . . 64
Instance's bus . . . 65
Metadata Management . . . 66
Process creation/destruction . . . 68
Sample using C interface . . . 72
Sample using C++ interface wrapper . . . 74
Sample using C++ interface wrapper with dynamic library loading . . . 75
Sample using Generic C++ interface (multi-process compatible) . . . 77
Update Management . . . 79
Web server . . . 81

# Chapter 8

# Namespace Index

## 8.1 Namespace List

Here is a list of all namespaces with brief descriptions:


sound4 . . . . . 87
sound4::bigvoice . . . . . 90
sound4::bigvoice::dyn . . . . . 98
sound4::bigvoice::dyn::helper . . . . . 107
sound4::bigvoice::helper . . . . . 112
sound4::helper . . . . . 117

# Chapter 9

# Hierarchical Index

## 9.1 Class Hierarchy

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


sound4::AuxiliaryOutputInfos . . . 119
bigvoice_CAudioConverter . . . 120
bigvoice_CBus . . . 121
bigvoice_CClientInstance . . . 121
bigvoice_CInstance . . . 121
bigvoice_CParameters . . . 121
sound4::bigvoice::CAudioConverter . . . 122
sound4::bigvoice::dyn::CAudioConverter . . . 124
sound4::CAudioConverter . . . 128
sound4::bigvoice::CBus . . . 131
sound4::bigvoice::dyn::CBus . . . 132
sound4::CBus . . . 133
sound4::bigvoice::CInstance::CClient . . . 134
sound4::bigvoice::dyn::CInstance::CClient . . . 136
sound4::CInstance::CClient . . . 137
sound4::bigvoice::dyn::CDynLib . . . 138
sound4::bigvoice::dyn::helper::CDynLoader . . . 151
sound4::helper::CDynLoader . . . 154
sound4::bigvoice::CInstance . . . 156
sound4::bigvoice::dyn::CInstance . . . 169
sound4::CInstance . . . 181
sound4::bigvoice::CPresetLoader . . . 194
sound4::bigvoice::dyn::CPresetLoader . . . 196
MyPresetManager . . . 209
sound4::CPresetLoader . . . 198
MyPresetManager . . . 209
sound4::CProcessor . . . 200
sound4::bigvoice::dyn::helper::DynFuncHolder< T > . . . 206
sound4::helper::DynFuncHolder< T > . . . 207
sound4::bigvoice::dyn::helper::SampleFormat< T > . . . 213
sound4::bigvoice::helper::SampleFormat< T > . . . 213
sound4::helper::SampleFormat< T > . . . 215
SOUND4_AuxiliaryOutputInfos . . . 215
SOUND4_ProcessInfo . . . 216

# Chapter 10

# Class Index

## 10.1 Class List

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


sound4::AuxiliaryOutputInfos . . . 119
bigvoice_CAudioConverter . . . 120
bigvoice_CBus . . . 121
bigvoice_CClientInstance . . . 121
bigvoice_CInstance . . . 121
bigvoice_CParameters . . . 121
sound4::bigvoice::CAudioConverter
    Audio converter helper . . . 122
sound4::bigvoice::dyn::CAudioConverter
    Audio converter helper . . . 124
sound4::CAudioConverter
    Audio converter helper . . . 128
sound4::bigvoice::CBus
    Shared Bus . . . 131
sound4::bigvoice::dyn::CBus
    Shared Bus . . . 132
sound4::CBus
    Shared Bus . . . 133
sound4::bigvoice::CInstance::CClient . . . 134
sound4::bigvoice::dyn::CInstance::CClient . . . 136
sound4::CInstance::CClient . . . 137
sound4::bigvoice::dyn::CDynLib
    Dynamic library interface . . . 138
sound4::bigvoice::dyn::helper::CDynLoader
    Helper to load dynamic library . . . 151
sound4::helper::CDynLoader
    Helper to load dynamic library . . . 154
sound4::bigvoice::CInstance
    Instance handling class . . . 156
sound4::bigvoice::dyn::CInstance
    Instance handling class . . . 169
sound4::CInstance
    Instance handling class . . . 181
sound4::bigvoice::CPresetLoader
    Custom preset handler helper . . . 194

sound4::bigvoice::dyn::CPresetLoader
Custom preset handler helper . . . 196
sound4::CPresetLoader
Custom preset handler helper . . . 198
sound4::CProcessor
Dynamic library interface . . . 200
sound4::bigvoice::dyn::helper::DynFuncHolder< T > 
Helper to wrap a function pointer . . . 206
sound4::helper::DynFuncHolder< T > 
Helper to wrap a function pointer . . . 207
MyPresetManager
[Custom preset manager] . . . 209
sound4::bigvoice::dyn::helper::SampleFormat< T > 
Helper for Sample format types . . . 213
sound4::bigvoice::helper::SampleFormat< T > 
Helper for Sample format types . . . 213
sound4::helper::SampleFormat< T > 
Helper for Sample format types . . . 215
SOUND4_AuxiliaryOutputInfos . . . 215
SOUND4_ProcessInfo . . . 216

# Chapter 11

# File Index

## 11.1 File List

Here is a list of all files with brief descriptions:


doc_examples.h . . . 219
doc_gencpp.h . . . 219
doc_json.h . . . 219
doc_main.h . . . 220
doc_params.h . . . 220
doc_only/documentation.h . . . 220
documentation.h . . . 220
sound4.bigvoice.cl-pipe-cxx.cpp
C++ sample using C++ interface wrapper . . . 220
sound4.bigvoice.cl-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading . . . 224
sound4.bigvoice.cl-pipe.cpp
C++ sample using only C interface . . . 229
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided . . . 234
sound4.bigvoice.cl.h
C interface for library . . . 239
sound4.bigvoice.cl.hpp
C++ wrapper for library . . . 247
sound4.bigvoice.cl_dyn.hpp
C++ wrapper for dynamic loading library . . . 259
sound4cl.hpp
C++ wrapper for dynamic loading library . . . 278
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries . . . 295

# Chapter 12

# Topic Documentation

## 12.1 Audio convert helpers

Helpers to convert audio.

### Classes

- struct bigvoice_CAudioConverter

### Enumerations

- enum bigvoice_ConverterTargetMode { ConverterTargetMode_AsInput = 0 , ConverterTargetMode_Mono , ConverterTargetMode_Stereo , ConverterTargetMode_StereoToMono , ConverterTargetMode_MonoToStereo }

### 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)
- struct bigvoice_CAudioConverter * bigvoice_GetAudioConverter (enum bigvoice_SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, enum bigvoice_ConverterTargetMode target_mode)
- void bigvoice_FreeAudioConverter (struct bigvoice_CAudioConverter *converter)
- void bigvoice_AudioConverter_From (struct bigvoice_CAudioConverter *converter, const uint8_t *payload, float *output)
- void bigvoice_AudioConverter_To (struct bigvoice_CAudioConverter *converter, const float *input, uint8_t *payload)

### 12.1.1 Detailed Description

Helpers to convert audio in an optimized way

### 12.1.2 Enumeration Type Documentation

#### 12.1.2.1 bigvoice_ConverterTargetMode

```
enum bigvoice_ConverterTargetMode
```

Kind of process-side samples for CAudioConverter

**Enumerator**

| | |
|---|---|
| ConverterTargetMode_AsInput | Same channels as input (no extract/insert) |
| ConverterTargetMode_Mono | Mono process (using mono extract/payload) |
| ConverterTargetMode_Stereo | Stereo process (using stereo extract/payload) |
| ConverterTargetMode_StereoToMono | Mono process, using stereo extract/payload (converts stereo to mono) |
| ConverterTargetMode_MonoToStereo | Stereo process, using mono extract/payload (converts mono to stereo) |

Definition at line 1083 of file sound4.bigvoice.cl.h.

### 12.1.3 Function Documentation

#### 12.1.3.1 bigvoice_AudioConverter_From()

```
void bigvoice_AudioConverter_From (
            struct bigvoice_CAudioConverter * converter,
            const uint8_t * payload,
            float * output )
```

Convert the payload to floating-point array, using a predefined converter.

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *converter* | a converter created with bigvoice_GetAudioConverter |
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |

#### 12.1.3.2 bigvoice_AudioConverter_To()

```
void bigvoice_AudioConverter_To (
```

```
            struct bigvoice_CAudioConverter * converter,
            const float * input,
            uint8_t * payload )
```

Convert floating-point array to payload, using a predefined converter.

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *converter* | a converter created with bigvoice_GetAudioConverter |
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |

#### 12.1.3.3 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 |

#### 12.1.3.4 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 |

#### 12.1.3.5 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) |

#### 12.1.3.6 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 |

#### 12.1.3.7 bigvoice_FreeAudioConverter()

```
void bigvoice_FreeAudioConverter (
            struct bigvoice_CAudioConverter * converter )
```

Free the audio converter created with bigvoice_GetAudioConverter.

#### 12.1.3.8 bigvoice_GetAudioConverter()

```
struct bigvoice_CAudioConverter * bigvoice_GetAudioConverter (
            enum bigvoice_SampleFormat fmt,
            size_t nFrame,
```

```
            size_t payl_ch,
            size_t payl_ch_offset,
            enum bigvoice_ConverterTargetMode target_mode )
```

Prepare an audio converter for extract/insert and convert.

This helps to extract/insert a subset of channels from/to the payload, and convert it to floating-point format.

The floating-point channels count depends on `target_mode`. It can be the same as payload, or fixed to mono or stereo.

It can be used with bigvoice_AudioConverter_From and bigvoice_AudioConverter_To.

The converter will have to be freed with bigvoice_FreeAudioConverter.

**Parameters**

| | |
|---|---|
| *fmt* | the format of the payload |
| *nFrame* | the number of frames per packet |
| *payl_ch* | the number of channels in the payload |
| *payl_ch_offset* | the offset of the first channel to extract/insert in the payload [must be <= (payload_channels-output_channels)] |
| *target_mode* | defines the number of channels in process, and the number of channels to extract from payload for mono<->stereo conversion |

**Returns**

The converter or null if not possible

**Note**

The converter is not multithread-safe, it uses internal cache buffers.

#### 12.1.3.9 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 |

#### 12.1.3.10 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 |

#### 12.1.3.11 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 |

#### 12.1.3.12 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++)

### Classes

- class sound4::CAudioConverter

  *Audio converter helper.*

### Enumerations

- enum sound4::ConverterTargetMode {
  sound4::ConverterTargetMode_AsInput = sound4cl_ConverterTargetMode_AsInput , sound4::ConverterTargetMode_Mono = sound4cl_ConverterTargetMode_Mono , sound4::ConverterTargetMode_Stereo = sound4cl_Converter↩
  TargetMode_Stereo , sound4::ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_↩
  StereoToMono ,
  sound4::ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo }

### 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 Enumeration Type Documentation

#### 12.2.2.1 ConverterTargetMode

```
enum sound4::ConverterTargetMode
```

Kind of process-side samples for CAudioConverter

**Enumerator**

| | |
|---|---|
| ConverterTargetMode_AsInput | Same channels as input (no extract/insert) |
| ConverterTargetMode_Mono | Mono process (using mono extract/payload) |
| ConverterTargetMode_Stereo | Stereo process (using stereo extract/payload) |
| ConverterTargetMode_StereoToMono | Mono process, using stereo extract/payload (converts stereo to mono) |
| ConverterTargetMode_MonoToStereo | Stereo process, using mono extract/payload (converts mono to stereo) |

Definition at line 1571 of file sound4cl.hpp.

### 12.2.3 Function Documentation

#### 12.2.3.1 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 655 of file sound4cl.hpp.

#### 12.2.3.2 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 668 of file sound4cl.hpp.

#### 12.2.3.3 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 727 of file sound4cl.hpp.

#### 12.2.3.4 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 736 of file sound4cl.hpp.

#### 12.2.3.5 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 692 of file sound4cl.hpp.

#### 12.2.3.6 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 718 of file sound4cl.hpp.

#### 12.2.3.7 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 680 of file sound4cl.hpp.

#### 12.2.3.8 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 705 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 " « (const
      char*)store_path.u8string().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 " « (const
      char*)store_path.u8string().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 1336 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 1341 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 1345 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 CAudioConverter

  *Audio converter helper.*
- 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 }
- enum ConverterTargetMode {
  ConverterTargetMode_AsInput = sound4cl_ConverterTargetMode_AsInput , ConverterTargetMode_Mono =
  sound4cl_ConverterTargetMode_Mono , ConverterTargetMode_Stereo = sound4cl_ConverterTargetMode↩
  _Stereo , ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_StereoToMono ,
  ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo }

**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 908 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 CAudioConverter

  *Audio converter helper.*
- 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 CAudioConverter

  *Audio converter helper.*
- 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 876 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 872 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 950 of file sound4.bigvoice.cl_dyn.hpp.

#### 13.3.2.4 SampleFormat

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

Definition at line 510 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 883 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 775 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 788 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 864 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 762 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 729 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 852 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 840 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 743 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 801 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 713 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 813 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 897 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 890 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 881 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 515 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 520 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 530 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 525 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 535 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 508 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 509 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 540 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 606 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 621 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 690 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 701 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 649 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 679 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 635 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 664 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 564 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 552 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_CAudioConverter Struct Reference

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

### 14.2.1 Detailed Description

Abstract holder for an audio conveerter

**See also**

bigvoice_GetAudioConverter

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

- sound4.bigvoice.cl.h

## 14.3 bigvoice_CBus Struct Reference

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

### 14.3.1 Detailed Description

Abstract holder for a instances' bus

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

- sound4.bigvoice.cl.h

## 14.4 bigvoice_CClientInstance Struct Reference

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

### 14.4.1 Detailed Description

Abstract pointer for a client instance

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

- sound4.bigvoice.cl.h

## 14.5 bigvoice_CInstance Struct Reference

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

### 14.5.1 Detailed Description

Abstract holder for a process instance

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

- sound4.bigvoice.cl.h

## 14.6 bigvoice_CParameters Struct Reference

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

### 14.6.1 Detailed Description

Abstract holder for a process instance parameters

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

- sound4.bigvoice.cl.h

## 14.7 sound4::bigvoice::CAudioConverter Class Reference

Audio converter helper.

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

#### Public Member Functions

- CAudioConverter ()=default
  *Construct a new empty CAudioConverter object.*
- virtual ~CAudioConverter ()
  *Destroy the CAudioConverter object.*
- bool Configure (SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, bigvoice_ConverterTargetMode target_mode)
- void Delete ()
- bool IsOK () const
- void ConvertFrom (const uint8_t *payload, float *output)
- void ConvertTo (const float *input, uint8_t *payload)

### 14.7.1 Detailed Description

This helps to extract/insert a subset of channels from/to the payload, and convert it to floating-point format.

Definition at line 1108 of file sound4.bigvoice.cl.hpp.

### 14.7.2 Constructor & Destructor Documentation

#### 14.7.2.1 CAudioConverter()

`sound4::bigvoice::CAudioConverter::CAudioConverter ( ) [default]`

#### 14.7.2.2 ~CAudioConverter()

`virtual sound4::bigvoice::CAudioConverter::~CAudioConverter ( ) [inline], [virtual]`

Definition at line 1118 of file sound4.bigvoice.cl.hpp.

### 14.7.3 Member Function Documentation

#### 14.7.3.1 Configure()

```
bool sound4::bigvoice::CAudioConverter::Configure (
            SampleFormat fmt,
            size_t nFrame,
            size_t payl_ch,
            size_t payl_ch_offset,
            bigvoice_ConverterTargetMode target_mode )  [inline]
```

Prepare an audio converter for extract/insert and convert.

The floating-point channels count depends on `target_mode`. It can be the same as payload, or fixed to mono or stereo.

**Parameters**

| | |
|---|---|
| *fmt* | the format of the payload |
| *nFrame* | the number of frames per packet |
| *payl_ch* | the number of channels in the payload |
| *payl_ch_offset* | the offset of the first channel to extract/insert in the payload [must be <= (payl_ch-proc_ch)] |
| *target_mode* | defines the number of channels in process, and the number of channels to extract from payload for mono<->stereo conversion |

**Returns**

true if was created

**Note**

The converter is not multithread-safe, it uses internal cache buffers.

Definition at line 1138 of file sound4.bigvoice.cl.hpp.

#### 14.7.3.2 ConvertFrom()

```
void sound4::bigvoice::CAudioConverter::ConvertFrom (
            const uint8_t * payload,
            float * output )  [inline]
```

Convert the payload to floating-point array, using a predefined converter.

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |

Definition at line 1161 of file sound4.bigvoice.cl.hpp.

#### 14.7.3.3 ConvertTo()

```
void sound4::bigvoice::CAudioConverter::ConvertTo (
            const float * input,
            uint8_t * payload )  [inline]
```

Convert floating-point array to payload, using a predefined converter.

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |

Definition at line 1172 of file sound4.bigvoice.cl.hpp.

#### 14.7.3.4 Delete()

```
void sound4::bigvoice::CAudioConverter::Delete ( )  [inline]
```

Definition at line 1144 of file sound4.bigvoice.cl.hpp.

#### 14.7.3.5 IsOK()

```
bool sound4::bigvoice::CAudioConverter::IsOK ( ) const  [inline]
```

Definition at line 1151 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.8 sound4::bigvoice::dyn::CAudioConverter Class Reference

Audio converter helper.

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

**Public Member Functions**

- CAudioConverter (CDynLib &dynlib)

  *Construct a new empty CAudioConverter object.*
- virtual ~CAudioConverter ()

  *Destroy the CAudioConverter object.*
- bool Configure (SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, bigvoice_ConverterTargetMode target_mode)
- void Delete ()
- bool IsOK () const
- void ConvertFrom (const uint8_t *payload, float *output)
- void ConvertTo (const float *input, uint8_t *payload)

## 14.8.1 Detailed Description

This helps to extract/insert a subset of channels from/to the payload, and convert it to floating-point format.

Definition at line 1590 of file sound4.bigvoice.cl_dyn.hpp.

## 14.8.2 Constructor & Destructor Documentation

### 14.8.2.1 CAudioConverter()

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

**Parameters**

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

Definition at line 1597 of file sound4.bigvoice.cl_dyn.hpp.

### 14.8.2.2 ~CAudioConverter()

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

Definition at line 1604 of file sound4.bigvoice.cl_dyn.hpp.

## 14.8.3 Member Function Documentation

### 14.8.3.1 Configure()

```
bool sound4::bigvoice::dyn::CAudioConverter::Configure (
            SampleFormat fmt,
            size_t nFrame,
            size_t payl_ch,
```

```
        size_t payl_ch_offset,
        bigvoice_ConverterTargetMode target_mode )  [inline]
```

Prepare an audio converter for extract/insert and convert.

The floating-point channels count depends on `target_mode`. It can be the same as payload, or fixed to mono or stereo.

**Parameters**

| | |
|---|---|
| *fmt* | the format of the payload |
| *nFrame* | the number of frames per packet |
| *payl_ch* | the number of channels in the payload |
| *payl_ch_offset* | the offset of the first channel to extract/insert in the payload [must be <= (payload_channels-output_channels)] |
| *target_mode* | defines the number of channels in process, and the number of channels to extract from payload for mono<->stereo conversion |

**Returns**

true if was created

**Note**

The converter is not multithread-safe, it uses internal cache buffers.

Definition at line 1624 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.8.3.2 ConvertFrom()

```
void sound4::bigvoice::dyn::CAudioConverter::ConvertFrom (
            const uint8_t * payload,
            float * output )  [inline]
```

Convert the payload to floating-point array, using a predefined converter.

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |

Definition at line 1648 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.8.3.3 ConvertTo()

```
void sound4::bigvoice::dyn::CAudioConverter::ConvertTo (
            const float * input,
            uint8_t * payload )  [inline]
```

Convert floating-point array to payload, using a predefined converter.

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |

Definition at line 1659 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.8.3.4 Delete()

```
void sound4::bigvoice::dyn::CAudioConverter::Delete ( )  [inline]
```

Definition at line 1631 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.8.3.5 IsOK()

```
bool sound4::bigvoice::dyn::CAudioConverter::IsOK ( ) const  [inline]
```

Definition at line 1638 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.9 sound4::CAudioConverter Class Reference

Audio converter helper.

`#include <sound4cl.hpp>`

### Public Member Functions

- CAudioConverter (CProcessor &dynlib)
  *Construct a new empty CAudioConverter object.*
- virtual ~CAudioConverter ()
  *Destroy the CAudioConverter object.*
- bool Configure (SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, ConverterTargetMode target_mode)
- void Delete ()
- bool IsOK () const
- void ConvertFrom (const uint8_t *payload, float *output)
- void ConvertTo (const float *input, uint8_t *payload)

### 14.9.1 Detailed Description

This helps to extract/insert a subset of channels from/to the payload, and convert it to floating-point format.

Definition at line 1586 of file sound4cl.hpp.

### 14.9.2 Constructor & Destructor Documentation

#### 14.9.2.1 CAudioConverter()

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

**Parameters**

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

Definition at line 1593 of file sound4cl.hpp.

#### 14.9.2.2 ~CAudioConverter()

```
virtual sound4::CAudioConverter::~CAudioConverter ( )  [inline], [virtual]
```

Definition at line 1600 of file sound4cl.hpp.

### 14.9.3 Member Function Documentation

#### 14.9.3.1 Configure()

```
bool sound4::CAudioConverter::Configure (
            SampleFormat fmt,
            size_t nFrame,
            size_t payl_ch,
            size_t payl_ch_offset,
            ConverterTargetMode target_mode )  [inline]
```

Configure the converter.

The floating-point channels count depends on `target_mode`. It can be the same as payload, or fixed to mono or stereo.

**Parameters**

| | |
|---|---|
| *fmt* | the format of the payload |
| *nFrame* | the number of frames per packet |
| *payl_ch* | the number of channels in the payload |
| *payl_ch_offset* | the offset of the first channel to extract/insert in the payload [must be <= (payload_channels-output_channels)] |
| *target_mode* | defines the number of channels in process, and the number of channels to extract from payload for mono<->stereo conversion |

**Returns**

true if was created

**Note**

The converter is not multithread-safe, it uses internal cache buffers.

Definition at line 1620 of file sound4cl.hpp.

#### 14.9.3.2 ConvertFrom()

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

Convert the payload to floating-point array, using a predefined converter.

**Warning**

Output buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *payload* | the binary raw audio buffer |
| *output* | audio output buffer (64 byte aligned) |

Definition at line 1644 of file sound4cl.hpp.

#### 14.9.3.3 ConvertTo()

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

Convert floating-point array to payload, using a predefined converter.

**Warning**

Input buffer must be 64-byte aligned

**Parameters**

| | |
|---|---|
| *input* | audio input buffer (64 byte aligned) |
| *payload* | the binary raw audio buffer |

Definition at line 1655 of file sound4cl.hpp.

#### 14.9.3.4 Delete()

```
void sound4::CAudioConverter::Delete ( )  [inline]
```

Definition at line 1627 of file sound4cl.hpp.

#### 14.9.3.5 IsOK()

```
bool sound4::CAudioConverter::IsOK ( ) const  [inline]
```

Definition at line 1634 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.10 sound4::bigvoice::CBus Class Reference

Shared Bus.

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

### Public Member Functions

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

### 14.10.1 Detailed Description

Some processor can share a bus for realtime data exchange

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

### 14.10.2 Constructor & Destructor Documentation

#### 14.10.2.1 CBus()

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

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

#### 14.10.2.2 ~CBus()

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

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

### 14.10.3 Member Function Documentation

#### 14.10.3.1 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

## 14.11 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

### 14.11.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 909 of file sound4.bigvoice.cl_dyn.hpp.

### 14.11.2 Constructor & Destructor Documentation

#### 14.11.2.1 CBus()

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

Definition at line 911 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.11.2.2 ~CBus()

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

Definition at line 918 of file sound4.bigvoice.cl_dyn.hpp.

### 14.11.3 Member Function Documentation

#### 14.11.3.1 Get()

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

Definition at line 925 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.12 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

### 14.12.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 866 of file sound4cl.hpp.

### 14.12.2 Constructor & Destructor Documentation

#### 14.12.2.1 CBus()

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

Definition at line 868 of file sound4cl.hpp.

#### 14.12.2.2 ~CBus()

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

Definition at line 875 of file sound4cl.hpp.

### 14.12.3 Member Function Documentation

#### 14.12.3.1 Get()

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

Definition at line 884 of file sound4cl.hpp.

### 14.12.4 Friends And Related Symbol Documentation

#### 14.12.4.1 CInstance

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

Definition at line 883 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.13 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.*

### 14.13.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 986 of file sound4.bigvoice.cl.hpp.

## 14.13.2 Constructor & Destructor Documentation

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

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

Definition at line 988 of file sound4.bigvoice.cl.hpp.

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

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

### 14.13.2.3 ~CClient()

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

Definition at line 997 of file sound4.bigvoice.cl.hpp.

## 14.13.3 Member Function Documentation

### 14.13.3.1 operator=()

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

### 14.13.3.2 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 1011 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

## 14.14 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.*

### 14.14.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1461 of file sound4.bigvoice.cl_dyn.hpp.

### 14.14.2 Constructor & Destructor Documentation

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

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

Definition at line 1463 of file sound4.bigvoice.cl_dyn.hpp.

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

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

#### 14.14.2.3 ~CClient()

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

Definition at line 1473 of file sound4.bigvoice.cl_dyn.hpp.

### 14.14.3 Member Function Documentation

#### 14.14.3.1 operator=()

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

#### 14.14.3.2 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 1487 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.15 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.*

### 14.15.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1442 of file sound4cl.hpp.

### 14.15.2 Constructor & Destructor Documentation

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

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

Definition at line 1444 of file sound4cl.hpp.

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

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

#### 14.15.2.3 ~CClient()

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

Definition at line 1454 of file sound4cl.hpp.

### 14.15.3 Member Function Documentation

#### 14.15.3.1 operator=()

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

#### 14.15.3.2 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 1468 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.16 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
- helper::DynFuncHolder< decltype(bigvoice_GetAudioConverter) > GetAudioConverter
- helper::DynFuncHolder< decltype(bigvoice_FreeAudioConverter) > FreeAudioConverter
- helper::DynFuncHolder< decltype(bigvoice_AudioConverter_From) > AudioConverter_From
- helper::DynFuncHolder< decltype(bigvoice_AudioConverter_To) > AudioConverter_To

### 14.16.1 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.

### 14.16.2 Constructor & Destructor Documentation

#### 14.16.2.1 CDynLib()

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

### 14.16.3 Member Function Documentation

#### 14.16.3.1 IsOk()

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

**Returns**

true if OK.

false if not OK.

Definition at line 500 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.3.2 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 344 of file sound4.bigvoice.cl_dyn.hpp.

### 14.16.4 Member Data Documentation

#### 14.16.4.1 AddAudio

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

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

#### 14.16.4.2 AddPadAudio

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

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

#### 14.16.4.3 AudioConverter_From

```
helper::DynFuncHolder< decltype(bigvoice_AudioConverter_From ) > sound4::bigvoice::dyn::CDyn↩
Lib::AudioConverter_From
```

Definition at line 334 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.4.4 AudioConverter_To

```
helper::DynFuncHolder< decltype(bigvoice_AudioConverter_To ) > sound4::bigvoice::dyn::CDyn↩
Lib::AudioConverter_To
```

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

#### 14.16.4.5 AudioConvertFrom

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

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

#### 14.16.4.6 AudioConvertTo

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

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

#### 14.16.4.7 AudioMonoFromLiveStereo

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

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

#### 14.16.4.8 AudioMonoToLiveStereo

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

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

#### 14.16.4.9 DeleteClient

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

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

#### 14.16.4.10 ExitProcess

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

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

#### 14.16.4.11 FreeAudioConverter

helper::DynFuncHolder< decltype(bigvoice_FreeAudioConverter ) > sound4::bigvoice::dyn::CDyn↩
Lib::FreeAudioConverter

Definition at line 333 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.4.12 FreeBus

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

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

#### 14.16.4.13 FreeJsonAnswer

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

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

#### 14.16.4.14 FreeMetadataInfos

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

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

#### 14.16.4.15 FreeParameters

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

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

#### 14.16.4.16 FreeParameterValue

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

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

#### 14.16.4.17 GetAudio

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

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

#### 14.16.4.18 GetAudioConverter

```
helper::DynFuncHolder< decltype(bigvoice_GetAudioConverter ) > sound4::bigvoice::dyn::CDyn↩
Lib::GetAudioConverter
```

Definition at line 332 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.16.4.19 GetAudioInputCount

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

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

#### 14.16.4.20 GetAudioOutputCount

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

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

#### 14.16.4.21 GetBufferIn

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

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

#### 14.16.4.22 GetBufferOut

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

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

#### 14.16.4.23 GetBytesFromFormat

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

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

#### 14.16.4.24 GetChannelCount

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

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

#### 14.16.4.25 GetChunkSizeInFrames

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

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

#### 14.16.4.26 GetEstimatedDelay

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

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

#### 14.16.4.27 GetFormatFromName

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

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

#### 14.16.4.28 GetFormatName

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

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

#### 14.16.4.29 GetMaxPacketFrame

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

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

#### 14.16.4.30 GetMetadataInfos

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

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

#### 14.16.4.31 GetOutputCount

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

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

#### 14.16.4.32 GetParameter

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

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

#### 14.16.4.33 GetPossibleChunkSizeInFrames

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

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

#### 14.16.4.34 GetProcessChunkFrames

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

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

#### 14.16.4.35 GetSampleRate

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

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

#### 14.16.4.36 GetVersion

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

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

#### 14.16.4.37 InitProcess

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

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

#### 14.16.4.38 InitProcess2

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

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

#### 14.16.4.39 InitProcess3

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

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

#### 14.16.4.40 MonoToStereo

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

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

#### 14.16.4.41 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.

#### 14.16.4.42 NewBus

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

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

#### 14.16.4.43 NewClient

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

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

#### 14.16.4.44 NewParameters

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

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

#### 14.16.4.45 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.

#### 14.16.4.46 ProcessAudio

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

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

#### 14.16.4.47 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.

#### 14.16.4.48 ProcessJson

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

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

#### 14.16.4.49 SaveState

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

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

#### 14.16.4.50 SetInstanceBus

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

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

#### 14.16.4.51 SetInstanceTracing

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

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

#### 14.16.4.52 SetInstanceTracingProcessActivity

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

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

#### 14.16.4.53 SetLoggerCallback

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

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

#### 14.16.4.54 SetLogSeverity

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

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

#### 14.16.4.55 SetMetadata

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

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

#### 14.16.4.56 SetMetadataMulti

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

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

#### 14.16.4.57 SetParameter

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

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

#### 14.16.4.58 SetPresetManager

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

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

#### 14.16.4.59 StartUpdateThread

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

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

#### 14.16.4.60 StereoToMono

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

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

#### 14.16.4.61 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.

#### 14.16.4.62 StopUpdateThread

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

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

#### 14.16.4.63 TerminateProcess

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

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

#### 14.16.4.64 WaitUpdateThreadReady

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

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

#### 14.16.4.65 Webserver

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

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

#### 14.16.4.66 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.

#### 14.16.4.67 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.

#### 14.16.4.68 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.

#### 14.16.4.69 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.

#### 14.16.4.70 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.

#### 14.16.4.71 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.

#### 14.16.4.72 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

## 14.17 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)

## 14.17.1 Detailed Description

Definition at line 108 of file sound4.bigvoice.cl_dyn.hpp.

## 14.17.2 Constructor & Destructor Documentation

### 14.17.2.1 CDynLoader() [1/3]

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

### 14.17.2.2 CDynLoader() [2/3]

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

### 14.17.2.3 CDynLoader() [3/3]

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

### 14.17.2.4 ~CDynLoader()

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

Definition at line 218 of file sound4.bigvoice.cl_dyn.hpp.

## 14.17.3 Member Function Documentation

### 14.17.3.1 Close()

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

Definition at line 179 of file sound4.bigvoice.cl_dyn.hpp.

### 14.17.3.2 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.

### 14.17.3.3 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.

### 14.17.3.4 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.

### 14.17.3.5 IsOk()

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

Definition at line 126 of file sound4.bigvoice.cl_dyn.hpp.

### 14.17.3.6 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.

#### 14.17.3.7 operator bool()

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

Definition at line 123 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.17.3.8 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

## 14.18 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)

### 14.18.1 Detailed Description

Definition at line 155 of file sound4cl.hpp.

## 14.18.2 Constructor & Destructor Documentation

### 14.18.2.1 CDynLoader() [1/3]

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

### 14.18.2.2 CDynLoader() [2/3]

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

### 14.18.2.3 CDynLoader() [3/3]

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

### 14.18.2.4 ~CDynLoader()

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

Definition at line 265 of file sound4cl.hpp.

## 14.18.3 Member Function Documentation

### 14.18.3.1 Close()

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

Definition at line 226 of file sound4cl.hpp.

### 14.18.3.2 GetSymbol()

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

Definition at line 239 of file sound4cl.hpp.

### 14.18.3.3 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.

#### 14.18.3.4 GetThisLibraryPath()

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

Definition at line 251 of file sound4cl.hpp.

#### 14.18.3.5 IsOk()

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

Definition at line 173 of file sound4cl.hpp.

#### 14.18.3.6 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.

#### 14.18.3.7 operator bool()

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

Definition at line 170 of file sound4cl.hpp.

#### 14.18.3.8 operator=()

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

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

- sound4cl.hpp

## 14.19 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)

## 14.19.1 Detailed Description

Holds an instance of processing

Definition at line 478 of file sound4.bigvoice.cl.hpp.

### 14.19.2 Constructor & Destructor Documentation

#### 14.19.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 483 of file sound4.bigvoice.cl.hpp.

#### 14.19.2.2 CInstance() [2/2]

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

#### 14.19.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 494 of file sound4.bigvoice.cl.hpp.

### 14.19.3 Member Function Documentation

#### 14.19.3.1 ActivateAuxiliaryAudioID()

```
void sound4::bigvoice::CInstance::ActivateAuxiliaryAudioID (
            const std::string & id )  [inline]
```

Definition at line 759 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.2 AddAudio()

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

Definition at line 1080 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.3 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.

#### 14.19.3.4 GetAudio()

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

Definition at line 1086 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.5 GetAuxiliaryAudioOutputInfos()

```
std::vector< AuxiliaryOutputInfos > sound4::bigvoice::CInstance::GetAuxiliaryAudioOutputInfos
( )  [inline]
```

Get information on auxiliary audio outputs.

**Returns**

a list of AuxiliaryOutputInfos

Definition at line 740 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.6 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 901 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.7 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 919 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.8 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 638 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.9 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 883 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.10 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 716 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.11 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.

#### 14.19.3.12 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 830 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.13 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.

#### 14.19.3.14 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 1034 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.15 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 766 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.16 OnUpdateThreadStop()

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

Can be overloaded to call at end of UpdateThread in the thread

Definition at line 770 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.17 operator=()

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

#### 14.19.3.18 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 652 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.19 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 962 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.20 ProcessAudio()

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

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 928 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.21 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 944 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.22 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 1050 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.23 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.

#### 14.19.3.24 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 1064 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.25 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 1073 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.26 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 668 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.27 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 686 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.28 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.

#### 14.19.3.29 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.

#### 14.19.3.30 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 822 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.31 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 786 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.32 Stop()

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

Stops the instance, its web server and its update thread.

Definition at line 860 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.33 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 811 of file sound4.bigvoice.cl.hpp.

#### 14.19.3.34 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 851 of file sound4.bigvoice.cl.hpp.

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

- sound4.bigvoice.cl.hpp

# 14.20 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)

## 14.20.1 Detailed Description

Holds an instance of processing

Definition at line 957 of file sound4.bigvoice.cl_dyn.hpp.

## 14.20.2 Constructor & Destructor Documentation

### 14.20.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 962 of file sound4.bigvoice.cl_dyn.hpp.

### 14.20.2.2 CInstance() [2/2]

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

### 14.20.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 974 of file sound4.bigvoice.cl_dyn.hpp.

## 14.20.3 Member Function Documentation

### 14.20.3.1 AddAudio()

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

Definition at line 1561 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.2 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 1079 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.3 GetAudio()

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

Definition at line 1567 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.4 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 1376 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.5 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 1394 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.6 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 1126 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.7 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 1358 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.8 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 1215 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.9 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 1003 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.10 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 1304 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.11 IsOk()

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

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

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

#### 14.20.3.12 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 1511 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.13 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 1239 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.14 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 1243 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.15 operator=()

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

#### 14.20.3.16 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 1140 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.17 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 1437 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.18 ProcessAudio()

```
void sound4::bigvoice::dyn::CInstance::ProcessAudio ( )  [inline]
```

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 1403 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.19 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 1419 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.20 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 1527 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.21 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 1020 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.22 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 1541 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.23 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 1552 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.24 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 1158 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.25 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 1179 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.26 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 992 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.27 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 1038 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.28 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 1295 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.29 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 1259 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.30 Stop()

```
void sound4::bigvoice::dyn::CInstance::Stop ( )  [inline]
```

Stops the instance, its web server and its update thread.

Definition at line 1335 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.31 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 1284 of file sound4.bigvoice.cl_dyn.hpp.

#### 14.20.3.32 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 1326 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.21 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)

## 14.21.1 Detailed Description

Holds an instance of processing

Definition at line 918 of file sound4cl.hpp.

## 14.21.2 Constructor & Destructor Documentation

### 14.21.2.1 CInstance() [1/3]

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

Creates the instance, with default parameters

**Parameters**

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

Definition at line 925 of file sound4cl.hpp.

### 14.21.2.2 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 940 of file sound4cl.hpp.

### 14.21.2.3 CInstance() [3/3]

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

### 14.21.2.4 ~CInstance()

```
virtual sound4::CInstance::~CInstance ( )  [inline], [virtual]
```

Destructor, stopping any running tasks

Definition at line 952 of file sound4cl.hpp.

### 14.21.3 Member Function Documentation

#### 14.21.3.1 AddAudio()

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

Definition at line 1542 of file sound4cl.hpp.

#### 14.21.3.2 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 1056 of file sound4cl.hpp.

#### 14.21.3.3 GetAudio()

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

Definition at line 1548 of file sound4cl.hpp.

#### 14.21.3.4 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 1359 of file sound4cl.hpp.

#### 14.21.3.5 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 1376 of file sound4cl.hpp.

#### 14.21.3.6 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 1106 of file sound4cl.hpp.

#### 14.21.3.7 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 1341 of file sound4cl.hpp.

#### 14.21.3.8 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 1194 of file sound4cl.hpp.

#### 14.21.3.9 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 981 of file sound4cl.hpp.

#### 14.21.3.10 GetWebServerAppHealth()

```
void sound4::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 1287 of file sound4cl.hpp.

#### 14.21.3.11 IsOk()

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

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

Definition at line 960 of file sound4cl.hpp.

#### 14.21.3.12 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 1492 of file sound4cl.hpp.

#### 14.21.3.13 OnUpdateThreadStart()

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

Can be overloaded, called at start of UpdateThread in the thread

Definition at line 1218 of file sound4cl.hpp.

#### 14.21.3.14 OnUpdateThreadStop()

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

Can be overloaded to call at end of UpdateThread in the thread

Definition at line 1222 of file sound4cl.hpp.

#### 14.21.3.15 operator=()

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

#### 14.21.3.16 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 1120 of file sound4cl.hpp.

#### 14.21.3.17 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 1418 of file sound4cl.hpp.

#### 14.21.3.18 ProcessAudio()

```
void sound4::CInstance::ProcessAudio ( )  [inline]
```

Process one audio chunk, from GetBufferIn() to GetBufferOut()

Definition at line 1384 of file sound4cl.hpp.

#### 14.21.3.19 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 1400 of file sound4cl.hpp.

#### 14.21.3.20 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 1508 of file sound4cl.hpp.

#### 14.21.3.21 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 998 of file sound4cl.hpp.

#### 14.21.3.22 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 1522 of file sound4cl.hpp.

#### 14.21.3.23 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 1533 of file sound4cl.hpp.

#### 14.21.3.24 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 1138 of file sound4cl.hpp.

#### 14.21.3.25 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 1158 of file sound4cl.hpp.

#### 14.21.3.26 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 970 of file sound4cl.hpp.

#### 14.21.3.27 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 1015 of file sound4cl.hpp.

#### 14.21.3.28 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 1278 of file sound4cl.hpp.

#### 14.21.3.29 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 1238 of file sound4cl.hpp.

#### 14.21.3.30 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 1318 of file sound4cl.hpp.

#### 14.21.3.31 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 1266 of file sound4cl.hpp.

#### 14.21.3.32 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 1307 of file sound4cl.hpp.

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

- sound4cl.hpp

## 14.22 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

### 14.22.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 458 of file sound4.bigvoice.cl.hpp.

### 14.22.2 Constructor & Destructor Documentation

#### 14.22.2.1 CPresetLoader()

```
sound4::bigvoice::CPresetLoader::CPresetLoader ( )  [default]
```

#### 14.22.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.22.3 Member Function Documentation

#### 14.22.3.1 Exists()

```
virtual bool sound4::bigvoice::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

#### 14.22.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::bigvoice::CPresetLoader::GetAll ( )
[pure virtual]
```

#### 14.22.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

#### 14.22.3.4 Read()

```
virtual std::string sound4::bigvoice::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

#### 14.22.3.5 Remove()

```
virtual bool sound4::bigvoice::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

#### 14.22.3.6 Rename()

```
virtual bool sound4::bigvoice::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

#### 14.22.3.7 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:

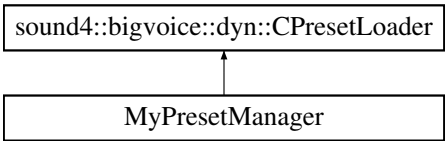
- sound4.bigvoice.cl.hpp

## 14.23 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

### 14.23.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

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

### 14.23.2 Constructor & Destructor Documentation

#### 14.23.2.1 CPresetLoader()

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

#### 14.23.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::dyn::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.23.3 Member Function Documentation

#### 14.23.3.1 Exists()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.23.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::bigvoice::dyn::CPresetLoader::GetAll ( )
[pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.23.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.23.3.4 Read()

```
virtual std::string sound4::bigvoice::dyn::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.23.3.5 Remove()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.23.3.6 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.

#### 14.23.3.7 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:

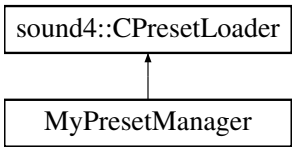
- sound4.bigvoice.cl_dyn.hpp

## 14.24 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

### 14.24.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 895 of file sound4cl.hpp.

### 14.24.2 Constructor & Destructor Documentation

#### 14.24.2.1 CPresetLoader()

```
sound4::CPresetLoader::CPresetLoader ( )  [default]
```

#### 14.24.2.2 ~CPresetLoader()

```
virtual sound4::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

### 14.24.3 Member Function Documentation

#### 14.24.3.1 Exists()

```
virtual bool sound4::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::CPresetLoader::GetAll ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.3 IsReadOnly()

```
virtual bool sound4::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.4 Read()

```
virtual std::string sound4::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.5 Remove()

```
virtual bool sound4::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.6 Rename()

```
virtual bool sound4::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

#### 14.24.3.7 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

## 14.25 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
- class CAudioConverter

### 14.25.1 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.

## 14.25.2 Constructor & Destructor Documentation

### 14.25.2.1 CProcessor()

```
sound4::CProcessor::CProcessor ( )  [default]
```

## 14.25.3 Member Function Documentation

### 14.25.3.1 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 798 of file sound4cl.hpp.

### 14.25.3.2 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 808 of file sound4cl.hpp.

### 14.25.3.3 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 845 of file sound4cl.hpp.

#### 14.25.3.4 GetChannelCount()

```
unsigned int sound4::CProcessor::GetChannelCount ( )  [inline]
```

Get the number of channels needed by this process

**Returns**

The process channels count

Definition at line 788 of file sound4cl.hpp.

#### 14.25.3.5 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 759 of file sound4cl.hpp.

#### 14.25.3.6 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 836 of file sound4cl.hpp.

#### 14.25.3.7 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 827 of file sound4cl.hpp.

#### 14.25.3.8 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 770 of file sound4cl.hpp.

#### 14.25.3.9 GetPrefixSymbol()

```
template<typename T >
helper::DynFuncHolder< T > sound4::CProcessor::GetPrefixSymbol (
            const std::string & name )  [inline], [protected]
```

Definition at line 416 of file sound4cl.hpp.

#### 14.25.3.10 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 818 of file sound4cl.hpp.

#### 14.25.3.11 GetVersion()

```
std::string sound4::CProcessor::GetVersion ( )  [inline]
```

Get the process version string

**Returns**

version string

Definition at line 746 of file sound4cl.hpp.

#### 14.25.3.12 IsOk()

```
bool sound4::CProcessor::IsOk ( ) const  [inline]
```

**Returns**

true if OK.

false if not OK.

Definition at line 633 of file sound4cl.hpp.

#### 14.25.3.13 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 431 of file sound4cl.hpp.

#### 14.25.3.14 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 857 of file sound4cl.hpp.

#### 14.25.3.15 SetLogSeverity()

```
void sound4::CProcessor::SetLogSeverity (
            LogSeverity severity )  [inline]
```

Sets the log severity (global)

Definition at line 851 of file sound4cl.hpp.

### 14.25.4 Friends And Related Symbol Documentation

#### 14.25.4.1 CAudioConverter

```
friend class CAudioConverter  [friend]
```

Definition at line 317 of file sound4cl.hpp.

#### 14.25.4.2 CBus

```
friend class CBus  [friend]
```

Definition at line 315 of file sound4cl.hpp.

#### 14.25.4.3 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

## 14.26 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

### 14.26.1 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.

### 14.26.2 Constructor & Destructor Documentation

#### 14.26.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::DynFuncHolder ( )  [default]
```

#### 14.26.2.2 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.

### 14.26.3 Member Function Documentation

#### 14.26.3.1 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.

#### 14.26.3.2 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.

#### 14.26.3.3 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

## 14.27 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

### 14.27.1 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.

### 14.27.2 Constructor & Destructor Documentation

#### 14.27.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::helper::DynFuncHolder< T >::DynFuncHolder ( )  [default]
```

#### 14.27.2.2 DynFuncHolder() [2/2]

```
template<typename T >
sound4::helper::DynFuncHolder< T >::DynFuncHolder (
            DynFunc_t a_ptr )  [inline]
```

Definition at line 145 of file sound4cl.hpp.

### 14.27.3 Member Function Documentation

#### 14.27.3.1 IsOk()

```
template<typename T >
bool sound4::helper::DynFuncHolder< T >::IsOk ( ) const  [inline]
```

Definition at line 147 of file sound4cl.hpp.

#### 14.27.3.2 operator bool()

```
template<typename T >
sound4::helper::DynFuncHolder< T >::operator bool ( ) const  [inline]
```

Definition at line 146 of file sound4cl.hpp.

#### 14.27.3.3 operator T*()

```
template<typename T >
sound4::helper::DynFuncHolder< T >::operator T* ( ) const  [inline]
```

Definition at line 148 of file sound4cl.hpp.

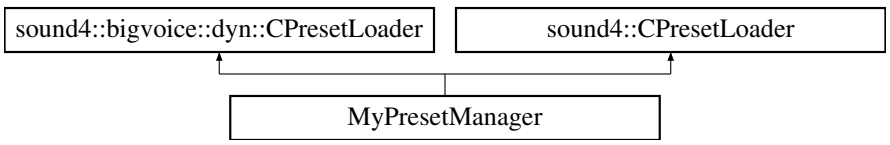
The documentation for this class was generated from the following file:

- sound4cl.hpp

# 14.28 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

## 14.28.1 Detailed Description

Definition at line 120 of file sound4.bigvoice.cl-pipe-dyn.cpp.

## 14.28.2 Constructor & Destructor Documentation

### 14.28.2.1 MyPresetManager() [1/2]

```
MyPresetManager::MyPresetManager ( )  [default]
```

### 14.28.2.2 ~MyPresetManager() [1/2]

```
virtual MyPresetManager::~MyPresetManager ( )  [virtual], [default]
```

### 14.28.2.3 MyPresetManager() [2/2]

```
MyPresetManager::MyPresetManager ( )  [default]
```

### 14.28.2.4 ~MyPresetManager() [2/2]

```
virtual MyPresetManager::~MyPresetManager ( )  [virtual], [default]
```

## 14.28.3 Member Function Documentation

### 14.28.3.1 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.

### 14.28.3.2 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.

### 14.28.3.3 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.

#### 14.28.3.4 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.

#### 14.28.3.5 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.

#### 14.28.3.6 IsReadOnly() [2/2]

```
virtual bool MyPresetManager::IsReadOnly ( )  [inline], [override], [virtual]
```

Implements sound4::bigvoice::dyn::CPresetLoader.

Definition at line 121 of file sound4cl-pipe.cpp.

#### 14.28.3.7 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.

#### 14.28.3.8 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.

#### 14.28.3.9 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.

#### 14.28.3.10 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.

#### 14.28.3.11 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.

#### 14.28.3.12 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.

#### 14.28.3.13 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.

#### 14.28.3.14 SetupPath() [2/2]

```
bool MyPresetManager::SetupPath (
            const std::filesystem::path & a_preset_path )  [inline]
```

Definition at line 109 of file sound4cl-pipe.cpp.

#### 14.28.3.15 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.

#### 14.28.3.16 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

## 14.29 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

### 14.29.1 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 579 of file sound4.bigvoice.cl_dyn.hpp.

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

- sound4.bigvoice.cl_dyn.hpp

## 14.30 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

### 14.30.1 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

## 14.31 sound4::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

`#include <sound4cl.hpp>`

### 14.31.1 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

## 14.32 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.*

### 14.32.1 Detailed Description

Structure holding information on an auxiliary output

Definition at line 810 of file sound4.bigvoice.cl.h.

### 14.32.2 Member Data Documentation

#### 14.32.2.1 buffer

```
const float* SOUND4_AuxiliaryOutputInfos::buffer
```

Definition at line 815 of file sound4.bigvoice.cl.h.

#### 14.32.2.2 channel_count

```
unsigned int SOUND4_AuxiliaryOutputInfos::channel_count
```

Definition at line 814 of file sound4.bigvoice.cl.h.

#### 14.32.2.3 description

```
char* SOUND4_AuxiliaryOutputInfos::description
```

Definition at line 812 of file sound4.bigvoice.cl.h.

#### 14.32.2.4 id

```
char* SOUND4_AuxiliaryOutputInfos::id
```

Definition at line 811 of file sound4.bigvoice.cl.h.

#### 14.32.2.5 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

## 14.33 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.*

## 14.33.1 Detailed Description

Information on this processor.

**See also**

SOUND4_GetProcessInfo

Definition at line 77 of file sound4.bigvoice.cl.h.

## 14.33.2 Member Data Documentation

### 14.33.2.1 name

```
const char* SOUND4_ProcessInfo::name
```

Definition at line 79 of file sound4.bigvoice.cl.h.

### 14.33.2.2 prefix

```
const char* SOUND4_ProcessInfo::prefix
```

Definition at line 80 of file sound4.bigvoice.cl.h.

### 14.33.2.3 shortname

```
const char* SOUND4_ProcessInfo::shortname
```

Definition at line 81 of file sound4.bigvoice.cl.h.

### 14.33.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

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

### 14.33.2.5 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 '"«(const char*)store_path.u8string().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 '"«(const char*)store_path.u8string().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 " « (const
    char*)store_path.u8string().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 '"«(const char*)fs_store_path.u8string().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 '"«(const char*)store_path.u8string().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 " « (const
      char*)store_path.u8string().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 }
- enum bigvoice_ConverterTargetMode {
  ConverterTargetMode_AsInput = 0 , ConverterTargetMode_Mono , ConverterTargetMode_Stereo ,
  ConverterTargetMode_StereoToMono ,
  ConverterTargetMode_MonoToStereo }

### 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_CAudioConverter * bigvoice_GetAudioConverter (enum bigvoice_SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, enum bigvoice_ConverterTargetMode target_mode)
- void bigvoice_FreeAudioConverter (struct bigvoice_CAudioConverter *converter)
- void bigvoice_AudioConverter_From (struct bigvoice_CAudioConverter *converter, const uint8_t *payload, float *output)
- void bigvoice_AudioConverter_To (struct bigvoice_CAudioConverter *converter, 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
01078 struct bigvoice_CAudioConverter;
01079
```

```
01083 enum bigvoice_ConverterTargetMode {
01084     ConverterTargetMode_AsInput = 0,
01085     ConverterTargetMode_Mono,
01086     ConverterTargetMode_Stereo,
01087     ConverterTargetMode_StereoToMono,
01088     ConverterTargetMode_MonoToStereo,
01089 };
01090
01112 BIGVOICE_DECL struct bigvoice_CAudioConverter* bigvoice_GetAudioConverter(enum bigvoice_SampleFormat
      fmt, size_t nFrame
01113     , size_t payl_ch, size_t payl_ch_offset
01114     , enum bigvoice_ConverterTargetMode target_mode);
01115
01119 BIGVOICE_DECL void bigvoice_FreeAudioConverter(struct bigvoice_CAudioConverter *converter);
01120
01130 BIGVOICE_DECL void bigvoice_AudioConverter_From(struct bigvoice_CAudioConverter *converter, const
      uint8_t *payload, float *output);
01131
01141 BIGVOICE_DECL void bigvoice_AudioConverter_To(struct bigvoice_CAudioConverter *converter, const float
      *input, uint8_t *payload);
01142
01145 // ------------------------------------------------------------------
01152 // ------------------------------------------------------------------
01156 struct bigvoice_CClientInstance;
01157
01166 BIGVOICE_DECL struct bigvoice_CClientInstance *bigvoice_NewClient(struct bigvoice_CInstance
      *instance);
01167
01173 BIGVOICE_DECL void bigvoice_DeleteClient(struct bigvoice_CClientInstance *client);
01174
01188 BIGVOICE_DECL const char *bigvoice_ProcessJson(struct bigvoice_CClientInstance *client, const char
      *json_str, int *need_save);
01189
01195 BIGVOICE_DECL void bigvoice_FreeJsonAnswer(const char *json_str);
01196
01208 BIGVOICE_DECL int bigvoice_SaveState(struct bigvoice_CInstance *instance);
01209
01212 #if BIGVOICE_HAS_WEBSERVER
01213 // ------------------------------------------------------------------
01231 // ------------------------------------------------------------------
01239 BIGVOICE_DECL uint64_t bigvoice_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port);
01240
01241 // ------------------------------------------------------------------
01261 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);
01262
01280 BIGVOICE_DECL uint64_t bigvoice_Webserver(unsigned int listenport, unsigned int listenport_secure,
      struct bigvoice_CInstance *instance);
01281
01291 BIGVOICE_DECL int bigvoice_Webserver_Stop(uint64_t id, int timeout_ms);
01292
01301 BIGVOICE_DECL int bigvoice_Webserver_Status(uint64_t id);
01302
01313 BIGVOICE_DECL void bigvoice_Webserver_SetAppHealth(struct bigvoice_CInstance *instance, int httpcode,
      const char *contenttype, const char *content);
01314
01323 BIGVOICE_DECL void bigvoice_Webserver_GetAppHealth(struct bigvoice_CInstance *instance, int* httpcode,
      char** contenttype, char** content);
01324
01330 BIGVOICE_DECL void bigvoice_Webserver_FreeString(char *str);
01331
01336 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
01337
01341 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
01345 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
01346
01348 #endif // BIGVOICE_HAS_WEBSERVER
01349
01350 #ifdef __cplusplus
01351 } // extern "C"
01352 #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
- class sound4::bigvoice::CAudioConverter

  *Audio converter helper.*

### 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 1198 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 1203 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 1213 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 1184 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 1189 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 1220 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 1225 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 1193 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                     #if (__cplusplus<202002L)
00603                     l_save_path = save_path.u8string();
00604                     #else // C++20
00605                     l_save_path = (const char*)save_path.u8string().c_str();
00606                     #endif // C++20
```

```
00607                 }
00608                 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);
00609                 if (!instance) return false;
00610                 if (!init_metadata.empty()) {
00611                     for (auto&& [key,value]: init_metadata) {
00612                         bigvoice_SetMetadata(instance, key.c_str(), value.c_str());
00613                     }
00614                     init_metadata.clear();
00615                 }
00616                 update_thread = std::thread([this, json_port](){
00617                     OnUpdateThreadStart();
00618                     bigvoice_StartUpdateThread(instance, json_port);
00619                     OnUpdateThreadStop();
00620                 });
00621                 int timeout=5000;
00622                 #ifdef DEBUG
00623                     timeout=60*1000; // For debugger pauses
00624                 #endif
00625                 if (bigvoice_WaitUpdateThreadReady(instance, timeout)<0) {
00626                     return false;
00627                 }
00628                 return true;
00629             }
00630
00638         unsigned int GetChunkFrames() {
00639             if (instance) {
00640                 return bigvoice_GetProcessChunkFrames(instance);
00641             }
00642             return 0;
00643         }
00644
00652         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
00653             {
00654                 bigvoice_PresetManager_InformChange(instance, relative_path.c_str(),
      (bigvoice_PresetChange_Kind) change_kind);
00655             }
00656
00668         void SetMetadata(const std::string& key, const char* value)
00669             {
00670                 if (instance) {
00671                     bigvoice_SetMetadata(instance, key.c_str(), value);
00672                 } else {
00673                     init_metadata.push_back( {key, value} );
00674                 }
00675             }
00676
00686         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
00687             {
00688                 if (instance) {
00689                     typedef const char* pchar;
00690                     pchar* keyvalue=new pchar[2*list.size()+1];
00691                     size_t n=0;
00692                     for (auto&& [key,value]: list) {
00693                         keyvalue[2*n+0]=key.c_str();
00694                         keyvalue[2*n+1]=value;
00695                         n++;
00696                     }
00697                     keyvalue[2*n+0]=nullptr;
00698
00699                     bigvoice_SetMetadataMulti(instance, keyvalue);
00700
00701                     delete[] keyvalue;
00702                 } else {
00703                     for (auto&& [key,value]: list) {
00704                         if (value) {
00705                             init_metadata.push_back( {key, value} );
00706                         }
00707                     }
00708                 }
00709             }
00710
00716         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
00717             {
00718                 std::vector< std::tuple<std::string,std::string> > values;
00719                 const char** c_values = bigvoice_GetMetadataInfos(instance);
00720                 if (c_values) {
00721                     for (const char** c_value=c_values; *c_value; ) {
00722                         std::string key(*c_value);
00723                         c_value++;
00724                         if (*c_value) {
00725                             std::string descr(*c_value);
00726                             c_value++;
00727                             values.push_back( {key,descr} );
00728                         }
00729                     }
```

```
00730                     bigvoice_FreeMetadataInfos(instance, c_values);
00731                 }
00732                 return values;
00733             }
00734
00740         std::vector<AuxiliaryOutputInfos> GetAuxiliaryAudioOutputInfos()
00741             {
00742                 std::vector<AuxiliaryOutputInfos> values;
00743                 SOUND4_AuxiliaryOutputInfos** c_values =
      bigvoice_GetAuxiliaryAudioOutputInfos(instance);
00744                 if (c_values) {
00745                     for (SOUND4_AuxiliaryOutputInfos** c_value=c_values; *c_value; c_value++) {
00746                         auto c_info=*c_value;
00747                         values.emplace_back( AuxiliaryOutputInfos{
00748                             .id            = c_info->id          ,
00749                             .description   = c_info->description  ,
00750                             .sample_rate   = c_info->sample_rate  ,
00751                             .channel_count = c_info->channel_count,
00752                             .buffer        = c_info->buffer       ,
00753                         });
00754                     }
00755                     bigvoice_FreeAuxiliaryAudioInfos(c_values);
00756                 }
00757                 return values;
00758             }
00759         void ActivateAuxiliaryAudioID(const std::string& id)
00760             {
00761                 bigvoice_ActivateAuxiliaryAudioID(instance, id.c_str());
00762             }
00766         virtual void OnUpdateThreadStart() {}
00770         virtual void OnUpdateThreadStop() {}
00771
00772 #if BIGVOICE_HAS_WEBSERVER
00786         bool StartWebServer(int http_port, int https_port=0)
00787             {
00788                 assert(instance);
00789                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00790                 webserver = bigvoice_Webserver(http_port,https_port,instance);
00791                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00792                     return false;
00793                 }
00794                 int web_status=bigvoice_Webserver_Status(webserver);
00795                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00796                     return false;
00797                 }
00798                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00799                     return false;
00800                 }
00801                 return true;
00802             }
00803
00811             void StopWebServer(int timeout_ms = 1000)
00812             {
00813                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00814                     bigvoice_Webserver_Stop(webserver, timeout_ms);
00815                     webserver = SOUND4_INVALID_WEBSERVER_ID;
00816                 }
00817             }
00818
00822         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
00823             {
00824                 assert(instance);
00825                 bigvoice_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
00826             }
00830         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
00831             {
00832                 assert(instance);
00833                 char* c_contenttype=nullptr;
00834                 char* c_content=nullptr;
00835                 bigvoice_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
00836                 contenttype=c_contenttype;
00837                 content=c_content;
00838                 bigvoice_Webserver_FreeString(c_contenttype);
00839                 bigvoice_Webserver_FreeString(c_content);
00840         }
00841 #endif // BIGVOICE_HAS_WEBSERVER
00842
00851         int Terminate()
00852             {
00853                 if (!instance) return 0;
00854                 return bigvoice_TerminateProcess(instance);
00855             }
00856
00860         void Stop()
00861             {
```

```
00862                 if (!instance) return;
00863 #if BIGVOICE_HAS_WEBSERVER
00864                 StopWebServer();
00865 #endif // BIGVOICE_HAS_WEBSERVER
00866                 if (update_thread.joinable()) {
00867                     bigvoice_StopUpdateThread(instance); // this will exit the update thread
00868                     update_thread.join();
00869                 }
00870                 bigvoice_ExitProcess(instance);
00871                 instance=nullptr;
00872             }
00873
00883         unsigned int GetEstimatedDelay()
00884             {
00885                 assert(instance);
00886                 return bigvoice_GetEstimatedDelay(instance);
00887             }
00888
00901         std::array<float,InputSampleSize>& GetBufferIn()
00902             {
00903                 assert(instance);
00904                 float *buf=bigvoice_GetBufferIn(instance);
00905                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
00906
00907             }
00919         std::array<float,OutputSampleSize>& GetBufferOut()
00920             {
00921                 assert(instance);
00922                 float *buf= bigvoice_GetBufferOut(instance);
00923                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00924             }
00928         void ProcessAudio()
00929             {
00930                 assert(instance);
00931                 bigvoice_ProcessAudio(instance, bigvoice_GetBufferIn(instance),
    bigvoice_GetBufferOut(instance));
00932             }
00944         void ProcessAudio_Planar(float const * const *input, float * const *output)
00945             {
00946                 assert(instance);
00947                 bigvoice_ProcessAudio_Planar(instance, input, output);
00948             }
00949
00961             template<typename T>
00962         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00963             {
00964                 assert(instance);
00965                 std::vector<T> output;
00966                 unsigned int out_offset=0;
00967                 unsigned int in_offset=0;
00968                 unsigned int todo = input.size();
00969                 while (todo>0) {
00970                     unsigned int left = AddAudio(&input[in_offset], todo);
00971                     unsigned int out_avail = bigvoice_GetOutputCount(instance);
00972                     output.resize(out_offset + out_avail);
00973                     GetAudio(&output[out_offset], out_avail);
00974                     out_offset+=out_avail;
00975                     in_offset += todo-left;
00976                     todo=left;
00977                 }
00978                 return output;
00979             }
00980
00986         class CClient {
00987             public:
00988                 CClient(bigvoice_CInstance* instance)
00989                     {
00990                         assert(instance);
00991                         client=bigvoice_NewClient(instance);
00992                         if (!client) throw std::bad_alloc();
00993                     }
00994                 // non-copyable
00995                 CClient( const CInstance& ) = delete;
00996                 CClient& operator=( const CInstance& ) = delete;
00997                 ~CClient()
00998                     {
00999                         if (client) {
01000                             bigvoice_DeleteClient(client);
01001                             client=nullptr;
01002                         }
01003                     }
01011                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01012                     {
01013                         assert(client);
01014                         int need_save=0;
01015                         const char *canswer = bigvoice_ProcessJson(client, request.c_str(),
    &need_save);
```

```
01016                         if (!canswer) return {};
01017                         std::string answer(canswer);
01018                         bigvoice_FreeJsonAnswer (canswer);
01019                         if (NeedSave) {
01020                             *NeedSave=(need_save!=0);
01021                         }
01022                         return answer;
01023                     }
01024             private:
01025                 bigvoice_CClientInstance *client = nullptr;
01026         };
01034         std::shared_ptr<CClient> NewClient()
01035             {
01036                 assert(instance);
01037                 return std::make_shared<CClient>(instance);
01038             }
01039
01050         bool SaveState() {
01051             assert(instance);
01052             if (bigvoice_SaveState(instance)!=0) {
01053                 return false;
01054             }
01055             return true;
01056         }
01057 #ifdef _WIN32
01064         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01065             bigvoice_SetInstanceTracing(params, tracing_provider, activity_guid);
01066         }
01073         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01074             assert(instance);
01075             bigvoice_SetInstanceTracingProcessActivity(instance, activity_guid);
01076         }
01077 #endif // _WIN32
01078     protected:
01079             template<typename T>
01080         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01081             {
01082                 assert(instance);
01083                 return bigvoice_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01084             }
01085             template<typename T>
01086         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01087             {
01088                 assert(instance);
01089                 return bigvoice_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01090             }
01091
01092     private:
01093         bigvoice_CParameters* params = nullptr;
01094         bigvoice_CInstance* instance = nullptr;
01095         std::thread update_thread;
01096         std::vector< std::pair<std::string,std::string> > init_metadata;
01097 #if BIGVOICE_HAS_WEBSERVER
01098         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01099 #endif // BIGVOICE_HAS_WEBSERVER
01100     };
01101
01108     class CAudioConverter {
01109         public:
01113             CAudioConverter() = default;
01114
01118             virtual ~CAudioConverter()
01119                 {
01120                     Delete();
01121                 }
01122
01138             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
      bigvoice_ConverterTargetMode target_mode)
01139                 {
01140                     Delete();
01141                     m_cvt = bigvoice_GetAudioConverter((bigvoice_SampleFormat)fmt, nFrame, payl_ch,
      payl_ch_offset, target_mode);
01142                     return IsOK();
01143                 }
01144             void Delete()
01145                 {
01146                     if (m_cvt) {
01147                         bigvoice_FreeAudioConverter(m_cvt);
01148                     }
01149                     m_cvt = nullptr;
01150                 }
01151             bool IsOK() const { return (m_cvt!=nullptr); }
01152
01161             void ConvertFrom(const uint8_t *payload, float *output)
01162                 { bigvoice_AudioConverter_From(m_cvt, payload, output); }
```

```
01163
01172             void ConvertTo(const float *input, uint8_t *payload)
01173                 { bigvoice_AudioConverter_To(m_cvt, input, payload); }
01174
01175         private:
01176             bigvoice_CAudioConverter *m_cvt = nullptr;
01177     };
01178
01179 }; // namespace bigvoice
01180 }; // namespace sound4
01181
01182 // bridge C callbacks to CPresetLoader
01183 extern "C" {
01184     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
01185         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01186         std::string content=preset_loader->Read(filename);
01187         return strdup(content.c_str());
01188     }
01189     static void bigvoice_custom_reader_free(char *content, void* userdata) {
01190         if (!content) return;
01191         free(content);
01192     }
01193     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01194         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01195         auto res=preset_loader->Write(filename,content);
01196         return res?0:-1;
01197     }
01198     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
01199         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01200         auto res=preset_loader->Exists(filename);
01201         return res?0:-1;
01202     }
01203     static fs_char** bigvoice_custom_getall(void* userdata) {
01204         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01205         auto res=preset_loader->GetAll();
01206         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01207         for (size_t n=0;n<res.size();n++) {
01208             all[n]=fs_strdup(res[n].c_str());
01209         }
01210         all[res.size()]=nullptr;
01211         return all;
01212     }
01213     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
01214         if (!all) return;
01215         for (fs_char** one=all;*one;one++) {
01216             free(*one);
01217         }
01218         free(all);
01219     }
01220     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
01221         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01222         auto res=preset_loader->Remove(filename);
01223         return res?0:-1;
01224     }
01225     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01226         sound4::bigvoice::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
01227         auto res=preset_loader->Rename(from,to);
01228         return res?0:-1;
01229     }
01230
01231 };
```

## 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
- class sound4::bigvoice::dyn::CAudioConverter
  *Audio converter helper.*

### 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 1687 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 1692 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 1702 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 1673 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 1678 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 1709 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 1714 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 1682 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
00331         // added 2025-05-28
00332         helper::DynFuncHolder< decltype(bigvoice_GetAudioConverter      ) > GetAudioConverter      ;
00333         helper::DynFuncHolder< decltype(bigvoice_FreeAudioConverter     ) > FreeAudioConverter     ;
00334         helper::DynFuncHolder< decltype(bigvoice_AudioConverter_From    ) > AudioConverter_From    ;
00335         helper::DynFuncHolder< decltype(bigvoice_AudioConverter_To      ) > AudioConverter_To      ;
00336
00344         bool Load(const std::filesystem::path& path = {}) {
00345             std::string filename;
00346             #if defined(__APPLE__)
00347                 filename="libsound4.bigvoice.cl.dylib";
00348             #elif defined(__unix__)
00349                 filename="libsound4.bigvoice.cl.so";
00350             #else
00351                 filename="sound4.bigvoice.cl.dll";
00352             #endif
00353             m_lib.Close();
00354             // If a path is given, use it directly and do not try other path
00355             if (!path.empty() && !m_lib.Load(path / filename)) {
00356                 return false;
00357             } else if (!m_lib.IsOk()) {
00358                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00359                 #ifndef _WIN32
00360                 // Linux: if in a bin directory, try ../lib first
00361                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00362                     auto libdir = thisdir.parent_path() / "lib";
00363                     m_lib.Load(libdir / filename);
00364                 }
00365                 #endif // !_WIN32
00366                 if (!m_lib.IsOk()) {
00367                     // Search in the same directory this code is in
00368                     m_lib.Load(thisdir / filename);
00369                 }
00370                 if (!m_lib.IsOk()) {
00371                     // Try current path
00372                     std::error_code ec;
00373                     auto p = std::filesystem::current_path(ec);
00374                     if (!ec) {
00375                         m_lib.Load(p / filename);
00376                     }
00377                 }
00378                 if (!m_lib.IsOk()) {
00379                     return false;
00380                 }
00381             }
00382             // Load all C functions
00383             try {
00384                 GetVersion              = m_lib.GetSymbol< decltype(bigvoice_GetVersion             )
    >("bigvoice_GetVersion"             );
00385                 GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(bigvoice_GetChunkSizeInFrames   )
    >("bigvoice_GetChunkSizeInFrames"   );
00386                 GetChannelCount         = m_lib.GetSymbol< decltype(bigvoice_GetChannelCount        )
    >("bigvoice_GetChannelCount"        );
00387                 GetAudioInputCount      = m_lib.GetSymbol< decltype(bigvoice_GetAudioInputCount     )
    >("bigvoice_GetAudioInputCount"     );
00388                 GetAudioOutputCount     = m_lib.GetSymbol< decltype(bigvoice_GetAudioOutputCount    )
    >("bigvoice_GetAudioOutputCount"    );
00389                 GetSampleRate           = m_lib.GetSymbol< decltype(bigvoice_GetSampleRate          )
    >("bigvoice_GetSampleRate"          );
00390                 SetLoggerCallback       = m_lib.GetSymbol< decltype(bigvoice_SetLoggerCallback      )
```

```
    >("bigvoice_SetLoggerCallback"      );
00391               SetLogSeverity          = m_lib.GetSymbol< decltype(bigvoice_SetLogSeverity         )
    >("bigvoice_SetLogSeverity"         );
00392               NewParameters           = m_lib.GetSymbol< decltype(bigvoice_NewParameters          )
    >("bigvoice_NewParameters"          );
00393               FreeParameters          = m_lib.GetSymbol< decltype(bigvoice_FreeParameters         )
    >("bigvoice_FreeParameters"         );
00394               SetParameter            = m_lib.GetSymbol< decltype(bigvoice_SetParameter           )
    >("bigvoice_SetParameter"           );
00395               GetParameter            = m_lib.GetSymbol< decltype(bigvoice_GetParameter           )
    >("bigvoice_GetParameter"           );
00396               FreeParameterValue      = m_lib.GetSymbol< decltype(bigvoice_FreeParameterValue     )
    >("bigvoice_FreeParameterValue"     );
00397               InitProcess             = m_lib.GetSymbol< decltype(bigvoice_InitProcess            )
    >("bigvoice_InitProcess"            );
00398               InitProcess2            = m_lib.GetSymbol< decltype(bigvoice_InitProcess2           )
    >("bigvoice_InitProcess2"           );
00399               TerminateProcess        = m_lib.GetSymbol< decltype(bigvoice_TerminateProcess       )
    >("bigvoice_TerminateProcess"       );
00400               ExitProcess             = m_lib.GetSymbol< decltype(bigvoice_ExitProcess            )
    >("bigvoice_ExitProcess"            );
00401               StartUpdateThread       = m_lib.GetSymbol< decltype(bigvoice_StartUpdateThread      )
    >("bigvoice_StartUpdateThread"      );
00402               StopUpdateThread        = m_lib.GetSymbol< decltype(bigvoice_StopUpdateThread       )
    >("bigvoice_StopUpdateThread"       );
00403               WaitUpdateThreadReady   = m_lib.GetSymbol< decltype(bigvoice_WaitUpdateThreadReady  )
    >("bigvoice_WaitUpdateThreadReady"  );
00404               ProcessAudio            = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio           )
    >("bigvoice_ProcessAudio"           );
00405               ProcessAudio_Planar     = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio_Planar    )
    >("bigvoice_ProcessAudio_Planar"    );
00406               GetBufferIn             = m_lib.GetSymbol< decltype(bigvoice_GetBufferIn            )
    >("bigvoice_GetBufferIn"            );
00407               GetBufferOut            = m_lib.GetSymbol< decltype(bigvoice_GetBufferOut           )
    >("bigvoice_GetBufferOut"           );
00408               GetEstimatedDelay       = m_lib.GetSymbol< decltype(bigvoice_GetEstimatedDelay      )
    >("bigvoice_GetEstimatedDelay"      );
00409               GetFormatName           = m_lib.GetSymbol< decltype(bigvoice_GetFormatName          )
    >("bigvoice_GetFormatName"          );
00410               GetFormatFromName       = m_lib.GetSymbol< decltype(bigvoice_GetFormatFromName      )
    >("bigvoice_GetFormatFromName"      );
00411               GetBytesFromFormat      = m_lib.GetSymbol< decltype(bigvoice_GetBytesFromFormat     )
    >("bigvoice_GetBytesFromFormat"     );
00412               GetMaxPacketFrame       = m_lib.GetSymbol< decltype(bigvoice_GetMaxPacketFrame      )
    >("bigvoice_GetMaxPacketFrame"      );
00413               AddAudio                = m_lib.GetSymbol< decltype(bigvoice_AddAudio               )
    >("bigvoice_AddAudio"               );
00414               AddPadAudio             = m_lib.GetSymbol< decltype(bigvoice_AddPadAudio            )
    >("bigvoice_AddPadAudio"            );
00415               GetOutputCount          = m_lib.GetSymbol< decltype(bigvoice_GetOutputCount         )
    >("bigvoice_GetOutputCount"         );
00416               GetAudio                = m_lib.GetSymbol< decltype(bigvoice_GetAudio               )
    >("bigvoice_GetAudio"               );
00417               AudioConvertFrom        = m_lib.GetSymbol< decltype(bigvoice_AudioConvertFrom       )
    >("bigvoice_AudioConvertFrom"       );
00418               AudioConvertTo          = m_lib.GetSymbol< decltype(bigvoice_AudioConvertTo         )
    >("bigvoice_AudioConvertTo"         );
00419               StereoToMono            = m_lib.GetSymbol< decltype(bigvoice_StereoToMono           )
    >("bigvoice_StereoToMono"           );
00420               MonoToStereo            = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo           )
    >("bigvoice_MonoToStereo"           );
00421               AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(bigvoice_AudioMonoFromLiveStereo)
    >("bigvoice_AudioMonoFromLiveStereo");
00422               AudioMonoToLiveStereo   = m_lib.GetSymbol< decltype(bigvoice_AudioMonoToLiveStereo  )
    >("bigvoice_AudioMonoToLiveStereo"  );
00423               NewClient               = m_lib.GetSymbol< decltype(bigvoice_NewClient              )
    >("bigvoice_NewClient"              );
00424               DeleteClient            = m_lib.GetSymbol< decltype(bigvoice_DeleteClient           )
    >("bigvoice_DeleteClient"           );
00425               ProcessJson             = m_lib.GetSymbol< decltype(bigvoice_ProcessJson            )
    >("bigvoice_ProcessJson"            );
00426               FreeJsonAnswer          = m_lib.GetSymbol< decltype(bigvoice_FreeJsonAnswer         )
    >("bigvoice_FreeJsonAnswer"         );
00427               SaveState               = m_lib.GetSymbol< decltype(bigvoice_SaveState              )
    >("bigvoice_SaveState"              );
00428 #if BIGVOICE_HAS_WEBSERVER
00429               Webserver_tcp           = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp          )
    >("bigvoice_Webserver_tcp"          );
00430               Webserver_tcp2          = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp2         )
    >("bigvoice_Webserver_tcp2"         );
00431               Webserver               = m_lib.GetSymbol< decltype(bigvoice_Webserver              )
    >("bigvoice_Webserver"              );
00432               Webserver_Stop          = m_lib.GetSymbol< decltype(bigvoice_Webserver_Stop         )
    >("bigvoice_Webserver_Stop"         );
00433               Webserver_Status        = m_lib.GetSymbol< decltype(bigvoice_Webserver_Status       )
    >("bigvoice_Webserver_Status"       );
00434 #endif // BIGVOICE_HAS_WEBSERVER
```

```
00435
00436                 // added 2023-03-22
00437                 StereoToMono_Planar     = m_lib.GetSymbol< decltype(bigvoice_StereoToMono_Planar    )
      >("bigvoice_StereoToMono_Planar"    );
00438                 MonoToStereo_Planar     = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo_Planar    )
      >("bigvoice_MonoToStereo_Planar"    );
00439
00440             } catch (std::runtime_error& ) {
00441                 return false;
00442             }
00443             // C functions allowed to be missing
00444             try {
00445                 // added 2023-05-22
00446                 SetMetadata             = m_lib.GetSymbol< decltype(bigvoice_SetMetadata             )
      >("bigvoice_SetMetadata"            );
00447                 GetMetadataInfos        = m_lib.GetSymbol< decltype(bigvoice_GetMetadataInfos        )
      >("bigvoice_GetMetadataInfos"       );
00448                 FreeMetadataInfos       = m_lib.GetSymbol< decltype(bigvoice_FreeMetadataInfos       )
      >("bigvoice_FreeMetadataInfos"      );
00449                 // added 2023-06-20
00450                 SetPresetManager            = m_lib.GetSymbol< decltype(bigvoice_SetPresetManager
      ) >("bigvoice_SetPresetManager"             );
00451                 PresetManager_InformChange  = m_lib.GetSymbol<
      decltype(bigvoice_PresetManager_InformChange     ) >("bigvoice_PresetManager_InformChange"    );
00452
00453                 // added 2023-09-21
00454                 GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
      decltype(bigvoice_GetPossibleChunkSizeInFrames  ) >("bigvoice_GetPossibleChunkSizeInFrames"  );
00455                 GetProcessChunkFrames        = m_lib.GetSymbol<
      decltype(bigvoice_GetProcessChunkFrames         ) >("bigvoice_GetProcessChunkFrames"         );
00456                 InitProcess3                 = m_lib.GetSymbol< decltype(bigvoice_InitProcess3
      ) >("bigvoice_InitProcess3"                 );
00457
00458 #if BIGVOICE_HAS_CLOUDBUS
00459                 // added 2023-06-26
00460                 NewBus                  = m_lib.GetSymbol< decltype(bigvoice_NewBus                  )
      >("bigvoice_NewBus"                 );
00461                 FreeBus                 = m_lib.GetSymbol< decltype(bigvoice_FreeBus                 )
      >("bigvoice_FreeBus"                );
00462                 SetInstanceBus          = m_lib.GetSymbol< decltype(bigvoice_SetInstanceBus          )
      >("bigvoice_SetInstanceBus"         );
00463 #endif // BIGVOICE_HAS_CLOUDBUS
00464
00465                 // added 2024-02-19
00466                 SetMetadataMulti        = m_lib.GetSymbol< decltype(bigvoice_SetMetadataMulti        )
      >("bigvoice_SetMetadataMulti"       );
00467
00468 #if BIGVOICE_HAS_WEBSERVER
00469                 // added 2024-05-23
00470                 Webserver_SetAppHealth      = m_lib.GetSymbol<
      decltype(bigvoice_Webserver_SetAppHealth      ) >("Webserver_SetAppHealth"     );
00471                 Webserver_GetAppHealth      = m_lib.GetSymbol<
      decltype(bigvoice_Webserver_GetAppHealth      ) >("Webserver_GetAppHealth"     );
00472                 Webserver_FreeString        = m_lib.GetSymbol< decltype(bigvoice_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00473                 #endif // BIGVOICE_HAS_WEBSERVER
00474
00475 #ifdef _WIN32
00476                 // added 2025-05-28
00477                 SetInstanceTracing                = m_lib.GetSymbol<
      decltype(bigvoice_SetInstanceTracing               ) >("SetInstanceTracing"               );
00478                 SetInstanceTracingProcessActivity = m_lib.GetSymbol<
      decltype(bigvoice_SetInstanceTracingProcessActivity) >("SetInstanceTracingProcessActivity");
00479 #endif // _WIN32
00480
00481                 // added 2025-05-28
00482                 GetAudioConverter                 = m_lib.GetSymbol<
      decltype(bigvoice_GetAudioConverter                ) >("GetAudioConverter"                );
00483                 FreeAudioConverter                = m_lib.GetSymbol<
      decltype(bigvoice_FreeAudioConverter               ) >("FreeAudioConverter"               );
00484                 AudioConverter_From               = m_lib.GetSymbol<
      decltype(bigvoice_AudioConverter_From              ) >("AudioConverter_From"              );
00485                 AudioConverter_To                 = m_lib.GetSymbol<
      decltype(bigvoice_AudioConverter_To                ) >("AudioConverter_To"                );
00486
00487             }  catch (std::runtime_error& ) {
00488                 // Ignored, handler will take care
00489             }
00490
00491             m_bOK=true;
00492             return true;
00493         }
00500         bool IsOk() const {
00501             return m_bOK;
00502         }
00503
00504     };
```

```
00505
00506
00507
00508     static constexpr const char *process_name = "SOUND4 BIGVOICE.CL";
00509     static constexpr const char *process_shortname = "bigvoice";
00510     using SampleFormat = bigvoice_SampleFormat;
00515     const size_t ChunkFrames=BIGVOICE_AUDIOFRAME_COUNT;
00520     const size_t InputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT;
00525     const size_t OutputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT;
00530     const size_t InputSampleSize=ChunkFrames*InputChannels;
00535     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00540     const size_t SampleRate=BIGVOICE_SAMPLE_RATE;
00541
00545     namespace helper {
00546         #ifdef _WIN32
00552         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00553             if (wstr.empty()) return {};
00554             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00555             std::string str(size, 0);
00556             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00557             return str;
00558         }
00564         static inline std::wstring UTF8ToWString(const std::string& str) {
00565             if (str.empty()) return std::wstring();
00566             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00567             std::wstring wstr(size, 0);
00568             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00569             return wstr;
00570         }
00571         #endif // _WIN32
00572
00578             template<typename T>
00579         struct SampleFormat {
00580         };
00581             template<>
00582         struct SampleFormat<int16_t> {
00583             const bigvoice_SampleFormat format = S16_NATIVE;
00584         };
00585             template<>
00586         struct SampleFormat<int32_t> {
00587             const bigvoice_SampleFormat format = S32_NATIVE;
00588         };
00589             template<>
00590         struct SampleFormat<float> {
00591             const bigvoice_SampleFormat format = F32_NATIVE;
00592         };
00593
00594         // ------------------------------------------------------------
00606         static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
      size_t nSpl, bigvoice_SampleFormat fmt) {
00607             dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00608         }
00609
00621         static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
      size_t nSpl, bigvoice_SampleFormat fmt) {
00622             dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00623         }
00624
00635         static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00636             dynlib.StereoToMono(input, output, nFrame);
00637         }
00638
00649         static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00650             dynlib.MonoToStereo(input, output, nFrame);
00651         }
00652
00664         static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
      *inputR, float *output, size_t nFrame) {
00665             dynlib.StereoToMono_Planar(inputL, inputR, output, nFrame);
00666         }
00667
00679         static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
      float *outputR, size_t nFrame) {
00680             dynlib.MonoToStereo_Planar(input, outputL, outputR, nFrame);
00681         }
00682
00690         static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
      *output) {
00691             dynlib.AudioMonoFromLiveStereo(payload, output);
00692         }
00693
00701         static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
      *payload) {
00702             dynlib.AudioMonoToLiveStereo(input, payload);
```

```
00703         }
00704
00705     }; // namespace helper
00706
00713     static inline std::string GetVersion(CDynLib& dynlib)
00714         {
00715             return std::string(dynlib.GetVersion());
00716         }
00717
00729     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00730         {
00731             return dynlib.GetChunkSizeInFrames();
00732         }
00733
00743     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00744         {
00745             std::vector<unsigned int> list;
00746             if (dynlib.GetPossibleChunkSizeInFrames) {
00747                 for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00748                     list.push_back(*src);
00749                 }
00750             } else {
00751                 list.push_back(12);
00752             }
00753             return list;
00754         }
00755
00762     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00763         {
00764             return dynlib.GetChannelCount();
00765         }
00766
00775     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00776         {
00777             return dynlib.GetAudioInputCount();
00778         }
00779
00788     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00789         {
00790             return dynlib.GetAudioOutputCount();
00791         }
00792
00801     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00802         {
00803             return dynlib.GetSampleRate();
00804         }
00805
00813     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00814     {
00815         // Do some sanity checks
00816         if (dynlib.GetSampleRate()!=BIGVOICE_SAMPLE_RATE) {
00817             throw std::runtime_error("Bad library sampling rate");
00818         }
00819         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
00820             throw std::runtime_error("Bad library frame size");
00821         }
00822         if (dynlib.GetChannelCount()!=BIGVOICE_CHANNEL_COUNT) {
00823             throw std::runtime_error("Bad library channel count");
00824         }
00825         if (dynlib.GetAudioInputCount()!=BIGVOICE_AUDIO_INPUT_COUNT) {
00826             throw std::runtime_error("Bad library input count");
00827         }
00828         if (dynlib.GetAudioOutputCount()!=BIGVOICE_AUDIO_OUTPUT_COUNT) {
00829             throw std::runtime_error("Bad library output count");
00830         }
00831     }
00832
00840     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00841         {
00842             return std::string(dynlib.GetFormatName(fmt));
00843         }
00844
00852     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00853         {
00854             return dynlib.GetFormatFromName(name.c_str());
00855         }
00856
00864     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00865         {
00866             return dynlib.GetBytesFromFormat(fmt);
00867         }
00868
00872     using LogSeverity = bigvoice_LogSeverity;
00876     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00877
00881     static log_cb_t _log_cb;
00882     extern "C" {
```

```
00883         static inline void _log_cb_c(bigvoice_LogSeverity severity, const char *c_msg) {
00884             _log_cb(severity, std::string(c_msg));
00885         }
00886     }
00890     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00891         {
00892             dynlib.SetLogSeverity(severity);
00893         }
00897     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00898         {
00899             _log_cb=cb;
00900             dynlib.SetLoggerCallback(&_log_cb_c);
00901         }
00902
00903 #if BIGVOICE_HAS_CLOUDBUS
00909     class CBus {
00910         public:
00911             CBus(CDynLib& dynlib)
00912                 : m_dynlib(dynlib)
00913                 {
00914                     if (m_dynlib.NewBus) {
00915                         m_bus=m_dynlib.NewBus();
00916                     }
00917                 }
00918             ~CBus()
00919                 {
00920                     if (m_bus && m_dynlib.FreeBus) {
00921                         m_dynlib.FreeBus(m_bus);
00922                     }
00923                     m_bus = nullptr;
00924                 }
00925             bigvoice_CBus *Get() const { return m_bus; }
00926         private:
00927             CDynLib& m_dynlib;
00928             bigvoice_CBus *m_bus;
00929     };
00930 #endif // BIGVOICE_HAS_CLOUDBUS
00931
00937     class CPresetLoader {
00938     public:
00939         CPresetLoader() = default;
00940         virtual ~CPresetLoader() = default;
00941
00942         virtual bool IsReadOnly() = 0;
00943         virtual bool Exists(const std::filesystem::path &name) = 0;
00944         virtual bool Remove(const std::filesystem::path &name) = 0;
00945         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00946         virtual std::vector<std::filesystem::path> GetAll() = 0;
00947         virtual std::string Read(const std::filesystem::path &filename) = 0;
00948         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00949     };
00950     using PresetChange_Kind = bigvoice_PresetChange_Kind;
00951
00957     class CInstance {
00958     public:
00962         CInstance(CDynLib& dynlib)
00963         : m_dynlib(dynlib)
00964             {
00965                 params = m_dynlib.NewParameters();
00966                 if (!params) throw std::bad_alloc();
00967             }
00968         // non-copyable
00969         CInstance( const CInstance& ) = delete;
00970         CInstance& operator=( const CInstance& ) = delete;
00974         virtual ~CInstance()
00975             {
00976                 Stop();
00977                 if (params) m_dynlib.FreeParameters(params);
00978             }
00982        bool IsOk() const { return instance!=nullptr; }
00992         void SetParam(const std::string& name, const std::string& value)
00993             {
00994                 assert(params);
00995                 m_dynlib.SetParameter(params, name.c_str(), value.c_str());
00996             }
01003         std::string GetParam(const std::string& name)
01004             {
01005                 assert(params);
01006                 auto c_value=m_dynlib.GetParameter(params, name.c_str());
01007                 std::string ret(c_value);
01008                 m_dynlib.FreeParameterValue(c_value);
01009                 return ret;
01010             }
01011 #if BIGVOICE_HAS_CLOUDBUS
01020             void SetBus(const CBus& bus) {
01021                 assert(params);
01022                 if (m_dynlib.SetInstanceBus) {
```

```
01023                     m_dynlib.SetInstanceBus(params,bus.Get());
01024                 }
01025             }
01026 #endif // BIGVOICE_HAS_CLOUDBUS
01027
01038         void SetPresetManager(CPresetLoader *preset_manager)
01039             {
01040                 if (!preset_manager) return;
01041                 if (!m_dynlib.SetPresetManager) return;
01042                 assert(params);
01043                 m_dynlib.SetPresetManager(params,
01044                     bigvoice_custom_reader,
01045                     bigvoice_custom_reader_free,
01046                     bigvoice_custom_writer,
01047                     bigvoice_custom_exists,
01048                     bigvoice_custom_getall,
01049                     bigvoice_custom_getall_free,
01050                     bigvoice_custom_remove,
01051                     bigvoice_custom_rename,
01052                     preset_manager->IsReadOnly(),
01053                     preset_manager
01054                 );
01055             }
01056
01079         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)
01080             {
01081                 assert(!instance);
01082                 std::string l_save_path;
01083                 if (!save_path.empty()) {
01084                     #if (__cplusplus<202002L)
01085                     l_save_path = save_path.u8string();
01086                     #else // C++20
01087                     l_save_path = (const char*)save_path.u8string().c_str();
01088                     #endif // C++20
01089                 }
01090                 if (m_dynlib.InitProcess3) {
01091                     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);
01092                 } else if (frames_per_chunk!=12) {
01093                     return false;
01094                 } else {
01095                     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);
01096                 }
01097                 if (!instance) return false;
01098                 if (!init_metadata.empty()) {
01099                     for (auto&& [key,value]: init_metadata) {
01100                         m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
01101                     }
01102                     init_metadata.clear();
01103                 }
01104                 update_thread = std::thread([this, json_port](){
01105                     OnUpdateThreadStart();
01106                     m_dynlib.StartUpdateThread(instance, json_port);
01107                     OnUpdateThreadStop();
01108                 });
01109                 int timeout=5000;
01110                 #ifdef DEBUG
01111                     timeout=60*1000; // For debugger pauses
01112                 #endif
01113                 if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
01114                     return false;
01115                 }
01116                 return true;
01117             }
01118
01126         unsigned int GetChunkFrames() {
01127             if (instance && m_dynlib.GetProcessChunkFrames) {
01128                 return m_dynlib.GetProcessChunkFrames(instance);
01129             }
01130             return 12;
01131         }
01132
01140         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01141             {
01142                 if (m_dynlib.PresetManager_InformChange) {
01143                     m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
      (bigvoice_PresetChange_Kind) change_kind);
01144                 }
01145             }
01146
01158         void SetMetadata(const std::string& key, const char* value)
01159             {
01160                 if (m_dynlib.SetMetadata) {
```

```
01161                     if (instance) {
01162                         m_dynlib.SetMetadata(instance, key.c_str(), value);
01163                     } else {
01164                         init_metadata.push_back( {key, value} );
01165                     }
01166                 }
01167             }
01168 
01169 
01179         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01180             {
01181                 if (instance) {
01182                     if (m_dynlib.SetMetadataMulti) {
01183                         typedef const char* pchar;
01184                         pchar* keyvalue=new pchar[2*list.size()+1];
01185                         size_t n=0;
01186                         for (auto&& [key,value]: list) {
01187                             keyvalue[2*n+0]=key.c_str();
01188                             keyvalue[2*n+1]=value;
01189                             n++;
01190                         }
01191                         keyvalue[2*n+0]=nullptr;
01192 
01193                         bigvoice_SetMetadataMulti(instance, keyvalue);
01194 
01195                         delete[] keyvalue;
01196                     } else {
01197                         for (auto&& [key,value]: list) {
01198                             SetMetadata(key, value);
01199                         }
01200                     }
01201                 } else {
01202                     for (auto&& [key,value]: list) {
01203                         if (value) {
01204                             init_metadata.push_back( {key, value} );
01205                         }
01206                     }
01207                 }
01208             }
01209 
01215         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01216             {
01217                 std::vector< std::tuple<std::string,std::string> > values;
01218                 if (m_dynlib.GetMetadataInfos) {
01219                     const char** c_values = m_dynlib.GetMetadataInfos(instance);
01220                     if (c_values) {
01221                         for (const char** c_value=c_values; *c_value; ) {
01222                             std::string key(*c_value);
01223                             c_value++;
01224                             if (*c_value) {
01225                                 std::string descr(*c_value);
01226                                 c_value++;
01227                                 values.push_back( {key,descr} );
01228                             }
01229                         }
01230                         m_dynlib.FreeMetadataInfos(instance, c_values);
01231                     }
01232                 }
01233                 return values;
01234             }
01235 
01239         virtual void OnUpdateThreadStart() {}
01243         virtual void OnUpdateThreadStop() {}
01244 
01245 #if BIGVOICE_HAS_WEBSERVER
01259         bool StartWebServer(int http_port, int https_port=0)
01260             {
01261                 assert(instance);
01262                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01263                 webserver = m_dynlib.Webserver(http_port,https_port,instance);
01264                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01265                     return false;
01266                 }
01267                 int web_status=m_dynlib.Webserver_Status(webserver);
01268                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01269                     return false;
01270                 }
01271                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01272                     return false;
01273                 }
01274                 return true;
01275             }
01276 
01284         void StopWebServer(int timeout_ms = 1000)
01285             {
01286                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01287                     m_dynlib.Webserver_Stop(webserver, timeout_ms);
```

```
01288                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01289                 }
01290             }
01291
01295         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
01296             {
01297                 assert(instance);
01298                 if (!m_dynlib.Webserver_SetAppHealth) return;
01299                 m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
01300             }
01304         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01305             {
01306                 assert(instance);
01307                 if (!m_dynlib.Webserver_GetAppHealth) return;
01308                 char* c_contenttype=nullptr;
01309                 char* c_content=nullptr;
01310                 m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01311                 contenttype=c_contenttype;
01312                 content=c_content;
01313                 m_dynlib.Webserver_FreeString(c_contenttype);
01314                 m_dynlib.Webserver_FreeString(c_content);
01315             }
01316 #endif // BIGVOICE_HAS_WEBSERVER
01317
01326         int Terminate()
01327             {
01328                 if (!instance) return 0;
01329                 return m_dynlib.TerminateProcess(instance);
01330             }
01331
01335         void Stop()
01336             {
01337                 if (!instance) return;
01338 #if BIGVOICE_HAS_WEBSERVER
01339                 StopWebServer();
01340 #endif // BIGVOICE_HAS_WEBSERVER
01341                 if (update_thread.joinable()) {
01342                     m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01343                     update_thread.join();
01344                 }
01345                 m_dynlib.ExitProcess(instance);
01346                 instance=nullptr;
01347             }
01348
01358         unsigned int GetEstimatedDelay()
01359             {
01360                 assert(instance);
01361                 return m_dynlib.GetEstimatedDelay(instance);
01362             }
01363
01376         std::array<float,InputSampleSize>& GetBufferIn()
01377             {
01378                 assert(instance);
01379                 float *buf=m_dynlib.GetBufferIn(instance);
01380                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
01381
01382             }
01394         std::array<float,OutputSampleSize>& GetBufferOut()
01395             {
01396                 assert(instance);
01397                 float *buf= m_dynlib.GetBufferOut(instance);
01398                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
01399             }
01403         void ProcessAudio()
01404             {
01405                 assert(instance);
01406                 m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
      m_dynlib.GetBufferOut(instance));
01407             }
01419         void ProcessAudio_Planar(float const * const *input, float * const *output)
01420             {
01421                 assert(instance);
01422                 m_dynlib.ProcessAudio_Planar(instance, input, output);
01423             }
01424
01436             template<typename T>
01437         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01438             {
01439                 assert(instance);
01440                 std::vector<T> output;
01441                 unsigned int out_offset=0;
01442                 unsigned int in_offset=0;
01443                 unsigned int todo = input.size();
01444                 while (todo>0) {
01445                     unsigned int left = AddAudio(&input[in_offset], todo);
```

```
01446                     unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01447                     output.resize(out_offset + out_avail);
01448                     GetAudio(&output[out_offset], out_avail);
01449                     out_offset+=out_avail;
01450                     in_offset += todo-left;
01451                     todo=left;
01452                 }
01453                 return output;
01454             }
01455 
01461         class CClient {
01462             public:
01463                 CClient(bigvoice_CInstance* instance, CDynLib& dynlib)
01464                 : m_dynlib(dynlib)
01465                     {
01466                         assert(instance);
01467                         client=m_dynlib.NewClient(instance);
01468                         if (!client) throw std::bad_alloc();
01469                     }
01470                 // non-copyable
01471                 CClient( const CInstance& ) = delete;
01472                 CClient& operator=( const CInstance& ) = delete;
01473                 ~CClient()
01474                     {
01475                         if (client) {
01476                             m_dynlib.DeleteClient(client);
01477                             client=nullptr;
01478                         }
01479                     }
01487                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01488                     {
01489                         assert(client);
01490                         int need_save=0;
01491                         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
    &need_save);
01492                         if (!canswer) return {};
01493                         std::string answer(canswer);
01494                         m_dynlib.FreeJsonAnswer (canswer);
01495                         if (NeedSave) {
01496                             *NeedSave=(need_save!=0);
01497                         }
01498                         return answer;
01499                     }
01500             private:
01501                 CDynLib& m_dynlib;
01502                 bigvoice_CClientInstance *client = nullptr;
01503         };
01511         std::shared_ptr<CClient> NewClient()
01512             {
01513                 assert(instance);
01514                 return std::make_shared<CClient>(instance, m_dynlib);
01515             }
01516 
01527         bool SaveState() {
01528             assert(instance);
01529             if (m_dynlib.SaveState(instance)!=0) {
01530                 return false;
01531             }
01532             return true;
01533         }
01534 #ifdef _WIN32
01541         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01542             if (m_dynlib.SetInstanceTracing) {
01543                 m_dynlib.SetInstanceTracing(params, tracing_provider, activity_guid);
01544             }
01545         }
01552         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01553             assert(instance);
01554             if (m_dynlib.SetInstanceTracingProcessActivity) {
01555                 m_dynlib.SetInstanceTracingProcessActivity(instance, activity_guid);
01556             }
01557         }
01558 #endif // _WIN32
01559     protected:
01560             template<typename T>
01561         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01562             {
01563                 assert(instance);
01564                 return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
    helper::SampleFormat<T>::format);
01565             }
01566             template<typename T>
01567         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01568             {
01569                 assert(instance);
01570                 return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
    helper::SampleFormat<T>::format);
```

```
01571             }
01572
01573     private:
01574         CDynLib& m_dynlib;
01575         bigvoice_CParameters* params = nullptr;
01576         bigvoice_CInstance* instance = nullptr;
01577         std::thread update_thread;
01578         std::vector< std::pair<std::string,std::string> > init_metadata;
01579 #if BIGVOICE_HAS_WEBSERVER
01580         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01581 #endif // BIGVOICE_HAS_WEBSERVER
01582     };
01583
01590     class CAudioConverter {
01591         public:
01597             CAudioConverter(CDynLib& dynlib)
01598                 : m_dynlib(dynlib)
01599                 { }
01600
01604             virtual ~CAudioConverter()
01605                 {
01606                     Delete();
01607                 }
01608
01624             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
    bigvoice_ConverterTargetMode target_mode)
01625                 {
01626                     Delete();
01627                     if (m_dynlib.GetAudioConverter)
01628                         m_cvt = m_dynlib.GetAudioConverter((bigvoice_SampleFormat)fmt, nFrame,
    payl_ch, payl_ch_offset, target_mode);
01629                     return IsOK();
01630                 }
01631             void Delete()
01632                 {
01633                     if (m_cvt && m_dynlib.FreeAudioConverter) {
01634                         m_dynlib.FreeAudioConverter(m_cvt);
01635                     }
01636                     m_cvt = nullptr;
01637                 }
01638             bool IsOK() const { return (m_cvt!=nullptr); }
01639
01648             void ConvertFrom(const uint8_t *payload, float *output)
01649                 { m_dynlib.AudioConverter_From(m_cvt, payload, output); }
01650
01659             void ConvertTo(const float *input, uint8_t *payload)
01660                 { m_dynlib.AudioConverter_To(m_cvt, input, payload); }
01661
01662         private:
01663             CDynLib& m_dynlib;
01664             bigvoice_CAudioConverter *m_cvt = nullptr;
01665     };
01666
01667 }; // namespace dyn
01668 }; // namespace bigvoice
01669 }; // namespace sound4
01670
01671 // bridge C callbacks to CPresetLoader
01672 extern "C" {
01673     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
01674         sound4::bigvoice::dyn::CPresetLoader
    *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01675         std::string content=preset_loader->Read(filename);
01676         return strdup(content.c_str());
01677     }
01678     static void bigvoice_custom_reader_free(char *content, void* userdata) {
01679         if (!content) return;
01680         free(content);
01681     }
01682     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01683         sound4::bigvoice::dyn::CPresetLoader
    *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01684         auto res=preset_loader->Write(filename,content);
01685         return res?0:-1;
01686     }
01687     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
01688         sound4::bigvoice::dyn::CPresetLoader
    *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01689         auto res=preset_loader->Exists(filename);
01690         return res?0:-1;
01691     }
01692     static fs_char** bigvoice_custom_getall(void* userdata) {
01693         sound4::bigvoice::dyn::CPresetLoader
    *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01694         auto res=preset_loader->GetAll();
01695         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01696         for (size_t n=0;n<res.size();n++) {
```

```
01697             all[n]=fs_strdup(res[n].c_str());
01698         }
01699         all[res.size()]=nullptr;
01700         return all;
01701     }
01702     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
01703         if (!all) return;
01704         for (fs_char** one=all;*one;one++) {
01705             free(*one);
01706         }
01707         free(all);
01708     }
01709     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
01710         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01711         auto res=preset_loader->Remove(filename);
01712         return res?0:-1;
01713     }
01714     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01715         sound4::bigvoice::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01716         auto res=preset_loader->Rename(from,to);
01717         return res?0:-1;
01718     }
01719
01720 };
```

## 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
- class sound4::CAudioConverter

  *Audio converter helper.*

### 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 }
- enum sound4::ConverterTargetMode {
  sound4::ConverterTargetMode_AsInput = sound4cl_ConverterTargetMode_AsInput , sound4::ConverterTargetMode_Mono
  = sound4cl_ConverterTargetMode_Mono , sound4::ConverterTargetMode_Stereo = sound4cl_Converter↩
  TargetMode_Stereo , sound4::ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_↩
  StereoToMono ,
  sound4::ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo }

### 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         friend class CAudioConverter;
00318
00319         // All those wrapped functions have the same signature as there C sound4cl_XXX
00320         helper::DynFuncHolder< decltype(sound4cl_GetVersion             ) > fnGetVersion             ;
00321         helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames   ) > fnGetChunkSizeInFrames   ;
00322         helper::DynFuncHolder< decltype(sound4cl_GetChannelCount        ) > fnGetChannelCount        ;
00323         helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount     ) > fnGetAudioInputCount     ;
00324         helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount    ) > fnGetAudioOutputCount    ;
00325         helper::DynFuncHolder< decltype(sound4cl_GetSampleRate          ) > fnGetSampleRate          ;
00326         helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback      ) > fnSetLoggerCallback      ;
00327         helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity         ) > fnSetLogSeverity         ;
00328         helper::DynFuncHolder< decltype(sound4cl_NewParameters           ) > fnNewParameters          ;
00329         helper::DynFuncHolder< decltype(sound4cl_FreeParameters          ) > fnFreeParameters         ;
00330         helper::DynFuncHolder< decltype(sound4cl_SetParameter            ) > fnSetParameter           ;
00331         helper::DynFuncHolder< decltype(sound4cl_GetParameter            ) > fnGetParameter           ;
00332         helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue      ) > fnFreeParameterValue     ;
00333         helper::DynFuncHolder< decltype(sound4cl_InitProcess             ) > fnInitProcess            ;
00334         helper::DynFuncHolder< decltype(sound4cl_InitProcess2            ) > fnInitProcess2           ;
00335         helper::DynFuncHolder< decltype(sound4cl_TerminateProcess        ) > fnTerminateProcess       ;
00336         helper::DynFuncHolder< decltype(sound4cl_ExitProcess             ) > fnExitProcess            ;
00337         helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread       ) > fnStartUpdateThread      ;
00338         helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread        ) > fnStopUpdateThread       ;
00339         helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady   ) > fnWaitUpdateThreadReady  ;
00340         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio            ) > fnProcessAudio           ;
00341         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar     ) > fnProcessAudio_Planar    ;
00342         helper::DynFuncHolder< decltype(sound4cl_GetBufferIn             ) > fnGetBufferIn            ;
00343         helper::DynFuncHolder< decltype(sound4cl_GetBufferOut            ) > fnGetBufferOut           ;
00344         helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay       ) > fnGetEstimatedDelay      ;
00345         helper::DynFuncHolder< decltype(sound4cl_GetFormatName           ) > fnGetFormatName          ;
00346         helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName       ) > fnGetFormatFromName      ;
00347         helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat      ) > fnGetBytesFromFormat     ;
00348         helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame       ) > fnGetMaxPacketFrame      ;
00349         helper::DynFuncHolder< decltype(sound4cl_AddAudio                ) > fnAddAudio               ;
00350         helper::DynFuncHolder< decltype(sound4cl_AddPadAudio             ) > fnAddPadAudio            ;
00351         helper::DynFuncHolder< decltype(sound4cl_GetOutputCount          ) > fnGetOutputCount         ;
00352         helper::DynFuncHolder< decltype(sound4cl_GetAudio                ) > fnGetAudio               ;
00353         helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom        ) > fnAudioConvertFrom       ;
00354         helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo          ) > fnAudioConvertTo         ;
00355         helper::DynFuncHolder< decltype(sound4cl_StereoToMono            ) > fnStereoToMono           ;
00356         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo            ) > fnMonoToStereo           ;
00357         helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
00358         helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo   ) > fnAudioMonoToLiveStereo  ;
00359         helper::DynFuncHolder< decltype(sound4cl_NewClient               ) > fnNewClient              ;
00360         helper::DynFuncHolder< decltype(sound4cl_DeleteClient            ) > fnDeleteClient           ;
00361         helper::DynFuncHolder< decltype(sound4cl_ProcessJson             ) > fnProcessJson            ;
00362         helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer          ) > fnFreeJsonAnswer         ;
00363         helper::DynFuncHolder< decltype(sound4cl_SaveState               ) > fnSaveState              ;
00364
00365         // only for processes supporting webserver
00366         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp           ) > fnWebserver_tcp          ;
00367         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2          ) > fnWebserver_tcp2         ;
00368         helper::DynFuncHolder< decltype(sound4cl_Webserver               ) > fnWebserver              ;
00369         helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop          ) > fnWebserver_Stop         ;
00370         helper::DynFuncHolder< decltype(sound4cl_Webserver_Status        ) > fnWebserver_Status       ;
00371
00372         // added 2023-03-22
00373         helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar     ) > fnStereoToMono_Planar    ;
00374         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar     ) > fnMonoToStereo_Planar    ;
00375
00376         // added 2023-05-22
```

```
00377         helper::DynFuncHolder< decltype(sound4cl_SetMetadata            ) > fnSetMetadata            ;
00378         helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos       ) > fnGetMetadataInfos       ;
00379         helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos      ) > fnFreeMetadataInfos      ;
00380 
00381         // added 2023-06-20
00382         helper::DynFuncHolder< decltype(sound4cl_SetPresetManager               ) >
      fnSetPresetManager      ;
00383         helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange     ) >
      fnPresetManager_InformChange      ;
00384 
00385         // added 2023-09-21
00386         helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) >
      fnGetPossibleChunkSizeInFrames   ;
00387         helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) >
      fnGetProcessChunkFrames   ;
00388         helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3
      ;
00389 
00390         // added 2023-06-26, only for processes supporting bus
00391         helper::DynFuncHolder< decltype(sound4cl_NewBus                  ) > fnNewBus      ;
00392         helper::DynFuncHolder< decltype(sound4cl_FreeBus                 ) > fnFreeBus      ;
00393         helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus          ) > fnSetInstanceBus      ;
00394 
00395         // added 2024-02-19
00396         helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti        ) > fnSetMetadataMulti
      ;
00397 
00398         // added 2024-05-23
00399         helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) >
      fnWebserver_SetAppHealth      ;
00400         helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) >
      fnWebserver_GetAppHealth      ;
00401         helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString
      ;
00402 
00403 #ifdef _WIN32
00404         // added 2025-05-28
00405         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracing                ) >
      fnSetInstanceTracing               ;
00406         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracingProcessActivity ) >
      fnSetInstanceTracingProcessActivity ;
00407 #endif // _WIN32
00408 
00409         // added 2026-02-17
00410         helper::DynFuncHolder< decltype(sound4cl_GetAudioConverter      ) > fnGetAudioConverter      ;
00411         helper::DynFuncHolder< decltype(sound4cl_FreeAudioConverter     ) > fnFreeAudioConverter     ;
00412         helper::DynFuncHolder< decltype(sound4cl_AudioConverter_From    ) > fnAudioConverter_From    ;
00413         helper::DynFuncHolder< decltype(sound4cl_AudioConverter_To      ) > fnAudioConverter_To      ;
00414 
00415             template <typename T>
00416         helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
00417             std::string prefix_name = m_prefix+name;
00418             return m_lib.GetSymbol<T>(prefix_name);
00419         }
00420 
00421     public:
00422         CProcessor() = default;
00423 
00431         bool Load(const std::filesystem::path& filepath) {
00432             m_lib.Close();
00433             // If a path is given, use it directly and do not try other path
00434             if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
00435                 return false;
00436             } else if (!m_lib.IsOk()) {
00437                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00438                 auto filename = filepath.filename();
00439                 #ifndef _WIN32
00440                 // Linux: if in a bin directory, try ../lib first
00441                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00442                     auto libdir = thisdir.parent_path() / "lib";
00443                     m_lib.Load(libdir / filename);
00444                 }
00445                 #endif // !_WIN32
00446                 if (!m_lib.IsOk()) {
00447                     // Search in the same directory this code is in
00448                     m_lib.Load(thisdir / filename);
00449                 }
00450                 if (!m_lib.IsOk()) {
00451                     // Try current path
00452                     std::error_code ec;
00453                     auto p = std::filesystem::current_path(ec);
00454                     if (!ec) {
00455                         m_lib.Load(p / filename);
00456                     }
00457                 }
00458                 if (!m_lib.IsOk()) {
00459                     return false;
```

```
00460                 }
00461             }
00462             // Get information from the library itself
00463             try {
00464                 auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             )
      >("SOUND4_GetProcessInfo"   );
00465                 auto info = GetInfo();
00466                 m_prefix = info->prefix;
00467             } catch (std::runtime_error& ) {
00468                 // Compatibility: search in known libraries without this function
00469                 #ifdef _WIN32
00470                     std::string filename = helper::WStringToUTF8(filepath.stem());
00471                 #else // !_WIN32
00472                     std::string filename = filepath.stem();
00473                     // remove lib prefix
00474                     filename=filename.substr(3);
00475                 #endif // ! _WIN32
00476                 if (filename=="sound4.x1.cloud") {
00477                     m_prefix="cloudx1_";
00478                 } else if (filename=="sound4.impact.cl") {
00479                     m_prefix="impact_";
00480                 } else if (filename=="sound4.bigvoice.cl") {
00481                     m_prefix="bigvoice_";
00482                 } else {
00483                     return false;
00484                 }
00485             }
00486 
00487             // Load all C functions
00488             try {
00489                 fnGetVersion              = GetPrefixSymbol< decltype(sound4cl_GetVersion
      ) >("GetVersion"            );
00490                 fnGetChunkSizeInFrames    = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
      ) >("GetChunkSizeInFrames"  );
00491                 fnGetChannelCount         = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
      ) >("GetChannelCount"       );
00492                 fnGetAudioInputCount      = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
      ) >("GetAudioInputCount"    );
00493                 fnGetAudioOutputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
      ) >("GetAudioOutputCount"   );
00494                 fnGetSampleRate           = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
      ) >("GetSampleRate"         );
00495                 fnSetLoggerCallback       = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
      ) >("SetLoggerCallback"     );
00496                 fnSetLogSeverity          = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
      ) >("SetLogSeverity"        );
00497                 fnNewParameters           = GetPrefixSymbol< decltype(sound4cl_NewParameters
      ) >("NewParameters"         );
00498                 fnFreeParameters          = GetPrefixSymbol< decltype(sound4cl_FreeParameters
      ) >("FreeParameters"        );
00499                 fnSetParameter            = GetPrefixSymbol< decltype(sound4cl_SetParameter
      ) >("SetParameter"          );
00500                 fnGetParameter            = GetPrefixSymbol< decltype(sound4cl_GetParameter
      ) >("GetParameter"          );
00501                 fnFreeParameterValue      = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
      ) >("FreeParameterValue"    );
00502                 fnInitProcess             = GetPrefixSymbol< decltype(sound4cl_InitProcess
      ) >("InitProcess"           );
00503                 fnInitProcess2            = GetPrefixSymbol< decltype(sound4cl_InitProcess2
      ) >("InitProcess2"          );
00504                 fnTerminateProcess        = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
      ) >("TerminateProcess"      );
00505                 fnExitProcess             = GetPrefixSymbol< decltype(sound4cl_ExitProcess
      ) >("ExitProcess"           );
00506                 fnStartUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
      ) >("StartUpdateThread"     );
00507                 fnStopUpdateThread        = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
      ) >("StopUpdateThread"      );
00508                 fnWaitUpdateThreadReady   = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
      ) >("WaitUpdateThreadReady" );
00509                 fnProcessAudio            = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
      ) >("ProcessAudio"          );
00510                 fnProcessAudio_Planar     = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
      ) >("ProcessAudio_Planar"   );
00511                 fnGetBufferIn             = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
      ) >("GetBufferIn"           );
00512                 fnGetBufferOut            = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
      ) >("GetBufferOut"          );
00513                 fnGetEstimatedDelay       = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
      ) >("GetEstimatedDelay"     );
00514                 fnGetFormatName           = GetPrefixSymbol< decltype(sound4cl_GetFormatName
      ) >("GetFormatName"         );
00515                 fnGetFormatFromName       = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
      ) >("GetFormatFromName"     );
00516                 fnGetBytesFromFormat      = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
      ) >("GetBytesFromFormat"    );
00517                 fnGetMaxPacketFrame       = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
```

```
    ) >("GetMaxPacketFrame"      );
00518                fnAddAudio                = GetPrefixSymbol< decltype(sound4cl_AddAudio
    ) >("AddAudio"               );
00519                fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
    ) >("AddPadAudio"            );
00520                fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
    ) >("GetOutputCount"         );
00521                fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
    ) >("GetAudio"               );
00522                fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
    ) >("AudioConvertFrom"       );
00523                fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
    ) >("AudioConvertTo"         );
00524                fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
    ) >("StereoToMono"           );
00525                fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo
    ) >("MonoToStereo"           );
00526                fnAudioMonoFromLiveStereo = GetPrefixSymbol<
    decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
00527                fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo
    ) >("AudioMonoToLiveStereo"  );
00528                fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient
    ) >("NewClient"              );
00529                fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient
    ) >("DeleteClient"           );
00530                fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson
    ) >("ProcessJson"            );
00531                fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer
    ) >("FreeJsonAnswer"         );
00532                fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState
    ) >("SaveState"              );
00533            } catch (std::runtime_error& ) {
00534                return false;
00535            }
00536            // Webserver : not for all projects
00537            try {
00538                fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp
    ) >("Webserver_tcp"          );
00539                fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2
    ) >("Webserver_tcp2"         );
00540                fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver
    ) >("Webserver"              );
00541                fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop
    ) >("Webserver_Stop"         );
00542                fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status
    ) >("Webserver_Status"       );
00543            } catch (std::runtime_error& ) {
00544                // Ignored, handler will take care
00545            }
00546            // Stereo/Mono conversion : must be present
00547            try {
00548                // added 2023-03-22
00549                fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar
    ) >("StereoToMono_Planar"    );
00550                fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar
    ) >("MonoToStereo_Planar"    );
00551
00552            } catch (std::runtime_error& ) {
00553                return false;
00554            }
00555            // C functions allowed to be missing
00556            try {
00557                // added 2023-05-22
00558                fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata
    ) >("SetMetadata"            );
00559                fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos
    ) >("GetMetadataInfos"       );
00560                fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos
    ) >("FreeMetadataInfos"      );
00561                // added 2023-06-20
00562                fnSetPresetManager            = GetPrefixSymbol< decltype(sound4cl_SetPresetManager
    ) >("SetPresetManager"           );
00563                fnPresetManager_InformChange  = GetPrefixSymbol<
    decltype(sound4cl_PresetManager_InformChange     ) >("PresetManager_InformChange"    );
00564
00565                // added 2023-09-21
00566                fnGetPossibleChunkSizeInFrames = GetPrefixSymbol<
    decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
00567                fnGetProcessChunkFrames        = GetPrefixSymbol<
    decltype(sound4cl_GetProcessChunkFrames         ) >("GetProcessChunkFrames"         );
00568                fnInitProcess3                 = GetPrefixSymbol< decltype(sound4cl_InitProcess3
    ) >("InitProcess3"               );
00569            }  catch (std::runtime_error& ) {
00570                // Ignored, handler will take care
00571            }
00572
00573            // bus: not for all projects
```

```
00574             try {
00575                 // added 2023-06-26
00576                 fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus
      ) >("NewBus"                 );
00577                 fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus
      ) >("FreeBus"                );
00578                 fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus
      ) >("SetInstanceBus"         );
00579             }  catch (std::runtime_error& ) {
00580                 // Ignored, handler will take care
00581             }
00582
00583             // Other optional
00584             try {
00585                 // added 2024-02-19
00586                 fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti
      ) >("SetMetadataMulti"       );
00587
00588             }  catch (std::runtime_error& ) {
00589                 // Ignored, handler will take care
00590             }
00591
00592             // Other optional, web server only
00593             try {
00594                 // added 2024-05-23
00595                 fnWebserver_SetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
00596                 fnWebserver_GetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
00597                 fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00598             }  catch (std::runtime_error& ) {
00599                 // Ignored, handler will take care
00600             }
00601
00602 #ifdef _WIN32
00603             // Other optional, windows tracing
00604             try {
00605                 // added 2025-05-28
00606                 fnSetInstanceTracing              = GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracing                  ) >("SetInstanceTracing"               );
00607                 fnSetInstanceTracingProcessActivity= GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracingProcessActivity    ) >("SetInstanceTracingProcessActivity");
00608             }  catch (std::runtime_error& ) {
00609                 // Ignored, handler will take care
00610             }
00611 #endif // _WIN32
00612
00613             // Other optional
00614             try {
00615                 // added 2026-02-17
00616                 fnGetAudioConverter       = GetPrefixSymbol< decltype(sound4cl_GetAudioConverter
      ) >("GetAudioConverter"      );
00617                 fnFreeAudioConverter      = GetPrefixSymbol< decltype(sound4cl_FreeAudioConverter
      ) >("FreeAudioConverter"     );
00618                 fnAudioConverter_From     = GetPrefixSymbol< decltype(sound4cl_AudioConverter_From
      ) >("AudioConverter_From"    );
00619                 fnAudioConverter_To       = GetPrefixSymbol< decltype(sound4cl_AudioConverter_To
      ) >("AudioConverter_To"      );
00620             }  catch (std::runtime_error& ) {
00621                 // Ignored, handler will take care
00622             }
00623
00624             m_bOK=true;
00625             return true;
00626         }
00633         bool IsOk() const {
00634             return m_bOK;
00635         }
00636
00637         // ------------------------------------------------------------
00644         // ------------------------------------------------------------
00655         void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
00656             { fnAudioConvertFrom(payload, output, nSpl, (sound4cl_SampleFormat)fmt); }
00657
00668         void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
00669             { fnAudioConvertTo(input, payload, nSpl, (sound4cl_SampleFormat)fmt); }
00670
00680         void StereoToMono(const float *input, float *output, size_t nFrame)
00681             { fnStereoToMono(input, output, nFrame); }
00682
00692         void MonoToStereo(const float *input, float *output, size_t nFrame)
00693             { fnMonoToStereo(input, output, nFrame); }
00694
00705         void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame)
00706             { fnStereoToMono_Planar(inputL, inputR, output, nFrame); }
```

```
00707
00718         void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame)
00719             { fnMonoToStereo_Planar(input, outputL, outputR, nFrame); }
00720
00727         void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
00728             { fnAudioMonoFromLiveStereo(payload, output); }
00729
00736         void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
00737             { fnAudioMonoToLiveStereo(input, payload); }
00738
00746         std::string GetVersion()
00747             { return std::string(fnGetVersion()); }
00748
00759         unsigned int GetChunkSizeInFrames()
00760             { return fnGetChunkSizeInFrames(); }
00761
00770         std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00771             {
00772                 std::vector<unsigned int> list;
00773                 if (fnGetPossibleChunkSizeInFrames) {
00774                     for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
00775                         list.push_back(*src);
00776                     }
00777                 } else {
00778                     list.push_back(fnGetChunkSizeInFrames());
00779                 }
00780                 return list;
00781             }
00782
00788         unsigned int GetChannelCount()
00789             { return fnGetChannelCount(); }
00790
00798         unsigned int GetAudioInputCount()
00799             { return fnGetAudioInputCount(); }
00800
00808         unsigned int GetAudioOutputCount()
00809             { return fnGetAudioOutputCount(); }
00810
00818         unsigned int GetSampleRate()
00819             { return fnGetSampleRate(); }
00820
00827         std::string GetFormatName( const SampleFormat fmt)
00828             { return std::string(fnGetFormatName((sound4cl_SampleFormat)fmt)); }
00829
00836         SampleFormat GetFormatFromName( const std::string& name)
00837             { return SampleFormat(fnGetFormatFromName(name.c_str())); }
00838
00845         unsigned int GetBytesFromFormat( const SampleFormat fmt)
00846             { return fnGetBytesFromFormat((sound4cl_SampleFormat)fmt); }
00847
00851         void SetLogSeverity(LogSeverity severity)
00852             { fnSetLogSeverity((sound4cl_LogSeverity)severity); }
00853
00857         void SetLoggerCallback(log_cb_t cb)
00858             { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
00859     };
00860
00866     class CBus {
00867         public:
00868             CBus(CProcessor& dynlib)
00869                 : m_dynlib(dynlib)
00870                 {
00871                     if (m_dynlib.fnNewBus) {
00872                         m_bus=m_dynlib.fnNewBus();
00873                     }
00874                 }
00875             ~CBus()
00876                 {
00877                     if (m_bus && m_dynlib.fnFreeBus) {
00878                         m_dynlib.fnFreeBus(m_bus);
00879                     }
00880                     m_bus = nullptr;
00881                 }
00882         protected:
00883             friend class CInstance;
00884             sound4cl_CBus *Get() const { return m_bus; }
00885         private:
00886             CProcessor& m_dynlib;
00887             sound4cl_CBus *m_bus;
00888     };
00889
00895     class CPresetLoader {
00896     public:
00897         CPresetLoader() = default;
00898         virtual ~CPresetLoader() = default;
00899
00900         virtual bool IsReadOnly() = 0;
```

```
00901         virtual bool Exists(const std::filesystem::path &name) = 0;
00902         virtual bool Remove(const std::filesystem::path &name) = 0;
00903         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00904         virtual std::vector<std::filesystem::path> GetAll() = 0;
00905         virtual std::string Read(const std::filesystem::path &filename) = 0;
00906         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00907     };
00908     enum PresetChange_Kind {
00909         change_kind_created  = sound4cl_PresetChange_Kind::change_kind_created,
00910         change_kind_modified = sound4cl_PresetChange_Kind::change_kind_modified,
00911         change_kind_deleted  = sound4cl_PresetChange_Kind::change_kind_deleted,
00912     };
00918     class CInstance {
00919     public:
00925         CInstance(CProcessor& dynlib)
00926         : m_dynlib(dynlib)
00927             {
00928                 params = m_dynlib.fnNewParameters();
00929                 if (!params) throw std::bad_alloc();
00930             }
00931
00940         CInstance(CProcessor& dynlib, void* _instance)
00941         : m_dynlib(dynlib)
00942         , instance(reinterpret_cast<sound4cl_CInstance*>(_instance))
00943         , m_bOwned(false)
00944             {
00945             }
00946         // non-copyable
00947         CInstance( const CInstance& ) = delete;
00948         CInstance& operator=( const CInstance& ) = delete;
00952         virtual ~CInstance()
00953             {
00954                 Stop();
00955                 if (params) m_dynlib.fnFreeParameters(params);
00956             }
00960        bool IsOk() const { return instance!=nullptr; }
00970         void SetParam(const std::string& name, const std::string& value)
00971             {
00972                 assert(params);
00973                 m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
00974             }
00981         std::string GetParam(const std::string& name)
00982             {
00983                 assert(params);
00984                 auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
00985                 std::string ret(c_value);
00986                 m_dynlib.fnFreeParameterValue(c_value);
00987                 return ret;
00988             }
00989
00998         void SetBus(const CBus& bus) {
00999             assert(params);
01000             if (m_dynlib.fnSetInstanceBus) {
01001                 m_dynlib.fnSetInstanceBus(params,bus.Get());
01002             }
01003         }
01004
01015         void SetPresetManager(CPresetLoader *preset_manager)
01016             {
01017                 if (!preset_manager) return;
01018                 if (!m_dynlib.fnSetPresetManager) return;
01019                 assert(params);
01020                 m_dynlib.fnSetPresetManager(params,
01021                     sound4cl_custom_reader,
01022                     sound4cl_custom_reader_free,
01023                     sound4cl_custom_writer,
01024                     sound4cl_custom_exists,
01025                     sound4cl_custom_getall,
01026                     sound4cl_custom_getall_free,
01027                     sound4cl_custom_remove,
01028                     sound4cl_custom_rename,
01029                     preset_manager->IsReadOnly(),
01030                     preset_manager
01031                 );
01032             }
01033
01056         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)
01057             {
01058                 assert(!instance);
01059                 std::string l_save_path;
01060                 if (!save_path.empty()) {
01061                     #if (__cplusplus<202002L)
01062                     l_save_path = save_path.u8string();
01063                     #else // C++20
01064                     l_save_path = (const char*)save_path.u8string().c_str();
```

```
01065                     #endif // C++20
01066                 }
01067                 if (frames_per_chunk==0) {
01068                     frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
01069                 }
01070                 if (m_dynlib.fnInitProcess3) {
01071                     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);
01072                 } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
01073                     return false;
01074                 } else {
01075                     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);
01076                 }
01077                 if (!instance) return false;
01078                 if (!init_metadata.empty()) {
01079                     for (auto&& [key,value]: init_metadata) {
01080                         m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
01081                     }
01082                     init_metadata.clear();
01083                 }
01084                 update_thread = std::thread([this, json_port](){
01085                     OnUpdateThreadStart();
01086                     m_dynlib.fnStartUpdateThread(instance, json_port);
01087                     OnUpdateThreadStop();
01088                 });
01089                 int timeout=5000;
01090                 #ifdef DEBUG
01091                     timeout=60*1000; // For debugger pauses
01092                 #endif
01093                 if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
01094                     return false;
01095                 }
01096                 return true;
01097             }
01098
01106         unsigned int GetChunkFrames() {
01107             if (instance && m_dynlib.fnGetProcessChunkFrames) {
01108                 return m_dynlib.fnGetProcessChunkFrames(instance);
01109             }
01110             return m_dynlib.fnGetChunkSizeInFrames();
01111         }
01112
01120         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01121             {
01122                 if (m_dynlib.fnPresetManager_InformChange) {
01123                     m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(),
      (sound4cl_PresetChange_Kind) change_kind);
01124                 }
01125             }
01126
01138         void SetMetadata(const std::string& key, const char* value)
01139             {
01140                 if (m_dynlib.fnSetMetadata) {
01141                     if (instance) {
01142                         m_dynlib.fnSetMetadata(instance, key.c_str(), value);
01143                     } else {
01144                         init_metadata.push_back( {key, value} );
01145                     }
01146                 }
01147             }
01148
01158         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01159             {
01160                 if (instance) {
01161                     if (m_dynlib.fnSetMetadataMulti) {
01162                         typedef const char* pchar;
01163                         pchar* keyvalue=new pchar[2*list.size()+1];
01164                         size_t n=0;
01165                         for (auto&& [key,value]: list) {
01166                             keyvalue[2*n+0]=key.c_str();
01167                             keyvalue[2*n+1]=value;
01168                             n++;
01169                         }
01170                         keyvalue[2*n+0]=nullptr;
01171
01172                         sound4cl_SetMetadataMulti(instance, keyvalue);
01173
01174                         delete[] keyvalue;
01175                     } else {
01176                         for (auto&& [key,value]: list) {
01177                             SetMetadata(key, value);
01178                         }
01179                     }
01180                 } else {
01181                     for (auto&& [key,value]: list) {
```

```
01182                         if (value) {
01183                             init_metadata.push_back( {key, value} );
01184                         }
01185                     }
01186                 }
01187             }
01188
01194         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01195             {
01196                 std::vector< std::tuple<std::string,std::string> > values;
01197                 if (m_dynlib.fnGetMetadataInfos) {
01198                     const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
01199                     if (c_values) {
01200                         for (const char** c_value=c_values; *c_value; ) {
01201                             std::string key(*c_value);
01202                             c_value++;
01203                             if (*c_value) {
01204                                 std::string descr(*c_value);
01205                                 c_value++;
01206                                 values.push_back( {key,descr} );
01207                             }
01208                         }
01209                         m_dynlib.fnFreeMetadataInfos(instance, c_values);
01210                     }
01211                 }
01212                 return values;
01213             }
01214
01218         virtual void OnUpdateThreadStart() {}
01222         virtual void OnUpdateThreadStop() {}
01223
01224
01238         bool StartWebServer(int http_port, int https_port=0)
01239             {
01240                 assert(instance);
01241                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01242                 if (!m_dynlib.fnWebserver) {
01243                     return false;
01244                 }
01245                 webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
01246                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01247                     return false;
01248                 }
01249                 int web_status=m_dynlib.fnWebserver_Status(webserver);
01250                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01251                     return false;
01252                 }
01253                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01254                     return false;
01255                 }
01256                 return true;
01257             }
01258
01266             void StopWebServer(int timeout_ms = 1000)
01267             {
01268                 if (!m_dynlib.fnWebserver_Stop) return;
01269                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01270                     m_dynlib.fnWebserver_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.fnWebserver_SetAppHealth) return;
01282                 m_dynlib.fnWebserver_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.fnWebserver_GetAppHealth) return;
01291                 char* c_contenttype=nullptr;
01292                 char* c_content=nullptr;
01293                 m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01294                 contenttype=c_contenttype;
01295                 content=c_content;
01296                 m_dynlib.fnWebserver_FreeString(c_contenttype);
01297                 m_dynlib.fnWebserver_FreeString(c_content);
01298             }
01307         int Terminate()
01308             {
01309                 if (!instance) return 0;
01310                 return m_dynlib.fnTerminateProcess(instance);
01311             }
```

```
01312
01318         void Stop()
01319             {
01320                 if (!instance) return;
01321                 if (m_bOwned) {
01322                     StopWebServer();
01323                     if (update_thread.joinable()) {
01324                         m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
01325                         update_thread.join();
01326                     }
01327                     m_dynlib.fnExitProcess(instance);
01328                 }
01329                 instance=nullptr;
01330             }
01331
01341         unsigned int GetEstimatedDelay()
01342             {
01343                 assert(instance);
01344                 return m_dynlib.fnGetEstimatedDelay(instance);
01345             }
01346
01359         float* GetBufferIn()
01360             {
01361                 assert(instance);
01362                 return m_dynlib.fnGetBufferIn(instance);
01363
01364             }
01376         float* GetBufferOut()
01377             {
01378                 assert(instance);
01379                 return m_dynlib.fnGetBufferOut(instance);
01380             }
01384         void ProcessAudio()
01385             {
01386                 assert(instance);
01387                 m_dynlib.fnProcessAudio(instance, GetBufferIn(), GetBufferOut());
01388             }
01400         void ProcessAudio_Planar(float const * const *input, float * const *output)
01401             {
01402                 assert(instance);
01403                 m_dynlib.fnProcessAudio_Planar(instance, input, output);
01404             }
01405
01417             template<typename T>
01418         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01419             {
01420                 assert(instance);
01421                 std::vector<T> output;
01422                 unsigned int out_offset=0;
01423                 unsigned int in_offset=0;
01424                 unsigned int todo = input.size();
01425                 while (todo>0) {
01426                     unsigned int left = AddAudio(&input[in_offset], todo);
01427                     unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
01428                     output.resize(out_offset + out_avail);
01429                     GetAudio(&output[out_offset], out_avail);
01430                     out_offset+=out_avail;
01431                     in_offset += todo-left;
01432                     todo=left;
01433                 }
01434                 return output;
01435             }
01436
01442         class CClient {
01443             public:
01444                 CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
01445                 : m_dynlib(dynlib)
01446                     {
01447                         assert(instance);
01448                         client=m_dynlib.fnNewClient(instance);
01449                         if (!client) throw std::bad_alloc();
01450                     }
01451                 // non-copyable
01452                 CClient( const CInstance& ) = delete;
01453                 CClient& operator=( const CInstance& ) = delete;
01454                 ~CClient()
01455                     {
01456                         if (client) {
01457                             m_dynlib.fnDeleteClient(client);
01458                             client=nullptr;
01459                         }
01460                     }
01468                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01469                     {
01470                         assert(client);
01471                         int need_save=0;
01472                         const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(),
```

```
    &need_save);
01473                         if (!canswer) return {};
01474                         std::string answer(canswer);
01475                         m_dynlib.fnFreeJsonAnswer (canswer);
01476                         if (NeedSave) {
01477                             *NeedSave=(need_save!=0);
01478                         }
01479                         return answer;
01480                     }
01481             private:
01482                 CProcessor& m_dynlib;
01483                 sound4cl_CClientInstance *client = nullptr;
01484         };
01492         std::shared_ptr<CClient> NewClient()
01493             {
01494                 assert(instance);
01495                 return std::make_shared<CClient>(instance, m_dynlib);
01496             }
01497
01508         bool SaveState() {
01509             assert(instance);
01510             if (!m_dynlib.fnSaveState || m_dynlib.fnSaveState(instance)!=0) {
01511                 return false;
01512             }
01513             return true;
01514         }
01515 #ifdef _WIN32
01522         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01523             if (m_dynlib.fnSetInstanceTracing) {
01524                 m_dynlib.fnSetInstanceTracing(params, tracing_provider, activity_guid);
01525             }
01526         }
01533         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01534             assert(instance);
01535             if (m_dynlib.fnSetInstanceTracingProcessActivity) {
01536                 m_dynlib.fnSetInstanceTracingProcessActivity(instance, activity_guid);
01537             }
01538         }
01539 #endif // _WIN32
01540     protected:
01541             template<typename T>
01542         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01543             {
01544                 assert(instance);
01545                 return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
    helper::SampleFormat<T>::format);
01546             }
01547             template<typename T>
01548         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01549             {
01550                 assert(instance);
01551                 return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
    helper::SampleFormat<T>::format);
01552             }
01553
01554     private:
01555         CProcessor& m_dynlib;
01556         sound4cl_CParameters* params = nullptr;
01557         sound4cl_CInstance* instance = nullptr;
01558         std::thread update_thread;
01559         std::vector< std::pair<std::string,std::string> > init_metadata;
01560         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01561         bool m_bOwned = true;
01562     };
01563
01564
01565     // ------------------------------------------------------------
01571     enum ConverterTargetMode {
01572         ConverterTargetMode_AsInput      = sound4cl_ConverterTargetMode_AsInput,
01573         ConverterTargetMode_Mono         = sound4cl_ConverterTargetMode_Mono,
01574         ConverterTargetMode_Stereo       = sound4cl_ConverterTargetMode_Stereo,
01575         ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_StereoToMono,
01576         ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo,
01577     };
01578
01586     class CAudioConverter {
01587         public:
01593             CAudioConverter(CProcessor& dynlib)
01594                 : m_dynlib(dynlib)
01595                 { }
01596
01600             virtual ~CAudioConverter()
01601                 {
01602                     Delete();
01603                 }
01604
01620             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
```

```
    ConverterTargetMode target_mode)
01621                 {
01622                     Delete();
01623                     if (m_dynlib.fnGetAudioConverter)
01624                         m_cvt = m_dynlib.fnGetAudioConverter((sound4cl_SampleFormat)fmt, nFrame,
    payl_ch, payl_ch_offset, (sound4cl_ConverterTargetMode)target_mode);
01625                     return IsOK();
01626                 }
01627             void Delete()
01628                 {
01629                     if (m_cvt && m_dynlib.fnFreeAudioConverter) {
01630                         m_dynlib.fnFreeAudioConverter(m_cvt);
01631                     }
01632                     m_cvt = nullptr;
01633                 }
01634             bool IsOK() const { return (m_cvt!=nullptr); }
01635
01644             void ConvertFrom(const uint8_t *payload, float *output)
01645                 { m_dynlib.fnAudioConverter_From(m_cvt, payload, output); }
01646
01655             void ConvertTo(const float *input, uint8_t *payload)
01656                 { m_dynlib.fnAudioConverter_To(m_cvt, input, payload); }
01657
01658         private:
01659             CProcessor& m_dynlib;
01660             sound4cl_CAudioConverter *m_cvt = nullptr;
01661     };
01662
01663 }; // namespace sound4
01664
01665
01666 // bridge C callbacks to CPresetLoader
01667 extern "C" {
01668     static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
01669         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01670         std::string content=preset_loader->Read(filename);
01671         return strdup(content.c_str());
01672     }
01673     static void sound4cl_custom_reader_free(char *content, void* userdata) {
01674         if (!content) return;
01675         free(content);
01676     }
01677     static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01678         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01679         auto res=preset_loader->Write(filename,content);
01680         return res?0:-1;
01681     }
01682     static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
01683         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01684         auto res=preset_loader->Exists(filename);
01685         return res?0:-1;
01686     }
01687     static fs_char** sound4cl_custom_getall(void* userdata) {
01688         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01689         auto res=preset_loader->GetAll();
01690         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01691         for (size_t n=0;n<res.size();n++) {
01692             all[n]=fs_strdup(res[n].c_str());
01693         }
01694         all[res.size()]=nullptr;
01695         return all;
01696     }
01697     static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
01698         if (!all) return;
01699         for (fs_char** one=all;*one;one++) {
01700             free(*one);
01701         }
01702         free(all);
01703     }
01704     static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
01705         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01706         auto res=preset_loader->Remove(filename);
01707         return res?0:-1;
01708     }
01709     static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01710         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01711         auto res=preset_loader->Rename(from,to);
01712         return res?0:-1;
01713     }
01714
01715 };
```

## 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 Detailed Description

Used by sound4cl.hpp for dynamic library loading.

Definition in file sound4cl_cdef.h.

### 15.31.2 Macro Definition Documentation

#### 15.31.2.1 SOUND4_INVALID_WEBSERVER_ID

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

Definition at line 67 of file sound4cl_cdef.h.

#### 15.31.2.2 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 68 of file sound4cl_cdef.h.

#### 15.31.2.3 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 69 of file sound4cl_cdef.h.

### 15.31.3 Typedef Documentation

#### 15.31.3.1 fs_char

```
typedef wchar_t fs_char
```

Definition at line 19 of file sound4cl_cdef.h.

#### 15.31.3.2 TraceLoggingHProvider

```
typedef struct _tlgProvider_t const* TraceLoggingHProvider
```

Definition at line 227 of file sound4cl_cdef.h.

## 15.32 sound4cl_cdef.h

Go to the documentation of this file.

```
00001
00007 #pragma once
00008
00009 #include <stdint.h>
00010 #ifdef __unix__
00011     #include <sys/types.h>
00012 #endif
00013
00014 #ifdef _WIN32
00015     #include <stringapiset.h>
00016     #include <wchar.h>
00018     #define fs_strdup wcsdup
00019     typedef wchar_t fs_char;
00020     #ifdef _MSC_VER
00021         #define __BYTE_ORDER__  REG_DWORD
00022         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00023         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00024     #endif
00025     #ifdef _MSC_VER
00026         #pragma warning(disable : 4996)
00027     #endif
00028 #else // !_WIN32
00030     #define fs_strdup strdup
00031     typedef char fs_char;
00032 #endif // !_WIN32
00033
00034 #ifdef __cplusplus
00035 extern "C" {
00036 #endif // __cplusplus
00037
00038 struct sound4cl_SOUND4_ProcessInfo {
00039     size_t struct_size;
00040     const char *name;
00041     const char *prefix;
00042     const char *shortname;
00043     const char *version;
00044 };
00045 enum sound4cl_LogSeverity {
00046     sound4cl_none     = 0,
00047     sound4cl_fatal    = 1,
00048     sound4cl_error    = 2,
00049     sound4cl_warning  = 3,
00050     sound4cl_info     = 4,
00051     sound4cl_verbose  = 5,
00052     sound4cl_verbose2 = 6,
00053     sound4cl_verbose3 = 7,
00054     sound4cl_verbose4 = 8,
00055     sound4cl_verbose5 = 9,
00056 };
00057 struct sound4cl_CInstance;
00058 struct sound4cl_CParameters;
00059 struct sound4cl_CBus;
00060 struct sound4cl_CClientInstance;
00061 struct sound4cl_CAudioConverter;
```

```
00062 enum sound4cl_PresetChange_Kind {
00063     change_kind_created = 1,
00064     change_kind_modified   ,
00065     change_kind_deleted    ,
00066 };
00067 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
00068 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
00069 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
00070
00074 enum sound4cl_SampleFormat {
00075     INVALID_FORMAT=0,
00076     S16_LE,
00077     S16_BE,
00078     S24_LE,
00079     S24_BE,
00080     S32_LE,
00081     S32_BE,
00082     F32_LE,
00083     F32_BE,
00084
00085     // Native definitions, only links to the physical ones
00086     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00087         S16_NATIVE=S16_LE,
00088         S24_NATIVE=S24_LE,
00089         S32_NATIVE=S32_LE,
00090         F32_NATIVE=F32_LE,
00091     #else
00092         S16_NATIVE=S16_BE,
00093         S24_NATIVE=S24_BE,
00094         S32_NATIVE=S32_BE,
00095         F32_NATIVE=F32_BE,
00096     #endif
00097 };
00098
00102 enum sound4cl_ConverterTargetMode {
00103     sound4cl_ConverterTargetMode_AsInput = 0,
00104     sound4cl_ConverterTargetMode_Mono,
00105     sound4cl_ConverterTargetMode_Stereo,
00106     sound4cl_ConverterTargetMode_StereoToMono,
00107     sound4cl_ConverterTargetMode_MonoToStereo,
00108 };
00109
00110 const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
00111 const char *sound4cl_GetVersion();
00112 unsigned int sound4cl_GetChunkSizeInFrames();
00113 unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
00114 unsigned int sound4cl_GetChannelCount();
00115 unsigned int sound4cl_GetAudioInputCount();
00116 unsigned int sound4cl_GetAudioOutputCount();
00117 unsigned int sound4cl_GetSampleRate();
00118 typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
00119 void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
00120 void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
00121 struct sound4cl_CParameters *sound4cl_NewParameters();
00122 void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
00123 void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
00124 const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
00125 void sound4cl_FreeParameterValue(const char *value);
00126 struct sound4cl_CInstance *sound4cl_InitProcess(
00127         const char *LoginKey,
00128         const char *RadioName,
00129         const char *Access_Key_ID,
00130         const char *Access_Key_Secret,
00131         const char *save_path
00132     );
00133 struct sound4cl_CInstance *sound4cl_InitProcess2(
00134         const char *LoginKey,
00135         const char *RadioName,
00136         const char *Access_Key_ID,
00137         const char *Access_Key_Secret,
00138         const char *save_path,
00139         const struct sound4cl_CParameters *parameters
00140     );
00141 struct sound4cl_CInstance *sound4cl_InitProcess3(
00142         const char *LoginKey,
00143         const char *RadioName,
00144         const char *Access_Key_ID,
00145         const char *Access_Key_Secret,
00146         const char *save_path,
00147         const struct sound4cl_CParameters *parameters,
00148         unsigned int frames_per_chunk
00149     );
00150 unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
00151 int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
00152 void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
00153 struct sound4cl_CBus* sound4cl_NewBus();
00154 void sound4cl_FreeBus(struct sound4cl_CBus *bus);
```

```
00155 void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
00156 typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
00157 typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
00158 typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00159 typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
00160 typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
00161 typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
00162 typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
00163 typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00164 void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
00165     , sound4cl_storage_reader reader, sound4cl_storage_reader_free
00166     , sound4cl_storage_writer writer
00167     , sound4cl_storage_exists exists
00168     , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
00169     , sound4cl_storage_remove remove
00170     , sound4cl_storage_rename rename
00171     , int IsReadOnly
00172     , void* userdata
00173     );
00174 void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
      *relative_path, enum sound4cl_PresetChange_Kind change_kind);
00175 void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
00176 void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
00177 const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
00178 void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
00179 void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
00180 void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
00181 int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
00182 void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
00183 void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
00184     , float const * const *input
00185     , float * const *output);
00186 float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
00187 float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
00188 unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
00189 const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
00190 enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
00191 unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
00192 unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
00193 unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
      int nFrame, enum sound4cl_SampleFormat fmt );
00194 unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
00195 unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
00196 unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
      max_nFrame, enum sound4cl_SampleFormat fmt );
00197 void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00198 void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00199 void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
00200 void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
00201 void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame);
00202 void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
00203 void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00204 void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00205 struct sound4cl_CAudioConverter* sound4cl_GetAudioConverter(enum sound4cl_SampleFormat fmt, size_t
      nFrame, size_t payl_ch, size_t payl_ch_offset, sound4cl_ConverterTargetMode tartget_mode);
00206 void sound4cl_FreeAudioConverter(struct sound4cl_CAudioConverter *converter);
00207 void sound4cl_AudioConverter_From(struct sound4cl_CAudioConverter *converter, const uint8_t *payload,
      float *output);
00208 void sound4cl_AudioConverter_To(struct sound4cl_CAudioConverter *converter, const float *input,
      uint8_t *payload);
00209 struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
00210 void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
00211 const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
      *need_save);
00212 void sound4cl_FreeJsonAnswer(const char *json_str);
00213 int sound4cl_SaveState(struct sound4cl_CInstance *instance);
00214 uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port);
00215 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);
00216 uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
      sound4cl_CInstance *instance);
00217 int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
00218 int sound4cl_Webserver_Status(uint64_t id);
00219 void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
      contenttype, const char* content);
00220 void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
      contenttype, char** content);
00221 void sound4cl_Webserver_FreeString(char* str);
00222
00223 //   Tracing (Windows only)
00224 #ifdef _WIN32
00225     #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00226         struct _tlgProvider_t;
```

```
00227         typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00228     #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00229 void sound4cl_SetInstanceTracing(struct sound4cl_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00230 void sound4cl_SetInstanceTracingProcessActivity(struct sound4cl_CInstance *instance, GUID
      activity_guid);
00231 #endif // _WIN32
00232
00233
00234 #ifdef __cplusplus
00235 } // extern "C"
00236 #endif // __cplusplus
```

# Index


_log_cb
  sound4, 90
  sound4::bigvoice, 96
  sound4::bigvoice::dyn, 105
_log_cb_c
  sound4, 89
  sound4::bigvoice, 92
  sound4::bigvoice::dyn, 99
~CAudioConverter
  sound4::bigvoice::CAudioConverter, 122
  sound4::bigvoice::dyn::CAudioConverter, 125
  sound4::CAudioConverter, 129
~CBus
  sound4::bigvoice::CBus, 131
  sound4::bigvoice::dyn::CBus, 132
  sound4::CBus, 133
~CClient
  sound4::bigvoice::CInstance::CClient, 135
  sound4::bigvoice::dyn::CInstance::CClient, 136
  sound4::CInstance::CClient, 138
~CDynLoader
  sound4::bigvoice::dyn::helper::CDynLoader, 152
  sound4::helper::CDynLoader, 155
~CInstance
  sound4::bigvoice::CInstance, 158
  sound4::bigvoice::dyn::CInstance, 170
  sound4::CInstance, 183
~CPresetLoader
  sound4::bigvoice::CPresetLoader, 194
  sound4::bigvoice::dyn::CPresetLoader, 196
  sound4::CPresetLoader, 198
~MyPresetManager
  MyPresetManager, 210

ActivateAuxiliaryAudioID
  sound4::bigvoice::CInstance, 158
AddAudio
  sound4::bigvoice::CInstance, 158
  sound4::bigvoice::dyn::CDynLib, 141
  sound4::bigvoice::dyn::CInstance, 170
  sound4::CInstance, 184
AddPadAudio
  sound4::bigvoice::dyn::CDynLib, 141
Audio convert helpers, 29
  bigvoice_AudioConverter_From, 30
  bigvoice_AudioConverter_To, 30
  bigvoice_AudioConvertFrom, 31
  bigvoice_AudioConvertTo, 31
  bigvoice_AudioMonoFromLiveStereo, 32
  bigvoice_AudioMonoToLiveStereo, 32
  bigvoice_ConverterTargetMode, 30
  bigvoice_FreeAudioConverter, 32
  bigvoice_GetAudioConverter, 32
  bigvoice_MonoToStereo, 33
  bigvoice_MonoToStereo_Planar, 34
  bigvoice_StereoToMono, 34
  bigvoice_StereoToMono_Planar, 34
  ConverterTargetMode_AsInput, 30
  ConverterTargetMode_Mono, 30
  ConverterTargetMode_MonoToStereo, 30
  ConverterTargetMode_Stereo, 30
  ConverterTargetMode_StereoToMono, 30
Audio convert helpers (C++), 35
  AudioConvertFrom, 36
  AudioConvertTo, 36
  AudioMonoFromLiveStereo, 37
  AudioMonoToLiveStereo, 37
  ConverterTargetMode, 36
  ConverterTargetMode_AsInput, 36
  ConverterTargetMode_Mono, 36
  ConverterTargetMode_MonoToStereo, 36
  ConverterTargetMode_Stereo, 36
  ConverterTargetMode_StereoToMono, 36
  MonoToStereo, 37
  MonoToStereo_Planar, 38
  StereoToMono, 38
  StereoToMono_Planar, 39
Audio managing (any audio), 39
  bigvoice_AddAudio, 41
  bigvoice_AddPadAudio, 41
  bigvoice_GetAudio, 41
  bigvoice_GetBytesFromFormat, 43
  bigvoice_GetFormatFromName, 43
  bigvoice_GetFormatName, 43
  bigvoice_GetMaxPacketFrame, 44
  bigvoice_GetOutputCount, 44
  bigvoice_SampleFormat, 40
  F32_BE, 40
  F32_LE, 40
  F32_NATIVE, 40
  INVALID_FORMAT, 40
  S16_BE, 40
  S16_LE, 40
  S16_NATIVE, 40
  S24_BE, 40
  S24_LE, 40
  S24_NATIVE, 40
  S32_BE, 40
  S32_LE, 40

S32_NATIVE, 40
AudioConverter_From
sound4::bigvoice::dyn::CDynLib, 141
AudioConverter_To
sound4::bigvoice::dyn::CDynLib, 141
AudioConvertFrom
Audio convert helpers (C++), 36
sound4::bigvoice::dyn::CDynLib, 141
sound4::bigvoice::dyn::helper, 108
sound4::bigvoice::helper, 113
AudioConvertTo
Audio convert helpers (C++), 36
sound4::bigvoice::dyn::CDynLib, 142
sound4::bigvoice::dyn::helper, 108
sound4::bigvoice::helper, 113
AudioMonoFromLiveStereo
Audio convert helpers (C++), 37
sound4::bigvoice::dyn::CDynLib, 142
sound4::bigvoice::dyn::helper, 109
sound4::bigvoice::helper, 113
AudioMonoToLiveStereo
Audio convert helpers (C++), 37
sound4::bigvoice::dyn::CDynLib, 142
sound4::bigvoice::dyn::helper, 109
sound4::bigvoice::helper, 114
Auxiliary outputs, 44
bigvoice_ActivateAuxiliaryAudioID, 45
bigvoice_FreeAuxiliaryAudioInfos, 45
bigvoice_GetAuxiliaryAudioEstimatedDelay, 46
bigvoice_GetAuxiliaryAudioOutputInfos, 46

bigvoice_ActivateAuxiliaryAudioID
Auxiliary outputs, 45
bigvoice_AddAudio
Audio managing (any audio), 41
bigvoice_AddPadAudio
Audio managing (any audio), 41
BIGVOICE_AUDIO_INPUT_COUNT
Global setup, 59
BIGVOICE_AUDIO_OUTPUT_COUNT
Global setup, 59
bigvoice_AudioConverter_From
Audio convert helpers, 30
bigvoice_AudioConverter_To
Audio convert helpers, 30
bigvoice_AudioConvertFrom
Audio convert helpers, 31
bigvoice_AudioConvertTo
Audio convert helpers, 31
BIGVOICE_AUDIOFRAME_COUNT
Global setup, 59
bigvoice_AudioMonoFromLiveStereo
Audio convert helpers, 32
bigvoice_AudioMonoToLiveStereo
Audio convert helpers, 32
bigvoice_CAudioConverter, 120
bigvoice_CBus, 121
bigvoice_CClientInstance, 121
BIGVOICE_CHANNEL_COUNT
Global setup, 59
bigvoice_CInstance, 121
bigvoice_ConverterTargetMode
Audio convert helpers, 30
bigvoice_CParameters, 121
bigvoice_custom_exists
sound4.bigvoice.cl.hpp, 249
sound4.bigvoice.cl_dyn.hpp, 262
bigvoice_custom_getall
sound4.bigvoice.cl.hpp, 249
sound4.bigvoice.cl_dyn.hpp, 262
bigvoice_custom_getall_free
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 262
bigvoice_custom_reader
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 262
bigvoice_custom_reader_free
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 262
bigvoice_custom_remove
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 263
bigvoice_custom_rename
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 263
bigvoice_custom_writer
sound4.bigvoice.cl.hpp, 250
sound4.bigvoice.cl_dyn.hpp, 263
BIGVOICE_DECL
sound4.bigvoice.cl.h, 242
bigvoice_DeleteClient
Client's JSon config interfacing, 47
bigvoice_ExitProcess
Process creation/destruction, 68
bigvoice_FreeAudioConverter
Audio convert helpers, 32
bigvoice_FreeAuxiliaryAudioInfos
Auxiliary outputs, 45
bigvoice_FreeBus
Instance's bus, 65
bigvoice_FreeJsonAnswer
Client's JSon config interfacing, 47
bigvoice_FreeMetadataInfos
Metadata Management, 66
bigvoice_FreeParameters
Process creation/destruction, 69
bigvoice_FreeParameterValue
Process creation/destruction, 69
bigvoice_GetAudio
Audio managing (any audio), 41
bigvoice_GetAudioConverter
Audio convert helpers, 32
bigvoice_GetAudioInputCount
Global setup, 61
bigvoice_GetAudioOutputCount
Global setup, 61
bigvoice_GetAuxiliaryAudioEstimatedDelay

Auxiliary outputs, 46
bigvoice_GetAuxiliaryAudioOutputInfos
Auxiliary outputs, 46
bigvoice_GetBufferIn
Direct processing, 55
bigvoice_GetBufferOut
Direct processing, 55
bigvoice_GetBytesFromFormat
Audio managing (any audio), 43
bigvoice_GetChannelCount
Global setup, 61
bigvoice_GetChunkSizeInFrames
Global setup, 61
bigvoice_GetEstimatedDelay
Direct processing, 56
bigvoice_GetFormatFromName
Audio managing (any audio), 43
bigvoice_GetFormatName
Audio managing (any audio), 43
bigvoice_GetMaxPacketFrame
Audio managing (any audio), 44
bigvoice_GetMetadataInfos
Metadata Management, 67
bigvoice_GetOutputCount
Audio managing (any audio), 44
bigvoice_GetParameter
Process creation/destruction, 69
bigvoice_GetPossibleChunkSizeInFrames
Global setup, 62
bigvoice_GetProcessChunkFrames
Process creation/destruction, 69
bigvoice_GetSampleRate
Global setup, 62
bigvoice_GetVersion
Global setup, 62
BIGVOICE_HAS_CLOUDBUS
sound4.bigvoice.cl.h, 242
BIGVOICE_HAS_WEBSERVER
sound4.bigvoice.cl.h, 242
bigvoice_InitProcess
Process creation/destruction, 70
bigvoice_InitProcess2
Process creation/destruction, 70
bigvoice_InitProcess3
Process creation/destruction, 70
bigvoice_loggerfn
Global setup, 60
bigvoice_LogSeverity
Global setup, 60
bigvoice_MonoToStereo
Audio convert helpers, 33
bigvoice_MonoToStereo_Planar
Audio convert helpers, 34
bigvoice_NewBus
Instance's bus, 65
bigvoice_NewClient
Client's JSon config interfacing, 47
bigvoice_NewParameters
Process creation/destruction, 71
bigvoice_PresetChange_Kind
Custom preset managagement, 53
bigvoice_PresetManager_InformChange
Custom preset managagement, 53
bigvoice_ProcessAudio
Direct processing, 56
bigvoice_ProcessAudio_Planar
Direct processing, 57
bigvoice_ProcessJson
Client's JSon config interfacing, 48
BIGVOICE_SAMPLE_RATE
Global setup, 59
bigvoice_SampleFormat
Audio managing (any audio), 40
bigvoice_SaveState
Client's JSon config interfacing, 48
bigvoice_SetInstanceBus
Instance's bus, 65
bigvoice_SetInstanceTracing
Instance tracing (Windows only), 64
bigvoice_SetInstanceTracingProcessActivity
Instance tracing (Windows only), 64
bigvoice_SetLoggerCallback
Global setup, 63
bigvoice_SetLogSeverity
Global setup, 63
bigvoice_SetMetadata
Metadata Management, 67
bigvoice_SetMetadataMulti
Metadata Management, 67
bigvoice_SetParameter
Process creation/destruction, 72
bigvoice_SetPresetManager
Custom preset managagement, 54
bigvoice_StartUpdateThread
Update Management, 80
bigvoice_StereoToMono
Audio convert helpers, 34
bigvoice_StereoToMono_Planar
Audio convert helpers, 34
bigvoice_StopUpdateThread
Update Management, 80
bigvoice_storage_exists
Custom preset managagement, 50
bigvoice_storage_getall
Custom preset managagement, 50
bigvoice_storage_getall_free
Custom preset managagement, 51
bigvoice_storage_reader
Custom preset managagement, 51
bigvoice_storage_reader_free
Custom preset managagement, 51
bigvoice_storage_remove
Custom preset managagement, 52
bigvoice_storage_rename
Custom preset managagement, 52
bigvoice_storage_writer

Custom preset managagement, 52
bigvoice_TerminateProcess
Process creation/destruction, 72
bigvoice_WaitUpdateThreadReady
Update Management, 80
bigvoice_Webserver
Web server, 82
bigvoice_Webserver_FreeString
Web server, 82
bigvoice_Webserver_GetAppHealth
Web server, 83
bigvoice_Webserver_SetAppHealth
Web server, 83
bigvoice_Webserver_Status
Web server, 83
bigvoice_Webserver_Stop
Web server, 84
bigvoice_Webserver_tcp
Web server, 84
bigvoice_Webserver_tcp2
Web server, 85
buffer
sound4::AuxiliaryOutputInfos, 119
SOUND4_AuxiliaryOutputInfos, 216

CAudioConverter
sound4::bigvoice::CAudioConverter, 122
sound4::bigvoice::dyn::CAudioConverter, 125
sound4::CAudioConverter, 128
sound4::CProcessor, 205
CBus
sound4::bigvoice::CBus, 131
sound4::bigvoice::dyn::CBus, 132
sound4::CBus, 133
sound4::CProcessor, 205
CClient
sound4::bigvoice::CInstance::CClient, 135
sound4::bigvoice::dyn::CInstance::CClient, 136
sound4::CInstance::CClient, 137
CDynLib
sound4::bigvoice::dyn::CDynLib, 140
CDynLoader
sound4::bigvoice::dyn::helper::CDynLoader, 152
sound4::helper::CDynLoader, 155
change_kind_created
Custom preset managagement, 53
sound4, 89
change_kind_deleted
Custom preset managagement, 53
sound4, 89
change_kind_modified
Custom preset managagement, 53
sound4, 89
CHANNEL_COUNT
sound4.bigvoice.cl-pipe.cpp, 229
channel_count
sound4::AuxiliaryOutputInfos, 119
SOUND4_AuxiliaryOutputInfos, 216
CHUNK_SIZE
sound4.bigvoice.cl-pipe.cpp, 229
ChunkFrames
sound4::bigvoice, 96
sound4::bigvoice::dyn, 105
CInstance
sound4::bigvoice::CInstance, 158
sound4::bigvoice::dyn::CInstance, 170
sound4::CBus, 134
sound4::CInstance, 183
sound4::CProcessor, 205
Client's JSon config interfacing, 46
bigvoice_DeleteClient, 47
bigvoice_FreeJsonAnswer, 47
bigvoice_NewClient, 47
bigvoice_ProcessJson, 48
bigvoice_SaveState, 48
Close
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::helper::CDynLoader, 155
Configure
sound4::bigvoice::CAudioConverter, 123
sound4::bigvoice::dyn::CAudioConverter, 125
sound4::CAudioConverter, 129
ConverterTargetMode
Audio convert helpers (C++), 36
ConverterTargetMode_AsInput
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_Mono
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_MonoToStereo
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_Stereo
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_StereoToMono
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConvertFrom
sound4::bigvoice::CAudioConverter, 123
sound4::bigvoice::dyn::CAudioConverter, 127
sound4::CAudioConverter, 129
ConvertTo
sound4::bigvoice::CAudioConverter, 124
sound4::bigvoice::dyn::CAudioConverter, 127
sound4::CAudioConverter, 130
CPresetLoader
sound4::bigvoice::CPresetLoader, 194
sound4::bigvoice::dyn::CPresetLoader, 196
sound4::CPresetLoader, 198
CProcessor
sound4::CProcessor, 201
Create
sound4::bigvoice::CInstance, 158
sound4::bigvoice::dyn::CInstance, 170
sound4::CInstance, 184

Custom preset managagement, 49
bigvoice_PresetChange_Kind, 53
bigvoice_PresetManager_InformChange, 53
bigvoice_SetPresetManager, 54
bigvoice_storage_exists, 50
bigvoice_storage_getall, 50
bigvoice_storage_getall_free, 51
bigvoice_storage_reader, 51
bigvoice_storage_reader_free, 51
bigvoice_storage_remove, 52
bigvoice_storage_rename, 52
bigvoice_storage_writer, 52
change_kind_created, 53
change_kind_deleted, 53
change_kind_modified, 53
fs_char, 53

Delete
sound4::bigvoice::CAudioConverter, 124
sound4::bigvoice::dyn::CAudioConverter, 128
sound4::CAudioConverter, 130
DeleteClient
sound4::bigvoice::dyn::CDynLib, 142
Deprecated List, 17
description
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 216
Direct processing, 54
bigvoice_GetBufferIn, 55
bigvoice_GetBufferOut, 55
bigvoice_GetEstimatedDelay, 56
bigvoice_ProcessAudio, 56
bigvoice_ProcessAudio_Planar, 57
doc_examples.h, 219
doc_gencpp.h, 219
doc_json.h, 219
doc_main.h, 220
doc_params.h, 220
documentation.h, 220
DynFuncHolder
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 206
sound4::helper::DynFuncHolder< T >, 208

Environment variables and parameters, 13
error
Global setup, 60
sound4, 88
Examples, 57
Exists
MyPresetManager, 210
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 196
sound4::CPresetLoader, 198
ExitProcess
sound4::bigvoice::dyn::CDynLib, 142

F32_BE
Audio managing (any audio), 40
sound4, 89
F32_LE
Audio managing (any audio), 40
sound4, 89
F32_NATIVE
Audio managing (any audio), 40
sound4, 89
fatal
Global setup, 60
sound4, 88
FreeAudioConverter
sound4::bigvoice::dyn::CDynLib, 142
FreeBus
sound4::bigvoice::dyn::CDynLib, 142
FreeJsonAnswer
sound4::bigvoice::dyn::CDynLib, 143
FreeMetadataInfos
sound4::bigvoice::dyn::CDynLib, 143
FreeParameters
sound4::bigvoice::dyn::CDynLib, 143
FreeParameterValue
sound4::bigvoice::dyn::CDynLib, 143
fs_char
Custom preset managagement, 53
sound4cl_cdef.h, 296

Generic C++ access to SOUND4 .CL librairies, 3
Get
sound4::bigvoice::CBus, 132
sound4::bigvoice::dyn::CBus, 133
sound4::CBus, 134
GetAll
MyPresetManager, 210
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 196
sound4::CPresetLoader, 198
GetAudio
sound4::bigvoice::CInstance, 159
sound4::bigvoice::dyn::CDynLib, 143
sound4::bigvoice::dyn::CInstance, 171
sound4::CInstance, 185
GetAudioConverter
sound4::bigvoice::dyn::CDynLib, 143
GetAudioInputCount
sound4::bigvoice, 92
sound4::bigvoice::dyn, 99
sound4::bigvoice::dyn::CDynLib, 143
sound4::CProcessor, 201
GetAudioOutputCount
sound4::bigvoice, 92
sound4::bigvoice::dyn, 100
sound4::bigvoice::dyn::CDynLib, 144
sound4::CProcessor, 201
GetAuxiliaryAudioOutputInfos
sound4::bigvoice::CInstance, 159
GetBufferIn
sound4::bigvoice::CInstance, 160
sound4::bigvoice::dyn::CDynLib, 144
sound4::bigvoice::dyn::CInstance, 171

sound4::CInstance, 185
GetBufferOut
sound4::bigvoice::CInstance, 160
sound4::bigvoice::dyn::CDynLib, 144
sound4::bigvoice::dyn::CInstance, 172
sound4::CInstance, 185
GetBytesFromFormat
sound4::bigvoice, 92
sound4::bigvoice::dyn, 100
sound4::bigvoice::dyn::CDynLib, 144
sound4::CProcessor, 201
GetChannelCount
sound4::bigvoice, 93
sound4::bigvoice::dyn, 100
sound4::bigvoice::dyn::CDynLib, 144
sound4::CProcessor, 202
GetChunkFrames
sound4::bigvoice::CInstance, 160
sound4::bigvoice::dyn::CInstance, 172
sound4::CInstance, 185
GetChunkSizeInFrames
sound4::bigvoice, 93
sound4::bigvoice::dyn, 101
sound4::bigvoice::dyn::CDynLib, 144
sound4::CProcessor, 202
GetEnv
sound4.bigvoice.cl-pipe-cxx.cpp, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 224
sound4cl-pipe.cpp, 234
GetEstimatedDelay
sound4::bigvoice::CInstance, 161
sound4::bigvoice::dyn::CDynLib, 144
sound4::bigvoice::dyn::CInstance, 172
sound4::CInstance, 186
GetFormatFromName
sound4::bigvoice, 93
sound4::bigvoice::dyn, 101
sound4::bigvoice::dyn::CDynLib, 145
sound4::CProcessor, 202
GetFormatName
sound4::bigvoice, 94
sound4::bigvoice::dyn, 102
sound4::bigvoice::dyn::CDynLib, 145
sound4::CProcessor, 203
GetMaxPacketFrame
sound4::bigvoice::dyn::CDynLib, 145
GetMetadataInfos
sound4::bigvoice::CInstance, 161
sound4::bigvoice::dyn::CDynLib, 145
sound4::bigvoice::dyn::CInstance, 173
sound4::CInstance, 186
GetOutputCount
sound4::bigvoice::dyn::CDynLib, 145
GetParam
sound4::bigvoice::CInstance, 161
sound4::bigvoice::dyn::CInstance, 173
sound4::CInstance, 186
GetParameter
sound4::bigvoice::dyn::CDynLib, 145
GetPossibleChunkSizeInFrames
sound4::bigvoice, 94
sound4::bigvoice::dyn, 102
sound4::bigvoice::dyn::CDynLib, 145
sound4::CProcessor, 203
GetPrefixSymbol
sound4::CProcessor, 203
GetProcessChunkFrames
sound4::bigvoice::dyn::CDynLib, 146
GetSampleRate
sound4::bigvoice, 94
sound4::bigvoice::dyn, 103
sound4::bigvoice::dyn::CDynLib, 146
sound4::CProcessor, 204
GetSymbol
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::helper::CDynLoader, 155
GetSymbol_safe
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::helper::CDynLoader, 155
GetThisLibraryPath
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::helper::CDynLoader, 155
GetVersion
sound4::bigvoice, 95
sound4::bigvoice::dyn, 103
sound4::bigvoice::dyn::CDynLib, 146
sound4::CProcessor, 204
GetWebServerAppHealth
sound4::bigvoice::CInstance, 162
sound4::bigvoice::dyn::CInstance, 173
sound4::CInstance, 187
Global setup, 58
BIGVOICE_AUDIO_INPUT_COUNT, 59
BIGVOICE_AUDIO_OUTPUT_COUNT, 59
BIGVOICE_AUDIOFRAME_COUNT, 59
BIGVOICE_CHANNEL_COUNT, 59
bigvoice_GetAudioInputCount, 61
bigvoice_GetAudioOutputCount, 61
bigvoice_GetChannelCount, 61
bigvoice_GetChunkSizeInFrames, 61
bigvoice_GetPossibleChunkSizeInFrames, 62
bigvoice_GetSampleRate, 62
bigvoice_GetVersion, 62
bigvoice_loggerfn, 60
bigvoice_LogSeverity, 60
BIGVOICE_SAMPLE_RATE, 59
bigvoice_SetLoggerCallback, 63
bigvoice_SetLogSeverity, 63
error, 60
fatal, 60
info, 60
none, 60
SOUND4_GetProcessInfo, 63
verbose, 60
verbose2, 60
verbose3, 60

verbose4, 60
verbose5, 60
warning, 60

id
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 216
info
Global setup, 60
sound4, 88
InitProcess
sound4::bigvoice::dyn::CDynLib, 146
InitProcess2
sound4::bigvoice::dyn::CDynLib, 146
InitProcess3
sound4::bigvoice::dyn::CDynLib, 146
InputChannels
sound4::bigvoice, 96
sound4::bigvoice::dyn, 105
InputSampleSize
sound4::bigvoice, 96
sound4::bigvoice::dyn, 106
Instance tracing (Windows only), 64
bigvoice_SetInstanceTracing, 64
bigvoice_SetInstanceTracingProcessActivity, 64
Instance's bus, 65
bigvoice_FreeBus, 65
bigvoice_NewBus, 65
bigvoice_SetInstanceBus, 65
INVALID_FORMAT
Audio managing (any audio), 40
sound4, 89
IsOK
sound4::bigvoice::CAudioConverter, 124
sound4::bigvoice::dyn::CAudioConverter, 128
sound4::CAudioConverter, 130
IsOk
sound4::bigvoice::CInstance, 162
sound4::bigvoice::dyn::CDynLib, 140
sound4::bigvoice::dyn::CInstance, 174
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 207
sound4::CInstance, 187
sound4::CProcessor, 204
sound4::helper::CDynLoader, 156
sound4::helper::DynFuncHolder< T >, 208
IsReadOnly
MyPresetManager, 211
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 197
sound4::CPresetLoader, 199

JSON syntax, 5
JSON syntax for internal setup, 9

Load
sound4::bigvoice::dyn::CDynLib, 140
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::CProcessor, 204
sound4::helper::CDynLoader, 156
log_cb_t
sound4, 88
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99
LogSeverity
sound4, 88
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99

main
sound4.bigvoice.cl-pipe-cxx.cpp, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 224
sound4.bigvoice.cl-pipe.cpp, 230
sound4cl-pipe.cpp, 234
Metadata Management, 66
bigvoice_FreeMetadataInfos, 66
bigvoice_GetMetadataInfos, 67
bigvoice_SetMetadata, 67
bigvoice_SetMetadataMulti, 67
MonoToStereo
Audio convert helpers (C++), 37
sound4::bigvoice::dyn::CDynLib, 146
sound4::bigvoice::dyn::helper, 109
sound4::bigvoice::helper, 114
MonoToStereo_Planar
Audio convert helpers (C++), 38
sound4::bigvoice::dyn::CDynLib, 147
sound4::bigvoice::dyn::helper, 110
sound4::bigvoice::helper, 115
MyLogger
sound4.bigvoice.cl-pipe-cxx.cpp, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 225
sound4.bigvoice.cl-pipe.cpp, 230
MyPresetManager, 209
~MyPresetManager, 210
Exists, 210
GetAll, 210
IsReadOnly, 211
MyPresetManager, 210
Read, 211
Remove, 211
Rename, 212
SetupPath, 212
Write, 212

name
SOUND4_ProcessInfo, 217
NewBus
sound4::bigvoice::dyn::CDynLib, 147
NewClient
sound4::bigvoice::CInstance, 162
sound4::bigvoice::dyn::CDynLib, 147
sound4::bigvoice::dyn::CInstance, 174
sound4::CInstance, 187
NewParameters
sound4::bigvoice::dyn::CDynLib, 147
none

Global setup, 60
sound4, 88

OnUpdateThreadStart
sound4::bigvoice::CInstance, 162
sound4::bigvoice::dyn::CInstance, 174
sound4::CInstance, 187
OnUpdateThreadStop
sound4::bigvoice::CInstance, 162
sound4::bigvoice::dyn::CInstance, 174
sound4::CInstance, 187
operator bool
sound4::bigvoice::dyn::helper::CDynLoader, 153
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 207
sound4::helper::CDynLoader, 156
sound4::helper::DynFuncHolder< T >, 208
operator T*
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 207
sound4::helper::DynFuncHolder< T >, 208
operator=
sound4::bigvoice::CInstance, 163
sound4::bigvoice::CInstance::CClient, 135
sound4::bigvoice::dyn::CInstance, 174
sound4::bigvoice::dyn::CInstance::CClient, 136
sound4::bigvoice::dyn::helper::CDynLoader, 154
sound4::CInstance, 188
sound4::CInstance::CClient, 138
sound4::helper::CDynLoader, 156
OutputChannels
sound4::bigvoice, 96
sound4::bigvoice::dyn, 106
OutputSampleSize
sound4::bigvoice, 97
sound4::bigvoice::dyn, 106

prefix
SOUND4_ProcessInfo, 217
PresetChange_Kind
sound4, 88
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99
PresetManager_InformChange
sound4::bigvoice::CInstance, 163
sound4::bigvoice::dyn::CDynLib, 147
sound4::bigvoice::dyn::CInstance, 174
sound4::CInstance, 188
Process creation/destruction, 68
bigvoice_ExitProcess, 68
bigvoice_FreeParameters, 69
bigvoice_FreeParameterValue, 69
bigvoice_GetParameter, 69
bigvoice_GetProcessChunkFrames, 69
bigvoice_InitProcess, 70
bigvoice_InitProcess2, 70
bigvoice_InitProcess3, 70
bigvoice_NewParameters, 71
bigvoice_SetParameter, 72
bigvoice_TerminateProcess, 72
process_name
sound4::bigvoice, 97
sound4::bigvoice::dyn, 106
process_shortname
sound4::bigvoice, 97
sound4::bigvoice::dyn, 106
ProcessAnyAudio
sound4::bigvoice::CInstance, 163
sound4::bigvoice::dyn::CInstance, 175
sound4::CInstance, 188
ProcessAudio
sound4::bigvoice::CInstance, 164
sound4::bigvoice::dyn::CDynLib, 147
sound4::bigvoice::dyn::CInstance, 175
sound4::CInstance, 189
ProcessAudio_Planar
sound4::bigvoice::CInstance, 164
sound4::bigvoice::dyn::CDynLib, 147
sound4::bigvoice::dyn::CInstance, 175
sound4::CInstance, 189
ProcessJson
sound4::bigvoice::CInstance::CClient, 135
sound4::bigvoice::dyn::CDynLib, 148
sound4::bigvoice::dyn::CInstance::CClient, 136
sound4::CInstance::CClient, 138
ProcessLoop
sound4.bigvoice.cl-pipe-cxx.cpp, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 225
sound4.bigvoice.cl-pipe.cpp, 230
sound4cl-pipe.cpp, 235

Read
MyPresetManager, 211
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 197
sound4::CPresetLoader, 199
Remove
MyPresetManager, 211
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 197
sound4::CPresetLoader, 199
Rename
MyPresetManager, 212
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 197
sound4::CPresetLoader, 199

S16_BE
Audio managing (any audio), 40
sound4, 89
S16_LE
Audio managing (any audio), 40
sound4, 89
S16_NATIVE
Audio managing (any audio), 40
sound4, 89
S24_BE
Audio managing (any audio), 40

sound4, 89
S24_LE
Audio managing (any audio), 40
sound4, 89
S24_NATIVE
Audio managing (any audio), 40
sound4, 89
S32_BE
Audio managing (any audio), 40
sound4, 89
S32_LE
Audio managing (any audio), 40
sound4, 89
S32_NATIVE
Audio managing (any audio), 40
sound4, 89
Sample using C interface, 72
Sample using C++ interface wrapper, 74
Sample using C++ interface wrapper with dynamic library loading, 75
Sample using Generic C++ interface (multi-process compatible), 77
SAMPLE_RATE
sound4.bigvoice.cl-pipe.cpp, 229
sample_rate
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 216
SampleFormat
sound4, 89
sound4::bigvoice, 91
sound4::bigvoice::dyn, 99
SampleRate
sound4::bigvoice, 97
sound4::bigvoice::dyn, 107
SanityCheck
sound4::bigvoice, 95
sound4::bigvoice::dyn, 103
SaveState
sound4::bigvoice::CInstance, 164
sound4::bigvoice::dyn::CDynLib, 148
sound4::bigvoice::dyn::CInstance, 177
sound4::CInstance, 189
SetBus
sound4::bigvoice::CInstance, 165
sound4::bigvoice::dyn::CInstance, 177
sound4::CInstance, 190
SetInstanceBus
sound4::bigvoice::dyn::CDynLib, 148
SetInstanceTracing
sound4::bigvoice::CInstance, 165
sound4::bigvoice::dyn::CDynLib, 148
sound4::bigvoice::dyn::CInstance, 177
sound4::CInstance, 190
SetInstanceTracingProcessActivity
sound4::bigvoice::CInstance, 165
sound4::bigvoice::dyn::CDynLib, 148
sound4::bigvoice::dyn::CInstance, 178
sound4::CInstance, 190
SetLoggerCallback
sound4::bigvoice, 95
sound4::bigvoice::dyn, 105
sound4::bigvoice::dyn::CDynLib, 148
sound4::CProcessor, 205
SetLogSeverity
sound4::bigvoice, 95
sound4::bigvoice::dyn, 105
sound4::bigvoice::dyn::CDynLib, 148
sound4::CProcessor, 205
SetMetadata
sound4::bigvoice::CInstance, 166
sound4::bigvoice::dyn::CDynLib, 149
sound4::bigvoice::dyn::CInstance, 178
sound4::CInstance, 191
SetMetadataMulti
sound4::bigvoice::CInstance, 166
sound4::bigvoice::dyn::CDynLib, 149
sound4::bigvoice::dyn::CInstance, 178
sound4::CInstance, 191
SetParam
sound4::bigvoice::CInstance, 166
sound4::bigvoice::dyn::CInstance, 179
sound4::CInstance, 191
SetParameter
sound4::bigvoice::dyn::CDynLib, 149
SetPresetManager
sound4::bigvoice::CInstance, 167
sound4::bigvoice::dyn::CDynLib, 149
sound4::bigvoice::dyn::CInstance, 179
sound4::CInstance, 192
SetupPath
MyPresetManager, 212
SetWebServerAppHealth
sound4::bigvoice::CInstance, 167
sound4::bigvoice::dyn::CInstance, 179
sound4::CInstance, 192
shortname
SOUND4_ProcessInfo, 217
sound4, 87
_log_cb, 90
_log_cb_c, 89
change_kind_created, 89
change_kind_deleted, 89
change_kind_modified, 89
error, 88
F32_BE, 89
F32_LE, 89
F32_NATIVE, 89
fatal, 88
info, 88
INVALID_FORMAT, 89
log_cb_t, 88
LogSeverity, 88
none, 88
PresetChange_Kind, 88
S16_BE, 89
S16_LE, 89

S16_NATIVE, 89
S24_BE, 89
S24_LE, 89
S24_NATIVE, 89
S32_BE, 89
S32_LE, 89
S32_NATIVE, 89
SampleFormat, 89
verbose, 88
verbose2, 88
verbose3, 88
verbose4, 88
verbose5, 88
warning, 88
SOUND4 BIGVOICE.CL Library, 1
sound4.bigvoice.cl-pipe-cxx.cpp, 220, 222
GetEnv, 221
main, 221
MyLogger, 221
ProcessLoop, 221
Usage, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 224, 226
GetEnv, 224
main, 224
MyLogger, 225
ProcessLoop, 225
Usage, 225
sound4.bigvoice.cl-pipe.cpp, 229, 231
CHANNEL_COUNT, 229
CHUNK_SIZE, 229
main, 230
MyLogger, 230
ProcessLoop, 230
SAMPLE_RATE, 229
Usage, 230
sound4.bigvoice.cl.h, 239
BIGVOICE_DECL, 242
BIGVOICE_HAS_CLOUDBUS, 242
BIGVOICE_HAS_WEBSERVER, 242
TraceLoggingHProvider, 242
sound4.bigvoice.cl.hpp, 247
bigvoice_custom_exists, 249
bigvoice_custom_getall, 249
bigvoice_custom_getall_free, 250
bigvoice_custom_reader, 250
bigvoice_custom_reader_free, 250
bigvoice_custom_remove, 250
bigvoice_custom_rename, 250
bigvoice_custom_writer, 250
sound4.bigvoice.cl_dyn.hpp, 259
bigvoice_custom_exists, 262
bigvoice_custom_getall, 262
bigvoice_custom_getall_free, 262
bigvoice_custom_reader, 262
bigvoice_custom_reader_free, 262
bigvoice_custom_remove, 263
bigvoice_custom_rename, 263
bigvoice_custom_writer, 263
UNIXLIKE, 262
sound4::AuxiliaryOutputInfos, 119
buffer, 119
channel_count, 119
description, 120
id, 120
sample_rate, 120
sound4::bigvoice, 90
_log_cb, 96
_log_cb_c, 92
ChunkFrames, 96
GetAudioInputCount, 92
GetAudioOutputCount, 92
GetBytesFromFormat, 92
GetChannelCount, 93
GetChunkSizeInFrames, 93
GetFormatFromName, 93
GetFormatName, 94
GetPossibleChunkSizeInFrames, 94
GetSampleRate, 94
GetVersion, 95
InputChannels, 96
InputSampleSize, 96
log_cb_t, 91
LogSeverity, 91
OutputChannels, 96
OutputSampleSize, 97
PresetChange_Kind, 91
process_name, 97
process_shortname, 97
SampleFormat, 91
SampleRate, 97
SanityCheck, 95
SetLoggerCallback, 95
SetLogSeverity, 95
sound4::bigvoice::CAudioConverter, 122
~CAudioConverter, 122
CAudioConverter, 122
Configure, 123
ConvertFrom, 123
ConvertTo, 124
Delete, 124
IsOK, 124
sound4::bigvoice::CBus, 131
~CBus, 131
CBus, 131
Get, 132
sound4::bigvoice::CInstance, 156
~CInstance, 158
ActivateAuxiliaryAudioID, 158
AddAudio, 158
CInstance, 158
Create, 158
GetAudio, 159
GetAuxiliaryAudioOutputInfos, 159
GetBufferIn, 160
GetBufferOut, 160
GetChunkFrames, 160

GetEstimatedDelay, 161
GetMetadataInfos, 161
GetParam, 161
GetWebServerAppHealth, 162
IsOk, 162
NewClient, 162
OnUpdateThreadStart, 162
OnUpdateThreadStop, 162
operator=, 163
PresetManager_InformChange, 163
ProcessAnyAudio, 163
ProcessAudio, 164
ProcessAudio_Planar, 164
SaveState, 164
SetBus, 165
SetInstanceTracing, 165
SetInstanceTracingProcessActivity, 165
SetMetadata, 166
SetMetadataMulti, 166
SetParam, 166
SetPresetManager, 167
SetWebServerAppHealth, 167
StartWebServer, 167
Stop, 168
StopWebServer, 168
Terminate, 168
sound4::bigvoice::CInstance::CClient, 134
~CClient, 135
CClient, 135
operator=, 135
ProcessJson, 135
sound4::bigvoice::CPresetLoader, 194
~CPresetLoader, 194
CPresetLoader, 194
Exists, 195
GetAll, 195
IsReadOnly, 195
Read, 195
Remove, 195
Rename, 195
Write, 195
sound4::bigvoice::dyn, 98
_log_cb, 105
_log_cb_c, 99
ChunkFrames, 105
GetAudioInputCount, 99
GetAudioOutputCount, 100
GetBytesFromFormat, 100
GetChannelCount, 100
GetChunkSizeInFrames, 101
GetFormatFromName, 101
GetFormatName, 102
GetPossibleChunkSizeInFrames, 102
GetSampleRate, 103
GetVersion, 103
InputChannels, 105
InputSampleSize, 106
log_cb_t, 99
LogSeverity, 99
OutputChannels, 106
OutputSampleSize, 106
PresetChange_Kind, 99
process_name, 106
process_shortname, 106
SampleFormat, 99
SampleRate, 107
SanityCheck, 103
SetLoggerCallback, 105
SetLogSeverity, 105
sound4::bigvoice::dyn::CAudioConverter, 124
~CAudioConverter, 125
CAudioConverter, 125
Configure, 125
ConvertFrom, 127
ConvertTo, 127
Delete, 128
IsOK, 128
sound4::bigvoice::dyn::CBus, 132
~CBus, 132
CBus, 132
Get, 133
sound4::bigvoice::dyn::CDynLib, 138
AddAudio, 141
AddPadAudio, 141
AudioConverter_From, 141
AudioConverter_To, 141
AudioConvertFrom, 141
AudioConvertTo, 142
AudioMonoFromLiveStereo, 142
AudioMonoToLiveStereo, 142
CDynLib, 140
DeleteClient, 142
ExitProcess, 142
FreeAudioConverter, 142
FreeBus, 142
FreeJsonAnswer, 143
FreeMetadataInfos, 143
FreeParameters, 143
FreeParameterValue, 143
GetAudio, 143
GetAudioConverter, 143
GetAudioInputCount, 143
GetAudioOutputCount, 144
GetBufferIn, 144
GetBufferOut, 144
GetBytesFromFormat, 144
GetChannelCount, 144
GetChunkSizeInFrames, 144
GetEstimatedDelay, 144
GetFormatFromName, 145
GetFormatName, 145
GetMaxPacketFrame, 145
GetMetadataInfos, 145
GetOutputCount, 145
GetParameter, 145
GetPossibleChunkSizeInFrames, 145

GetProcessChunkFrames, 146
GetSampleRate, 146
GetVersion, 146
InitProcess, 146
InitProcess2, 146
InitProcess3, 146
IsOk, 140
Load, 140
MonoToStereo, 146
MonoToStereo_Planar, 147
NewBus, 147
NewClient, 147
NewParameters, 147
PresetManager_InformChange, 147
ProcessAudio, 147
ProcessAudio_Planar, 147
ProcessJson, 148
SaveState, 148
SetInstanceBus, 148
SetInstanceTracing, 148
SetInstanceTracingProcessActivity, 148
SetLoggerCallback, 148
SetLogSeverity, 148
SetMetadata, 149
SetMetadataMulti, 149
SetParameter, 149
SetPresetManager, 149
StartUpdateThread, 149
StereoToMono, 149
StereoToMono_Planar, 149
StopUpdateThread, 150
TerminateProcess, 150
WaitUpdateThreadReady, 150
Webserver, 150
Webserver_FreeString, 150
Webserver_GetAppHealth, 150
Webserver_SetAppHealth, 150
Webserver_Status, 151
Webserver_Stop, 151
Webserver_tcp, 151
Webserver_tcp2, 151
sound4::bigvoice::dyn::CInstance, 169
~CInstance, 170
AddAudio, 170
CInstance, 170
Create, 170
GetAudio, 171
GetBufferIn, 171
GetBufferOut, 172
GetChunkFrames, 172
GetEstimatedDelay, 172
GetMetadataInfos, 173
GetParam, 173
GetWebServerAppHealth, 173
IsOk, 174
NewClient, 174
OnUpdateThreadStart, 174
OnUpdateThreadStop, 174
operator=, 174
PresetManager_InformChange, 174
ProcessAnyAudio, 175
ProcessAudio, 175
ProcessAudio_Planar, 175
SaveState, 177
SetBus, 177
SetInstanceTracing, 177
SetInstanceTracingProcessActivity, 178
SetMetadata, 178
SetMetadataMulti, 178
SetParam, 179
SetPresetManager, 179
SetWebServerAppHealth, 179
StartWebServer, 180
Stop, 180
StopWebServer, 180
Terminate, 181
sound4::bigvoice::dyn::CInstance::CClient, 136
~CClient, 136
CClient, 136
operator=, 136
ProcessJson, 136
sound4::bigvoice::dyn::CPresetLoader, 196
~CPresetLoader, 196
CPresetLoader, 196
Exists, 196
GetAll, 196
IsReadOnly, 197
Read, 197
Remove, 197
Rename, 197
Write, 197
sound4::bigvoice::dyn::helper, 107
AudioConvertFrom, 108
AudioConvertTo, 108
AudioMonoFromLiveStereo, 109
AudioMonoToLiveStereo, 109
MonoToStereo, 109
MonoToStereo_Planar, 110
StereoToMono, 110
StereoToMono_Planar, 111
UTF8ToWString, 111
WStringToUTF8, 112
sound4::bigvoice::dyn::helper::CDynLoader, 151
~CDynLoader, 152
CDynLoader, 152
Close, 153
GetSymbol, 153
GetSymbol_safe, 153
GetThisLibraryPath, 153
IsOk, 153
Load, 153
operator bool, 153
operator=, 154
sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 206
DynFuncHolder, 206

IsOk, 207
operator bool, 207
operator T∗, 207
sound4::bigvoice::dyn::helper::SampleFormat< T >, 213
sound4::bigvoice::helper, 112
AudioConvertFrom, 113
AudioConvertTo, 113
AudioMonoFromLiveStereo, 113
AudioMonoToLiveStereo, 114
MonoToStereo, 114
MonoToStereo_Planar, 115
StereoToMono, 115
StereoToMono_Planar, 115
UTF8ToWString, 116
WStringToUTF8, 116
sound4::bigvoice::helper::SampleFormat< T >, 213
sound4::CAudioConverter, 128
∼CAudioConverter, 129
CAudioConverter, 128
Configure, 129
ConvertFrom, 129
ConvertTo, 130
Delete, 130
IsOK, 130
sound4::CBus, 133
∼CBus, 133
CBus, 133
CInstance, 134
Get, 134
sound4::CInstance, 181
∼CInstance, 183
AddAudio, 184
CInstance, 183
Create, 184
GetAudio, 185
GetBufferIn, 185
GetBufferOut, 185
GetChunkFrames, 185
GetEstimatedDelay, 186
GetMetadataInfos, 186
GetParam, 186
GetWebServerAppHealth, 187
IsOk, 187
NewClient, 187
OnUpdateThreadStart, 187
OnUpdateThreadStop, 187
operator=, 188
PresetManager_InformChange, 188
ProcessAnyAudio, 188
ProcessAudio, 189
ProcessAudio_Planar, 189
SaveState, 189
SetBus, 190
SetInstanceTracing, 190
SetInstanceTracingProcessActivity, 190
SetMetadata, 191
SetMetadataMulti, 191
SetParam, 191
SetPresetManager, 192
SetWebServerAppHealth, 192
StartWebServer, 192
Stop, 193
StopWebServer, 193
Terminate, 193
sound4::CInstance::CClient, 137
∼CClient, 138
CClient, 137
operator=, 138
ProcessJson, 138
sound4::CPresetLoader, 198
∼CPresetLoader, 198
CPresetLoader, 198
Exists, 198
GetAll, 198
IsReadOnly, 199
Read, 199
Remove, 199
Rename, 199
Write, 199
sound4::CProcessor, 200
CAudioConverter, 205
CBus, 205
CInstance, 205
CProcessor, 201
GetAudioInputCount, 201
GetAudioOutputCount, 201
GetBytesFromFormat, 201
GetChannelCount, 202
GetChunkSizeInFrames, 202
GetFormatFromName, 202
GetFormatName, 203
GetPossibleChunkSizeInFrames, 203
GetPrefixSymbol, 203
GetSampleRate, 204
GetVersion, 204
IsOk, 204
Load, 204
SetLoggerCallback, 205
SetLogSeverity, 205
sound4::helper, 117
UTF8ToWString, 117
WStringToUTF8, 117
sound4::helper::CDynLoader, 154
∼CDynLoader, 155
CDynLoader, 155
Close, 155
GetSymbol, 155
GetSymbol_safe, 155
GetThisLibraryPath, 155
IsOk, 156
Load, 156
operator bool, 156
operator=, 156
sound4::helper::DynFuncHolder< T >, 207
DynFuncHolder, 208

IsOk, 208
operator bool, 208
operator T∗, 208
sound4::helper::SampleFormat< T >, 215
SOUND4_AuxiliaryOutputInfos, 215
buffer, 216
channel_count, 216
description, 216
id, 216
sample_rate, 216
SOUND4_GetProcessInfo
Global setup, 63
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 295
Web server, 81
SOUND4_ProcessInfo, 216
name, 217
prefix, 217
shortname, 217
struct_size, 217
version, 217
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 295
Web server, 81
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 295
Web server, 82
sound4cl-pipe.cpp, 234, 236
GetEnv, 234
main, 234
ProcessLoop, 235
sound4cl.hpp, 278
UNIXLIKE, 280
sound4cl_cdef.h, 295
fs_char, 296
SOUND4_INVALID_WEBSERVER_ID, 295
SOUND4_WEBSERVER_HTTP_OK, 295
SOUND4_WEBSERVER_HTTPS_OK, 295
TraceLoggingHProvider, 296
StartUpdateThread
sound4::bigvoice::dyn::CDynLib, 149
StartWebServer
sound4::bigvoice::CInstance, 167
sound4::bigvoice::dyn::CInstance, 180
sound4::CInstance, 192
StereoToMono
Audio convert helpers (C++), 38
sound4::bigvoice::dyn::CDynLib, 149
sound4::bigvoice::dyn::helper, 110
sound4::bigvoice::helper, 115
StereoToMono_Planar
Audio convert helpers (C++), 39
sound4::bigvoice::dyn::CDynLib, 149
sound4::bigvoice::dyn::helper, 111
sound4::bigvoice::helper, 115
Stop
sound4::bigvoice::CInstance, 168
sound4::bigvoice::dyn::CInstance, 180
sound4::CInstance, 193
StopUpdateThread
sound4::bigvoice::dyn::CDynLib, 150
StopWebServer
sound4::bigvoice::CInstance, 168
sound4::bigvoice::dyn::CInstance, 180
sound4::CInstance, 193
struct_size
SOUND4_ProcessInfo, 217

Terminate
sound4::bigvoice::CInstance, 168
sound4::bigvoice::dyn::CInstance, 181
sound4::CInstance, 193
TerminateProcess
sound4::bigvoice::dyn::CDynLib, 150
TraceLoggingHProvider
sound4.bigvoice.cl.h, 242
sound4cl_cdef.h, 296

UNIXLIKE
sound4.bigvoice.cl_dyn.hpp, 262
sound4cl.hpp, 280
Update Management, 79
bigvoice_StartUpdateThread, 80
bigvoice_StopUpdateThread, 80
bigvoice_WaitUpdateThreadReady, 80
Usage
sound4.bigvoice.cl-pipe-cxx.cpp, 221
sound4.bigvoice.cl-pipe-dyn.cpp, 225
sound4.bigvoice.cl-pipe.cpp, 230
UTF8ToWString
sound4::bigvoice::dyn::helper, 111
sound4::bigvoice::helper, 116
sound4::helper, 117

verbose
Global setup, 60
sound4, 88
verbose2
Global setup, 60
sound4, 88
verbose3
Global setup, 60
sound4, 88
verbose4
Global setup, 60
sound4, 88
verbose5
Global setup, 60
sound4, 88
version
SOUND4_ProcessInfo, 217

WaitUpdateThreadReady
sound4::bigvoice::dyn::CDynLib, 150
warning
Global setup, 60
sound4, 88

Web server, 81
bigvoice_Webserver, 82
bigvoice_Webserver_FreeString, 82
bigvoice_Webserver_GetAppHealth, 83
bigvoice_Webserver_SetAppHealth, 83
bigvoice_Webserver_Status, 83
bigvoice_Webserver_Stop, 84
bigvoice_Webserver_tcp, 84
bigvoice_Webserver_tcp2, 85
SOUND4_INVALID_WEBSERVER_ID, 81
SOUND4_WEBSERVER_HTTP_OK, 81
SOUND4_WEBSERVER_HTTPS_OK, 82
Webserver
sound4::bigvoice::dyn::CDynLib, 150
Webserver_FreeString
sound4::bigvoice::dyn::CDynLib, 150
Webserver_GetAppHealth
sound4::bigvoice::dyn::CDynLib, 150
Webserver_SetAppHealth
sound4::bigvoice::dyn::CDynLib, 150
Webserver_Status
sound4::bigvoice::dyn::CDynLib, 151
Webserver_Stop
sound4::bigvoice::dyn::CDynLib, 151
Webserver_tcp
sound4::bigvoice::dyn::CDynLib, 151
Webserver_tcp2
sound4::bigvoice::dyn::CDynLib, 151
Write
MyPresetManager, 212
sound4::bigvoice::CPresetLoader, 195
sound4::bigvoice::dyn::CPresetLoader, 197
sound4::CPresetLoader, 199
WStringToUTF8
sound4::bigvoice::dyn::helper, 112
sound4::bigvoice::helper, 116
sound4::helper, 117