SOUND4 x1.CLOUD Library [1.5.7]

Generated by Doxygen 1.9.8

1 SOUND4 x1.CLOUD 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 . . . 7

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 . . . 8

3.3.11 Import a preset . . . 8

4 JSON syntax for internal setup 9

4.1 Get process and modules information . . . 9

4.2 Authentication management . . . 9

4.2.1 Difference between setup and admin account . . . 9

4.2.2 Access encoding . . . 10

4.2.3 Change setup account . . . 10

4.2.3.1 Set the setup/admin account . . . 10

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

4.2.3.3 Remove the setup/admin account . . . 10

4.2.4 Get user info . . . 10

4.2.5 Add or modify a user . . . 10

4.2.6 Removes a user . . . 11

4.2.7 List users . . . 11

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

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

6 Deprecated List **17**

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

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

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

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

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

12 Topic Documentation **29**
12.1 Audio convert helpers . . . 29
12.1.1 Detailed Description . . . 29
12.1.2 Function Documentation . . . 29
12.1.2.1 cloudx1_AudioConvertFrom() . . . 29
12.1.2.2 cloudx1_AudioConvertTo() . . . 30
12.1.2.3 cloudx1_AudioMonoFromLiveStereo() . . . 30

12.1.2.4 cloudx1_AudioMonoToLiveStereo() . . . 30
12.1.2.5 cloudx1_MonoToStereo() . . . 32
12.1.2.6 cloudx1_MonoToStereo_Planar() . . . 32
12.1.2.7 cloudx1_StereoToMono() . . . 33
12.1.2.8 cloudx1_StereoToMono_Planar() . . . 33
12.2 Audio convert helpers (C++) . . . 33
12.2.1 Detailed Description . . . 34
12.2.2 Function Documentation . . . 34
12.2.2.1 AudioConvertFrom() . . . 34
12.2.2.2 AudioConvertTo() . . . 34
12.2.2.3 AudioMonoFromLiveStereo() . . . 35
12.2.2.4 AudioMonoToLiveStereo() . . . 35
12.2.2.5 MonoToStereo() . . . 35
12.2.2.6 MonoToStereo_Planar() . . . 36
12.2.2.7 StereoToMono() . . . 36
12.2.2.8 StereoToMono_Planar() . . . 37
12.3 Audio managing (any audio) . . . 37
12.3.1 Detailed Description . . . 38
12.3.2 Enumeration Type Documentation . . . 38
12.3.2.1 cloudx1_SampleFormat . . . 38
12.3.3 Function Documentation . . . 39
12.3.3.1 cloudx1_AddAudio() . . . 39
12.3.3.2 cloudx1_AddPadAudio() . . . 39
12.3.3.3 cloudx1_GetAudio() . . . 39
12.3.3.4 cloudx1_GetBytesFromFormat() . . . 40
12.3.3.5 cloudx1_GetFormatFromName() . . . 40
12.3.3.6 cloudx1_GetFormatName() . . . 40
12.3.3.7 cloudx1_GetMaxPacketFrame() . . . 41
12.3.3.8 cloudx1_GetOutputCount() . . . 41
12.4 Client's JSon config interfacing . . . 41
12.4.1 Detailed Description . . . 42
12.4.2 Function Documentation . . . 42
12.4.2.1 cloudx1_DeleteClient() . . . 42
12.4.2.2 cloudx1_FreeJsonAnswer() . . . 42
12.4.2.3 cloudx1_NewClient() . . . 42
12.4.2.4 cloudx1_ProcessJson() . . . 43
12.4.2.5 cloudx1_SaveState() . . . 43
12.5 Custom preset managagement . . . 44
12.5.1 Detailed Description . . . 45
12.5.2 Typedef Documentation . . . 45
12.5.2.1 cloudx1_storage_exists . . . 45
12.5.2.2 cloudx1_storage_getall . . . 45

12.5.2.3 cloudx1_storage_getall_free . . . 46
12.5.2.4 cloudx1_storage_reader . . . 46
12.5.2.5 cloudx1_storage_reader_free . . . 46
12.5.2.6 cloudx1_storage_remove . . . 47
12.5.2.7 cloudx1_storage_rename . . . 47
12.5.2.8 cloudx1_storage_writer . . . 47
12.5.2.9 fs_char . . . 48
12.5.3 Enumeration Type Documentation . . . 48
12.5.3.1 cloudx1_PresetChange_Kind . . . 48
12.5.4 Function Documentation . . . 48
12.5.4.1 cloudx1_PresetManager_InformChange() . . . 48
12.5.4.2 cloudx1_SetPresetManager() . . . 49
12.6 Direct processing . . . 49
12.6.1 Detailed Description . . . 50
12.6.2 Function Documentation . . . 50
12.6.2.1 cloudx1_GetBufferIn() . . . 50
12.6.2.2 cloudx1_GetBufferOut() . . . 50
12.6.2.3 cloudx1_GetEstimatedDelay() . . . 51
12.6.2.4 cloudx1_ProcessAudio() . . . 51
12.6.2.5 cloudx1_ProcessAudio_Planar() . . . 52
12.7 Examples . . . 52
12.8 Global setup . . . 53
12.8.1 Detailed Description . . . 53
12.8.2 Macro Definition Documentation . . . 54
12.8.2.1 CLOUDX1_AUDIO_INPUT_COUNT . . . 54
12.8.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT . . . 54
12.8.2.3 CLOUDX1_AUDIOFRAME_COUNT . . . 54
12.8.2.4 CLOUDX1_CHANNEL_COUNT . . . 54
12.8.2.5 CLOUDX1_SAMPLE_RATE . . . 55
12.8.3 Typedef Documentation . . . 55
12.8.3.1 cloudx1_loggerfn . . . 55
12.8.4 Enumeration Type Documentation . . . 55
12.8.4.1 cloudx1_LogSeverity . . . 55
12.8.5 Function Documentation . . . 56
12.8.5.1 cloudx1_GetAudioInputCount() . . . 56
12.8.5.2 cloudx1_GetAudioOutputCount() . . . 56
12.8.5.3 cloudx1_GetChannelCount() . . . 56
12.8.5.4 cloudx1_GetChunkSizeInFrames() . . . 57
12.8.5.5 cloudx1_GetPossibleChunkSizeInFrames() . . . 57
12.8.5.6 cloudx1_GetSampleRate() . . . 57
12.8.5.7 cloudx1_GetVersion() . . . 58
12.8.5.8 cloudx1_SetLoggerCallback() . . . 58

12.8.5.9 cloudx1_SetLogSeverity() . . . 58
12.8.5.10 SOUND4_GetProcessInfo() . . . 58
12.9 Instance tracing (Windows only) . . . 59
12.9.1 Detailed Description . . . 59
12.9.2 Function Documentation . . . 59
12.9.2.1 cloudx1_SetInstanceTracing() . . . 59
12.9.2.2 cloudx1_SetInstanceTracingProcessActivity() . . . 59
12.10 Metadata Management . . . 60
12.10.1 Detailed Description . . . 60
12.10.2 Function Documentation . . . 60
12.10.2.1 cloudx1_FreeMetadataInfos() . . . 60
12.10.2.2 cloudx1_GetMetadataInfos() . . . 60
12.10.2.3 cloudx1_SetMetadata() . . . 61
12.10.2.4 cloudx1_SetMetadataMulti() . . . 61
12.11 Process creation/destruction . . . 61
12.11.1 Detailed Description . . . 62
12.11.2 Function Documentation . . . 62
12.11.2.1 cloudx1_ExitProcess() . . . 62
12.11.2.2 cloudx1_FreeParameters() . . . 62
12.11.2.3 cloudx1_FreeParameterValue() . . . 63
12.11.2.4 cloudx1_GetParameter() . . . 63
12.11.2.5 cloudx1_GetProcessChunkFrames() . . . 63
12.11.2.6 cloudx1_InitProcess() . . . 63
12.11.2.7 cloudx1_InitProcess2() . . . 64
12.11.2.8 cloudx1_InitProcess3() . . . 64
12.11.2.9 cloudx1_NewParameters() . . . 65
12.11.2.10 cloudx1_SetParameter() . . . 65
12.11.2.11 cloudx1_TerminateProcess() . . . 65
12.12 Sample using C interface . . . 66
12.12.1 Sample to create the processing instance . . . 66
12.12.2 Sample processing loop . . . 67
12.12.3 Sample to stop the processing instance . . . 67
12.13 Sample using C++ interface wrapper . . . 67
12.13.1 Setup and sanity checks . . . 68
12.13.2 Sample to create the processing instance . . . 68
12.13.3 Communicating with the instance . . . 68
12.13.4 Starting a web-server . . . 68
12.13.5 Sample processing loop . . . 68
12.14 Sample using C++ interface wrapper with dynamic library loading . . . 68
12.14.1 Dynamic loading . . . 68
12.14.2 Setup and sanity checks . . . 69
12.14.3 Sample to create the processing instance . . . 69

12.14.4 Communicating with the instance . . . 69
12.14.5 Starting a web-server . . . 69
12.14.6 Sample processing loop . . . 69
12.14.7 Custom preset manager . . . 69
12.14.8 Custom preset manager loading . . . 70
12.15 Sample using Generic C++ interface (multi-process compatible) . . . 70
12.15.1 Loading the library . . . 71
12.15.2 Setup and configure . . . 71
12.15.3 Create the processing instance . . . 71
12.15.4 Processing loop . . . 71
12.15.5 Starting a web-server . . . 71
12.15.6 Communicating with the instance . . . 71
12.15.7 Custom preset manager example . . . 72
12.15.8 Custom preset manager loading . . . 72
12.16 Update Management . . . 73
12.16.1 Detailed Description . . . 73
12.16.2 Function Documentation . . . 73
12.16.2.1 cloudx1_StartUpdateThread() . . . 73
12.16.2.2 cloudx1_StopUpdateThread() . . . 73
12.16.2.3 cloudx1_WaitUpdateThreadReady() . . . 74
12.17 Web server . . . 74
12.17.1 Detailed Description . . . 75
12.17.2 Macro Definition Documentation . . . 75
12.17.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 75
12.17.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 75
12.17.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 75
12.17.3 Function Documentation . . . 75
12.17.3.1 cloudx1_Webserver() . . . 75
12.17.3.2 cloudx1_Webserver_FreeString() . . . 76
12.17.3.3 cloudx1_Webserver_GetAppHealth() . . . 76
12.17.3.4 cloudx1_Webserver_SetAppHealth() . . . 76
12.17.3.5 cloudx1_Webserver_Status() . . . 78
12.17.3.6 cloudx1_Webserver_Stop() . . . 78
12.17.3.7 cloudx1_Webserver_tcp() . . . 79
12.17.3.8 cloudx1_Webserver_tcp2() . . . 79

13 Namespace Documentation **81**

13.1 sound4 Namespace Reference . . . 81
13.1.1 Detailed Description . . . 82
13.1.2 Typedef Documentation . . . 82
13.1.2.1 log_cb_t . . . 82
13.1.3 Enumeration Type Documentation . . . 82

13.1.3.1 LogSeverity . . . 82
13.1.3.2 PresetChange_Kind . . . 83
13.1.3.3 SampleFormat . . . 83
13.1.4 Function Documentation . . . 83
13.1.4.1 _log_cb_c() . . . 83
13.1.5 Variable Documentation . . . 84
13.1.5.1 _log_cb . . . 84
13.2 sound4::cloudx1 Namespace Reference . . . 84
13.2.1 Detailed Description . . . 85
13.2.2 Typedef Documentation . . . 85
13.2.2.1 log_cb_t . . . 85
13.2.2.2 LogSeverity . . . 85
13.2.2.3 PresetChange_Kind . . . 85
13.2.2.4 SampleFormat . . . 85
13.2.3 Function Documentation . . . 86
13.2.3.1 _log_cb_c() . . . 86
13.2.3.2 GetAudioInputCount() . . . 86
13.2.3.3 GetAudioOutputCount() . . . 86
13.2.3.4 GetBytesFromFormat() . . . 86
13.2.3.5 GetChannelCount() . . . 87
13.2.3.6 GetChunkSizeInFrames() . . . 87
13.2.3.7 GetFormatFromName() . . . 87
13.2.3.8 GetFormatName() . . . 88
13.2.3.9 GetPossibleChunkSizeInFrames() . . . 88
13.2.3.10 GetSampleRate() . . . 89
13.2.3.11 GetVersion() . . . 89
13.2.3.12 SanityCheck() . . . 89
13.2.3.13 SetLoggerCallback() . . . 89
13.2.3.14 SetLogSeverity() . . . 90
13.2.4 Variable Documentation . . . 90
13.2.4.1 _log_cb . . . 90
13.2.4.2 ChunkFrames . . . 90
13.2.4.3 InputChannels . . . 90
13.2.4.4 InputSampleSize . . . 90
13.2.4.5 OutputChannels . . . 91
13.2.4.6 OutputSampleSize . . . 91
13.2.4.7 process_name . . . 91
13.2.4.8 process_shortname . . . 91
13.2.4.9 SampleRate . . . 91
13.3 sound4::cloudx1::dyn Namespace Reference . . . 92
13.3.1 Detailed Description . . . 93
13.3.2 Typedef Documentation . . . 93

13.3.2.1 log_cb_t . . . 93
13.3.2.2 LogSeverity . . . 93
13.3.2.3 PresetChange_Kind . . . 93
13.3.2.4 SampleFormat . . . 93
13.3.3 Function Documentation . . . 93
13.3.3.1 _log_cb_c() . . . 93
13.3.3.2 GetAudioInputCount() . . . 93
13.3.3.3 GetAudioOutputCount() . . . 94
13.3.3.4 GetBytesFromFormat() . . . 94
13.3.3.5 GetChannelCount() . . . 95
13.3.3.6 GetChunkSizeInFrames() . . . 95
13.3.3.7 GetFormatFromName() . . . 95
13.3.3.8 GetFormatName() . . . 96
13.3.3.9 GetPossibleChunkSizeInFrames() . . . 96
13.3.3.10 GetSampleRate() . . . 97
13.3.3.11 GetVersion() . . . 97
13.3.3.12 SanityCheck() . . . 97
13.3.3.13 SetLoggerCallback() . . . 99
13.3.3.14 SetLogSeverity() . . . 99
13.3.4 Variable Documentation . . . 99
13.3.4.1 _log_cb . . . 99
13.3.4.2 ChunkFrames . . . 99
13.3.4.3 InputChannels . . . 100
13.3.4.4 InputSampleSize . . . 100
13.3.4.5 OutputChannels . . . 100
13.3.4.6 OutputSampleSize . . . 100
13.3.4.7 process_name . . . 100
13.3.4.8 process_shortname . . . 101
13.3.4.9 SampleRate . . . 101
13.4 sound4::cloudx1::dyn::helper Namespace Reference . . . 101
13.4.1 Detailed Description . . . 102
13.4.2 Function Documentation . . . 102
13.4.2.1 AudioConvertFrom() . . . 102
13.4.2.2 AudioConvertTo() . . . 102
13.4.2.3 AudioMonoFromLiveStereo() . . . 103
13.4.2.4 AudioMonoToLiveStereo() . . . 103
13.4.2.5 MonoToStereo() . . . 103
13.4.2.6 MonoToStereo_Planar() . . . 104
13.4.2.7 StereoToMono() . . . 104
13.4.2.8 StereoToMono_Planar() . . . 105
13.4.2.9 UTF8ToWString() . . . 105
13.4.2.10 WStringToUTF8() . . . 106

13.5 sound4::cloudx1::helper Namespace Reference . . . 106
13.5.1 Detailed Description . . . 106
13.5.2 Function Documentation . . . 107
13.5.2.1 AudioConvertFrom() . . . 107
13.5.2.2 AudioConvertTo() . . . 107
13.5.2.3 AudioMonoFromLiveStereo() . . . 108
13.5.2.4 AudioMonoToLiveStereo() . . . 108
13.5.2.5 MonoToStereo() . . . 108
13.5.2.6 MonoToStereo_Planar() . . . 109
13.5.2.7 StereoToMono() . . . 109
13.5.2.8 StereoToMono_Planar() . . . 110
13.5.2.9 UTF8ToWString() . . . 110
13.5.2.10 WStringToUTF8() . . . 110
13.6 sound4::helper Namespace Reference . . . 111
13.6.1 Detailed Description . . . 111
13.6.2 Function Documentation . . . 111
13.6.2.1 UTF8ToWString() . . . 111
13.6.2.2 WStringToUTF8() . . . 111

14 Class Documentation **113**
14.1 sound4::CBus Class Reference . . . 113
14.1.1 Detailed Description . . . 113
14.1.2 Constructor & Destructor Documentation . . . 113
14.1.2.1 CBus() . . . 113
14.1.2.2 ~CBus() . . . 114
14.1.3 Member Function Documentation . . . 114
14.1.3.1 Get() . . . 114
14.1.4 Friends And Related Symbol Documentation . . . 114
14.1.4.1 CInstance . . . 114
14.2 sound4::CInstance::CClient Class Reference . . . 114
14.2.1 Detailed Description . . . 114
14.2.2 Constructor & Destructor Documentation . . . 115
14.2.2.1 CClient() [1/2] . . . 115
14.2.2.2 CClient() [2/2] . . . 115
14.2.2.3 ~CClient() . . . 115
14.2.3 Member Function Documentation . . . 115
14.2.3.1 operator=() . . . 115
14.2.3.2 ProcessJson() . . . 115
14.3 sound4::cloudx1::CInstance::CClient Class Reference . . . 116
14.3.1 Detailed Description . . . 116
14.3.2 Constructor & Destructor Documentation . . . 116
14.3.2.1 CClient() [1/2] . . . 116

14.3.2.2 CClient() [2/2] . . . 116
14.3.2.3 ~CClient() . . . 116
14.3.3 Member Function Documentation . . . 116
14.3.3.1 operator=() . . . 116
14.3.3.2 ProcessJson() . . . 116
14.4 sound4::cloudx1::dyn::CInstance::CClient Class Reference . . . 117
14.4.1 Detailed Description . . . 117
14.4.2 Constructor & Destructor Documentation . . . 117
14.4.2.1 CClient() [1/2] . . . 117
14.4.2.2 CClient() [2/2] . . . 118
14.4.2.3 ~CClient() . . . 118
14.4.3 Member Function Documentation . . . 118
14.4.3.1 operator=() . . . 118
14.4.3.2 ProcessJson() . . . 118
14.5 sound4::cloudx1::dyn::CDynLib Class Reference . . . 118
14.5.1 Detailed Description . . . 120
14.5.2 Constructor & Destructor Documentation . . . 120
14.5.2.1 CDynLib() . . . 120
14.5.3 Member Function Documentation . . . 120
14.5.3.1 IsOk() . . . 120
14.5.3.2 Load() . . . 120
14.5.4 Member Data Documentation . . . 121
14.5.4.1 AddAudio . . . 121
14.5.4.2 AddPadAudio . . . 121
14.5.4.3 AudioConvertFrom . . . 121
14.5.4.4 AudioConvertTo . . . 121
14.5.4.5 AudioMonoFromLiveStereo . . . 121
14.5.4.6 AudioMonoToLiveStereo . . . 122
14.5.4.7 DeleteClient . . . 122
14.5.4.8 ExitProcess . . . 122
14.5.4.9 FreeJsonAnswer . . . 122
14.5.4.10 FreeMetadataInfos . . . 122
14.5.4.11 FreeParameters . . . 122
14.5.4.12 FreeParameterValue . . . 122
14.5.4.13 GetAudio . . . 123
14.5.4.14 GetAudioInputCount . . . 123
14.5.4.15 GetAudioOutputCount . . . 123
14.5.4.16 GetBufferIn . . . 123
14.5.4.17 GetBufferOut . . . 123
14.5.4.18 GetBytesFromFormat . . . 123
14.5.4.19 GetChannelCount . . . 123
14.5.4.20 GetChunkSizeInFrames . . . 124

14.5.4.21 GetEstimatedDelay . . . 124
14.5.4.22 GetFormatFromName . . . 124
14.5.4.23 GetFormatName . . . 124
14.5.4.24 GetMaxPacketFrame . . . 124
14.5.4.25 GetMetadataInfos . . . 124
14.5.4.26 GetOutputCount . . . 124
14.5.4.27 GetParameter . . . 125
14.5.4.28 GetPossibleChunkSizeInFrames . . . 125
14.5.4.29 GetProcessChunkFrames . . . 125
14.5.4.30 GetSampleRate . . . 125
14.5.4.31 GetVersion . . . 125
14.5.4.32 InitProcess . . . 125
14.5.4.33 InitProcess2 . . . 125
14.5.4.34 InitProcess3 . . . 126
14.5.4.35 MonoToStereo . . . 126
14.5.4.36 MonoToStereo_Planar . . . 126
14.5.4.37 NewClient . . . 126
14.5.4.38 NewParameters . . . 126
14.5.4.39 PresetManager_InformChange . . . 126
14.5.4.40 ProcessAudio . . . 126
14.5.4.41 ProcessAudio_Planar . . . 127
14.5.4.42 ProcessJson . . . 127
14.5.4.43 SaveState . . . 127
14.5.4.44 SetInstanceTracing . . . 127
14.5.4.45 SetInstanceTracingProcessActivity . . . 127
14.5.4.46 SetLoggerCallback . . . 127
14.5.4.47 SetLogSeverity . . . 127
14.5.4.48 SetMetadata . . . 128
14.5.4.49 SetMetadataMulti . . . 128
14.5.4.50 SetParameter . . . 128
14.5.4.51 SetPresetManager . . . 128
14.5.4.52 StartUpdateThread . . . 128
14.5.4.53 StereoToMono . . . 128
14.5.4.54 StereoToMono_Planar . . . 128
14.5.4.55 StopUpdateThread . . . 129
14.5.4.56 TerminateProcess . . . 129
14.5.4.57 WaitUpdateThreadReady . . . 129
14.5.4.58 Webserver . . . 129
14.5.4.59 Webserver_FreeString . . . 129
14.5.4.60 Webserver_GetAppHealth . . . 129
14.5.4.61 Webserver_SetAppHealth . . . 129
14.5.4.62 Webserver_Status . . . 130

14.5.4.63 Webserver_Stop . . . 130
14.5.4.64 Webserver_tcp . . . 130
14.5.4.65 Webserver_tcp2 . . . 130
14.6 sound4::cloudx1::dyn::helper::CDynLoader Class Reference . . . 130
14.6.1 Detailed Description . . . 131
14.6.2 Constructor & Destructor Documentation . . . 131
14.6.2.1 CDynLoader() [1/3] . . . 131
14.6.2.2 CDynLoader() [2/3] . . . 131
14.6.2.3 CDynLoader() [3/3] . . . 131
14.6.2.4 ~CDynLoader() . . . 131
14.6.3 Member Function Documentation . . . 131
14.6.3.1 Close() . . . 131
14.6.3.2 GetSymbol() . . . 131
14.6.3.3 GetSymbol_safe() . . . 132
14.6.3.4 GetThisLibraryPath() . . . 132
14.6.3.5 IsOk() . . . 132
14.6.3.6 Load() . . . 132
14.6.3.7 operator bool() . . . 132
14.6.3.8 operator=() . . . 132
14.7 sound4::helper::CDynLoader Class Reference . . . 133
14.7.1 Detailed Description . . . 133
14.7.2 Constructor & Destructor Documentation . . . 133
14.7.2.1 CDynLoader() [1/3] . . . 133
14.7.2.2 CDynLoader() [2/3] . . . 133
14.7.2.3 CDynLoader() [3/3] . . . 133
14.7.2.4 ~CDynLoader() . . . 134
14.7.3 Member Function Documentation . . . 134
14.7.3.1 Close() . . . 134
14.7.3.2 GetSymbol() . . . 134
14.7.3.3 GetSymbol_safe() . . . 134
14.7.3.4 GetThisLibraryPath() . . . 134
14.7.3.5 IsOk() . . . 134
14.7.3.6 Load() . . . 135
14.7.3.7 operator bool() . . . 135
14.7.3.8 operator=() . . . 135
14.8 sound4::CInstance Class Reference . . . 135
14.8.1 Detailed Description . . . 136
14.8.2 Constructor & Destructor Documentation . . . 137
14.8.2.1 CInstance() [1/3] . . . 137
14.8.2.2 CInstance() [2/3] . . . 137
14.8.2.3 CInstance() [3/3] . . . 137
14.8.2.4 ~CInstance() . . . 137

14.8.3 Member Function Documentation . . . 138
14.8.3.1 AddAudio() . . . 138
14.8.3.2 Create() . . . 138
14.8.3.3 GetAudio() . . . 139
14.8.3.4 GetBufferIn() . . . 139
14.8.3.5 GetBufferOut() . . . 139
14.8.3.6 GetChunkFrames() . . . 140
14.8.3.7 GetEstimatedDelay() . . . 140
14.8.3.8 GetMetadataInfos() . . . 140
14.8.3.9 GetParam() . . . 140
14.8.3.10 GetWebServerAppHealth() . . . 141
14.8.3.11 IsOk() . . . 141
14.8.3.12 NewClient() . . . 141
14.8.3.13 OnUpdateThreadStart() . . . 141
14.8.3.14 OnUpdateThreadStop() . . . 142
14.8.3.15 operator=() . . . 142
14.8.3.16 PresetManager_InformChange() . . . 142
14.8.3.17 ProcessAnyAudio() . . . 142
14.8.3.18 ProcessAudio() . . . 143
14.8.3.19 ProcessAudio_Planar() . . . 143
14.8.3.20 SaveState() . . . 143
14.8.3.21 SetBus() . . . 144
14.8.3.22 SetInstanceTracing() . . . 144
14.8.3.23 SetInstanceTracingProcessActivity() . . . 144
14.8.3.24 SetMetadata() . . . 145
14.8.3.25 SetMetadataMulti() . . . 145
14.8.3.26 SetParam() . . . 145
14.8.3.27 SetPresetManager() . . . 146
14.8.3.28 SetWebServerAppHealth() . . . 146
14.8.3.29 StartWebServer() . . . 146
14.8.3.30 Stop() . . . 147
14.8.3.31 StopWebServer() . . . 147
14.8.3.32 Terminate() . . . 147
14.9 sound4::cloudx1::CInstance Class Reference . . . 148
14.9.1 Detailed Description . . . 149
14.9.2 Constructor & Destructor Documentation . . . 149
14.9.2.1 CInstance() [1/2] . . . 149
14.9.2.2 CInstance() [2/2] . . . 149
14.9.2.3 ~CInstance() . . . 149
14.9.3 Member Function Documentation . . . 149
14.9.3.1 AddAudio() . . . 149
14.9.3.2 Create() . . . 150

14.9.3.3 GetAudio() . . . 150
14.9.3.4 GetBufferIn() . . . 151
14.9.3.5 GetBufferOut() . . . 151
14.9.3.6 GetChunkFrames() . . . 151
14.9.3.7 GetEstimatedDelay() . . . 152
14.9.3.8 GetMetadataInfos() . . . 152
14.9.3.9 GetParam() . . . 152
14.9.3.10 GetWebServerAppHealth() . . . 153
14.9.3.11 IsOk() . . . 153
14.9.3.12 NewClient() . . . 153
14.9.3.13 OnUpdateThreadStart() . . . 153
14.9.3.14 OnUpdateThreadStop() . . . 153
14.9.3.15 operator=() . . . 153
14.9.3.16 PresetManager_InformChange() . . . 153
14.9.3.17 ProcessAnyAudio() . . . 154
14.9.3.18 ProcessAudio() . . . 154
14.9.3.19 ProcessAudio_Planar() . . . 155
14.9.3.20 SaveState() . . . 156
14.9.3.21 SetInstanceTracing() . . . 156
14.9.3.22 SetInstanceTracingProcessActivity() . . . 156
14.9.3.23 SetMetadata() . . . 157
14.9.3.24 SetMetadataMulti() . . . 157
14.9.3.25 SetParam() . . . 157
14.9.3.26 SetPresetManager() . . . 158
14.9.3.27 SetWebServerAppHealth() . . . 158
14.9.3.28 StartWebServer() . . . 158
14.9.3.29 Stop() . . . 159
14.9.3.30 StopWebServer() . . . 159
14.9.3.31 Terminate() . . . 159
14.10 sound4::cloudx1::dyn::CInstance Class Reference . . . 160
14.10.1 Detailed Description . . . 161
14.10.2 Constructor & Destructor Documentation . . . 161
14.10.2.1 CInstance() [1/2] . . . 161
14.10.2.2 CInstance() [2/2] . . . 161
14.10.2.3 ~CInstance() . . . 161
14.10.3 Member Function Documentation . . . 161
14.10.3.1 AddAudio() . . . 161
14.10.3.2 Create() . . . 162
14.10.3.3 GetAudio() . . . 162
14.10.3.4 GetBufferIn() . . . 163
14.10.3.5 GetBufferOut() . . . 163
14.10.3.6 GetChunkFrames() . . . 163

14.10.3.7 GetEstimatedDelay() . . . 164
14.10.3.8 GetMetadataInfos() . . . 164
14.10.3.9 GetParam() . . . 164
14.10.3.10 GetWebServerAppHealth() . . . 165
14.10.3.11 IsOk() . . . 165
14.10.3.12 NewClient() . . . 165
14.10.3.13 OnUpdateThreadStart() . . . 165
14.10.3.14 OnUpdateThreadStop() . . . 165
14.10.3.15 operator=() . . . 165
14.10.3.16 PresetManager_InformChange() . . . 165
14.10.3.17 ProcessAnyAudio() . . . 166
14.10.3.18 ProcessAudio() . . . 166
14.10.3.19 ProcessAudio_Planar() . . . 167
14.10.3.20 SaveState() . . . 168
14.10.3.21 SetInstanceTracing() . . . 168
14.10.3.22 SetInstanceTracingProcessActivity() . . . 168
14.10.3.23 SetMetadata() . . . 169
14.10.3.24 SetMetadataMulti() . . . 169
14.10.3.25 SetParam() . . . 169
14.10.3.26 SetPresetManager() . . . 170
14.10.3.27 SetWebServerAppHealth() . . . 170
14.10.3.28 StartWebServer() . . . 170
14.10.3.29 Stop() . . . 171
14.10.3.30 StopWebServer() . . . 171
14.10.3.31 Terminate() . . . 171
14.11 cloudx1_CClientInstance Struct Reference . . . 172
14.11.1 Detailed Description . . . 172
14.12 cloudx1_CInstance Struct Reference . . . 172
14.12.1 Detailed Description . . . 172
14.13 cloudx1_CParameters Struct Reference . . . 172
14.13.1 Detailed Description . . . 172
14.14 sound4::cloudx1::CPresetLoader Class Reference . . . 172
14.14.1 Detailed Description . . . 173
14.14.2 Constructor & Destructor Documentation . . . 173
14.14.2.1 CPresetLoader() . . . 173
14.14.2.2 ~CPresetLoader() . . . 173
14.14.3 Member Function Documentation . . . 173
14.14.3.1 Exists() . . . 173
14.14.3.2 GetAll() . . . 173
14.14.3.3 IsReadOnly() . . . 173
14.14.3.4 Read() . . . 174
14.14.3.5 Remove() . . . 174

14.14.3.6 Rename() . . . 174
14.14.3.7 Write() . . . 174
14.15 sound4::cloudx1::dyn::CPresetLoader Class Reference . . . 174
14.15.1 Detailed Description . . . 175
14.15.2 Constructor & Destructor Documentation . . . 175
14.15.2.1 CPresetLoader() . . . 175
14.15.2.2 ~CPresetLoader() . . . 175
14.15.3 Member Function Documentation . . . 175
14.15.3.1 Exists() . . . 175
14.15.3.2 GetAll() . . . 175
14.15.3.3 IsReadOnly() . . . 175
14.15.3.4 Read() . . . 175
14.15.3.5 Remove() . . . 176
14.15.3.6 Rename() . . . 176
14.15.3.7 Write() . . . 176
14.16 sound4::CPresetLoader Class Reference . . . 176
14.16.1 Detailed Description . . . 177
14.16.2 Constructor & Destructor Documentation . . . 177
14.16.2.1 CPresetLoader() . . . 177
14.16.2.2 ~CPresetLoader() . . . 177
14.16.3 Member Function Documentation . . . 177
14.16.3.1 Exists() . . . 177
14.16.3.2 GetAll() . . . 177
14.16.3.3 IsReadOnly() . . . 177
14.16.3.4 Read() . . . 177
14.16.3.5 Remove() . . . 178
14.16.3.6 Rename() . . . 178
14.16.3.7 Write() . . . 178
14.17 sound4::CProcessor Class Reference . . . 178
14.17.1 Detailed Description . . . 179
14.17.2 Constructor & Destructor Documentation . . . 179
14.17.2.1 CProcessor() . . . 179
14.17.3 Member Function Documentation . . . 179
14.17.3.1 GetAudioInputCount() . . . 179
14.17.3.2 GetAudioOutputCount() . . . 180
14.17.3.3 GetBytesFromFormat() . . . 180
14.17.3.4 GetChannelCount() . . . 180
14.17.3.5 GetChunkSizeInFrames() . . . 181
14.17.3.6 GetFormatFromName() . . . 181
14.17.3.7 GetFormatName() . . . 181
14.17.3.8 GetPossibleChunkSizeInFrames() . . . 182
14.17.3.9 GetPrefixSymbol() . . . 182

14.17.3.10 GetSampleRate() . . . 182
14.17.3.11 GetVersion() . . . 183
14.17.3.12 IsOk() . . . 183
14.17.3.13 Load() . . . 183
14.17.3.14 SetLoggerCallback() . . . 183
14.17.3.15 SetLogSeverity() . . . 184
14.17.4 Friends And Related Symbol Documentation . . . 184
14.17.4.1 CBus . . . 184
14.17.4.2 CInstance . . . 184
14.18 sound4::cloudx1::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 184
14.18.1 Detailed Description . . . 184
14.18.2 Constructor & Destructor Documentation . . . 185
14.18.2.1 DynFuncHolder() [1/2] . . . 185
14.18.2.2 DynFuncHolder() [2/2] . . . 185
14.18.3 Member Function Documentation . . . 185
14.18.3.1 IsOk() . . . 185
14.18.3.2 operator bool() . . . 185
14.18.3.3 operator T*() . . . 185
14.19 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 186
14.19.1 Detailed Description . . . 186
14.19.2 Constructor & Destructor Documentation . . . 186
14.19.2.1 DynFuncHolder() [1/2] . . . 186
14.19.2.2 DynFuncHolder() [2/2] . . . 186
14.19.3 Member Function Documentation . . . 186
14.19.3.1 IsOk() . . . 186
14.19.3.2 operator bool() . . . 187
14.19.3.3 operator T*() . . . 187
14.20 MyPresetManager Class Reference . . . 187
14.20.1 Detailed Description . . . 188
14.20.2 Constructor & Destructor Documentation . . . 188
14.20.2.1 MyPresetManager() [1/2] . . . 188
14.20.2.2 ~MyPresetManager() [1/2] . . . 188
14.20.2.3 MyPresetManager() [2/2] . . . 188
14.20.2.4 ~MyPresetManager() [2/2] . . . 188
14.20.3 Member Function Documentation . . . 188
14.20.3.1 Exists() [1/2] . . . 188
14.20.3.2 Exists() [2/2] . . . 189
14.20.3.3 GetAll() [1/2] . . . 189
14.20.3.4 GetAll() [2/2] . . . 189
14.20.3.5 IsReadOnly() [1/2] . . . 189
14.20.3.6 IsReadOnly() [2/2] . . . 189
14.20.3.7 Read() [1/2] . . . 189

14.20.3.8 Read() [2/2] . . . 190
14.20.3.9 Remove() [1/2] . . . 190
14.20.3.10 Remove() [2/2] . . . 190
14.20.3.11 Rename() [1/2] . . . 190
14.20.3.12 Rename() [2/2] . . . 190
14.20.3.13 SetupPath() [1/2] . . . 191
14.20.3.14 SetupPath() [2/2] . . . 191
14.20.3.15 Write() [1/2] . . . 191
14.20.3.16 Write() [2/2] . . . 191
14.21 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference . . . 191
14.21.1 Detailed Description . . . 191
14.22 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference . . . 192
14.22.1 Detailed Description . . . 192
14.23 sound4::helper::SampleFormat< T > Struct Template Reference . . . 192
14.23.1 Detailed Description . . . 192
14.24 SOUND4_ProcessInfo Struct Reference . . . 193
14.24.1 Detailed Description . . . 193
14.24.2 Member Data Documentation . . . 193
14.24.2.1 name . . . 193
14.24.2.2 prefix . . . 194
14.24.2.3 shortname . . . 194
14.24.2.4 struct_size . . . 194
14.24.2.5 version . . . 194

15 File Documentation **195**

15.1 doc_examples.h File Reference . . . 195
15.2 doc_examples.h . . . 195
15.3 doc_gencpp.h File Reference . . . 195
15.4 doc_gencpp.h . . . 195
15.5 doc_json.h File Reference . . . 195
15.6 doc_json.h . . . 195
15.7 doc_main.h File Reference . . . 196
15.8 doc_main.h . . . 196
15.9 doc_params.h File Reference . . . 196
15.10 doc_params.h . . . 196
15.11 sound4.x1.cloud-pipe-cxx.cpp File Reference . . . 196
15.11.1 Function Documentation . . . 196
15.11.1.1 GetEnv() . . . 196
15.11.1.2 main() . . . 197
15.11.1.3 MyLogger() . . . 197
15.11.1.4 ProcessLoop() . . . 197
15.11.1.5 Usage() . . . 197

15.12 sound4.x1.cloud-pipe-cxx.cpp . . . 198
15.13 sound4.x1.cloud-pipe-dyn.cpp File Reference . . . 200
15.13.1 Function Documentation . . . 200
15.13.1.1 GetEnv() . . . 200
15.13.1.2 main() . . . 201
15.13.1.3 MyLogger() . . . 201
15.13.1.4 ProcessLoop() . . . 201
15.13.1.5 Usage() . . . 201
15.14 sound4.x1.cloud-pipe-dyn.cpp . . . 202
15.15 sound4.x1.cloud-pipe.cpp File Reference . . . 205
15.15.1 Macro Definition Documentation . . . 205
15.15.1.1 CHANNEL_COUNT . . . 205
15.15.1.2 CHUNK_SIZE . . . 205
15.15.1.3 SAMPLE_RATE . . . 206
15.15.2 Function Documentation . . . 206
15.15.2.1 main() . . . 206
15.15.2.2 MyLogger() . . . 206
15.15.2.3 ProcessLoop() . . . 206
15.15.2.4 Usage() . . . 206
15.16 sound4.x1.cloud-pipe.cpp . . . 207
15.17 sound4cl-pipe.cpp File Reference . . . 210
15.17.1 Function Documentation . . . 210
15.17.1.1 GetEnv() . . . 210
15.17.1.2 main() . . . 211
15.17.1.3 ProcessLoop() . . . 211
15.18 sound4cl-pipe.cpp . . . 212
15.19 sound4.x1.cloud.h File Reference . . . 215
15.19.1 Macro Definition Documentation . . . 217
15.19.1.1 CLOUDX1_DECL . . . 217
15.19.1.2 CLOUDX1_HAS_CLOUDBUS . . . 218
15.19.1.3 CLOUDX1_HAS_WEBSERVER . . . 218
15.19.2 Typedef Documentation . . . 218
15.19.2.1 TraceLoggingHProvider . . . 218
15.20 sound4.x1.cloud.h . . . 218
15.21 sound4.x1.cloud.hpp File Reference . . . 222
15.21.1 Function Documentation . . . 224
15.21.1.1 cloudx1_custom_exists() . . . 224
15.21.1.2 cloudx1_custom_getall() . . . 224
15.21.1.3 cloudx1_custom_getall_free() . . . 225
15.21.1.4 cloudx1_custom_reader() . . . 225
15.21.1.5 cloudx1_custom_reader_free() . . . 225
15.21.1.6 cloudx1_custom_remove() . . . 225

15.21.1.7 cloudx1_custom_rename() . . . 225
15.21.1.8 cloudx1_custom_writer() . . . 225
15.22 sound4.x1.cloud.hpp . . . 226
15.23 sound4.x1.cloud_dyn.hpp File Reference . . . 233
15.23.1 Detailed Description . . . 236
15.23.2 Macro Definition Documentation . . . 236
15.23.2.1 UNIXLIKE . . . 236
15.23.3 Function Documentation . . . 236
15.23.3.1 cloudx1_custom_exists() . . . 236
15.23.3.2 cloudx1_custom_getall() . . . 236
15.23.3.3 cloudx1_custom_getall_free() . . . 236
15.23.3.4 cloudx1_custom_reader() . . . 236
15.23.3.5 cloudx1_custom_reader_free() . . . 237
15.23.3.6 cloudx1_custom_remove() . . . 237
15.23.3.7 cloudx1_custom_rename() . . . 237
15.23.3.8 cloudx1_custom_writer() . . . 237
15.24 sound4.x1.cloud_dyn.hpp . . . 237
15.25 sound4cl.hpp File Reference . . . 251
15.25.1 Detailed Description . . . 253
15.25.2 Macro Definition Documentation . . . 253
15.25.2.1 UNIXLIKE . . . 253
15.26 sound4cl.hpp . . . 253
15.27 sound4cl_cdef.h File Reference . . . 267
15.27.1 Macro Definition Documentation . . . 267
15.27.1.1 SOUND4_INVALID_WEBSERVER_ID . . . 267
15.27.1.2 SOUND4_WEBSERVER_HTTP_OK . . . 267
15.27.1.3 SOUND4_WEBSERVER_HTTPS_OK . . . 267
15.27.2 Typedef Documentation . . . 268
15.27.2.1 fs_char . . . 268
15.27.2.2 TraceLoggingHProvider . . . 268
15.28 sound4cl_cdef.h . . . 268

Index **271**

Chapter 1

SOUND4 x1.CLOUD 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 cloudx1_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 cloudx1_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 x1.Cloud version <= 1.3.7, the state resetted also for filtered out objects. So avoid alterning `object` filter between calls, or you will miss some changes.

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

load the preset

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

save the preset.
If null is given, save onair preset

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

delete the preset.
can also be an array of presetnames

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

Create a preset from factory and load it

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

Rename a preset

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

Import a preset
```
{ "export" : "`presetname`"  }
```

Export a preset: get preset's content so it can be imported later
```
{ "new" : { "name": "`presetname`", "factory": "`presetname`" } }
```

Create a new preset from a factory.
Can use array for multiple.
```
{ "undo" : `count`  }
```

Undo last(s) `count` preset change
```
{ "redo" : `count`  }
```

Redo last(s) `count` preset change

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.

Note

In x1.Cloud version <= 1.2.2, the `OK` and `ERROR` answers had the syntax
```
{ "OK" : "original command" }
{ "ERROR" : "faulty command" }
```

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 CLOUDX1_AUDIO_INPUT_COUNT

For binary compatibility, this should be runtime checked to be the same as cloudx1_GetAudioInputCount()

Member CLOUDX1_AUDIO_OUTPUT_COUNT

For binary compatibility, this should be runtime checked to be the same as cloudx1_GetAudioOutputCount()

Member CLOUDX1_AUDIOFRAME_COUNT

For binary compatibility, this should be runtime checked to be the same as cloudx1_GetChunkSizeInFrames()

Member CLOUDX1_CHANNEL_COUNT

For binary compatibility, this should be runtime checked to be the same as cloudx1_GetChannelCount()

Member cloudx1_GetChunkSizeInFrames ()

Use cloudx1_GetPossibleChunkSizeInFrames

Member cloudx1_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)

Use cloudx1_InitProcess3

Member cloudx1_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters)

Use cloudx1_InitProcess3

Member CLOUDX1_SAMPLE_RATE

For binary compatibility, this should be runtime checked to be the same as cloudx1_GetSampleRate()

Member cloudx1_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)

Use cloudx1_Webserver_tcp2

Member sound4::cloudx1::ChunkFrames

use GetPossibleChunkSizeInFrames

Member sound4::cloudx1::dyn::ChunkFrames

use GetPossibleChunkSizeInFrames()

Member sound4::cloudx1::dyn::GetChunkSizeInFrames (CDynLib &dynlib)

Use GetPossibleChunkSizeInFrames

Member sound4::cloudx1::dyn::InputChannels

Use GetChannelCount and GetAudioInputCount

Member sound4::cloudx1::dyn::InputSampleSize

Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Member sound4::cloudx1::dyn::OutputChannels

Use GetChannelCount and GetAudioOutputCount

Member sound4::cloudx1::dyn::OutputSampleSize

Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Member sound4::cloudx1::dyn::SampleRate

Use GetSampleRate

Member sound4::cloudx1::GetChunkSizeInFrames ()

Use GetPossibleChunkSizeInFrames

Member sound4::cloudx1::InputChannels

Use GetChannelCount and GetAudioInputCount

Member sound4::cloudx1::InputSampleSize

Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Member sound4::cloudx1::OutputChannels

Use GetChannelCount and GetAudioOutputCount

Member sound4::cloudx1::OutputSampleSize

Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Member sound4::cloudx1::SampleRate

Use GetSampleRate

Member sound4::CProcessor::GetChunkSizeInFrames ()

Use GetPossibleChunkSizeInFrames

Chapter 7

Topic Index

7.1 Topics

Here is a list of all topics with brief descriptions:

Audio convert helpers 29
Audio convert helpers (C++) 33
Audio managing (any audio) 37
Client's JSon config interfacing 41
Custom preset managagement 44
Direct processing 49
Examples 52
Global setup 53
Instance tracing (Windows only) 59
Metadata Management 60
Process creation/destruction 61
Sample using C interface 66
Sample using C++ interface wrapper 67
Sample using C++ interface wrapper with dynamic library loading 68
Sample using Generic C++ interface (multi-process compatible) 70
Update Management 73
Web server 74

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

sound4 81
sound4::cloudx1 84
sound4::cloudx1::dyn 92
sound4::cloudx1::dyn::helper 101
sound4::cloudx1::helper 106
sound4::helper 111

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

sound4::CBus . . . 113
sound4::CInstance::CClient . . . 114
sound4::cloudx1::CInstance::CClient . . . 116
sound4::cloudx1::dyn::CInstance::CClient . . . 117
sound4::cloudx1::dyn::CDynLib . . . 118
sound4::cloudx1::dyn::helper::CDynLoader . . . 130
sound4::helper::CDynLoader . . . 133
sound4::CInstance . . . 135
sound4::cloudx1::CInstance . . . 148
sound4::cloudx1::dyn::CInstance . . . 160
cloudx1_CClientInstance . . . 172
cloudx1_CInstance . . . 172
cloudx1_CParameters . . . 172
sound4::cloudx1::CPresetLoader . . . 172
sound4::cloudx1::dyn::CPresetLoader . . . 174
MyPresetManager . . . 187
sound4::CPresetLoader . . . 176
MyPresetManager . . . 187
sound4::CProcessor . . . 178
sound4::cloudx1::dyn::helper::DynFuncHolder< T > . . . 184
sound4::helper::DynFuncHolder< T > . . . 186
sound4::cloudx1::dyn::helper::SampleFormat< T > . . . 191
sound4::cloudx1::helper::SampleFormat< T > . . . 192
sound4::helper::SampleFormat< T > . . . 192
SOUND4_ProcessInfo . . . 193

Chapter 10

Class Index

10.1 Class List

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

sound4::CBus
Shared Bus . . . 113
sound4::CInstance::CClient . . . 114
sound4::cloudx1::CInstance::CClient . . . 116
sound4::cloudx1::dyn::CInstance::CClient . . . 117
sound4::cloudx1::dyn::CDynLib
Dynamic library interface . . . 118
sound4::cloudx1::dyn::helper::CDynLoader
Helper to load dynamic library . . . 130
sound4::helper::CDynLoader
Helper to load dynamic library . . . 133
sound4::CInstance
Instance handling class . . . 135
sound4::cloudx1::CInstance
Instance handling class . . . 148
sound4::cloudx1::dyn::CInstance
Instance handling class . . . 160
cloudx1_CClientInstance . . . 172
cloudx1_CInstance . . . 172
cloudx1_CParameters . . . 172
sound4::cloudx1::CPresetLoader
Custom preset handler helper . . . 172
sound4::cloudx1::dyn::CPresetLoader
Custom preset handler helper . . . 174
sound4::CPresetLoader
Custom preset handler helper . . . 176
sound4::CProcessor
Dynamic library interface . . . 178
sound4::cloudx1::dyn::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 184
sound4::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 186
MyPresetManager
[Custom preset manager] . . . 187
sound4::cloudx1::dyn::helper::SampleFormat< T >
Helper for Sample format types . . . 191

sound4::cloudx1::helper::SampleFormat< T >
Helper for Sample format types 192
sound4::helper::SampleFormat< T >
Helper for Sample format types 192
SOUND4_ProcessInfo 193

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

doc_examples.h . . . 195
doc_gencpp.h . . . 195
doc_json.h . . . 195
doc_main.h . . . 196
doc_params.h . . . 196
sound4.x1.cloud-pipe-cxx.cpp
C++ sample using C++ interface wrapper . . . 196
sound4.x1.cloud-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading . . . 200
sound4.x1.cloud-pipe.cpp
C++ sample using only C interface . . . 205
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided . . . 210
sound4.x1.cloud.h
C interface for library . . . 215
sound4.x1.cloud.hpp
C++ wrapper for library . . . 222
sound4.x1.cloud_dyn.hpp
C++ wrapper for dynamic loading library . . . 233
sound4cl.hpp
C++ wrapper for dynamic loading library . . . 251
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries . . . 267

Chapter 12

Topic Documentation

12.1 Audio convert helpers

Helpers to convert audio.

Functions

- void cloudx1_AudioConvertFrom (const uint8_t ∗payload, float ∗output, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_AudioConvertTo (const float ∗input, uint8_t ∗payload, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_StereoToMono (const float ∗input, float ∗output, size_t nFrame)
- void cloudx1_MonoToStereo (const float ∗input, float ∗output, size_t nFrame)
- void cloudx1_StereoToMono_Planar (const float ∗inputL, const float ∗inputR, float ∗output, size_t nFrame)
- void cloudx1_MonoToStereo_Planar (const float ∗input, float ∗outputL, float ∗outputR, size_t nFrame)
- void cloudx1_AudioMonoFromLiveStereo (const uint8_t ∗payload, float ∗output)
- void cloudx1_AudioMonoToLiveStereo (const float ∗input, uint8_t ∗payload)

12.1.1 Detailed Description

Helpers to convert audio in an optimized way

12.1.2 Function Documentation

12.1.2.1 cloudx1_AudioConvertFrom()

```
void cloudx1_AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            enum cloudx1_SampleFormat fmt )
```

Convert the payload to floating-point array

Warning

Output buffer must be 64-byte aligned

Parameters

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

12.1.2.2 cloudx1_AudioConvertTo()

```
void cloudx1_AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            enum cloudx1_SampleFormat fmt )
```

Convert floating-point array to payload

Warning

Input buffer must be 64-byte aligned

Parameters

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

12.1.2.3 cloudx1_AudioMonoFromLiveStereo()

```
void cloudx1_AudioMonoFromLiveStereo (
            const uint8_t * payload,
            float * output )
```

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

Parameters

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

12.1.2.4 cloudx1_AudioMonoToLiveStereo()

```
void cloudx1_AudioMonoToLiveStereo (
            const float * input,
            uint8_t * payload )
```

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

Parameters

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

12.1.2.5 cloudx1_MonoToStereo()

```
void cloudx1_MonoToStereo (
            const float * input,
            float * output,
            size_t nFrame )
```

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

Warning

Buffers must be 64-byte aligned

Parameters

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

12.1.2.6 cloudx1_MonoToStereo_Planar()

```
void cloudx1_MonoToStereo_Planar (
            const float * input,
            float * outputL,
            float * outputR,
            size_t nFrame )
```

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

Warning

Buffers must be 64-byte aligned

Parameters

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

12.1.2.7 cloudx1_StereoToMono()

```
void cloudx1_StereoToMono (
            const float * input,
            float * output,
            size_t nFrame )
```

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

Warning

Buffers must be 64-byte aligned

Parameters

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

12.1.2.8 cloudx1_StereoToMono_Planar()

```
void cloudx1_StereoToMono_Planar (
            const float * inputL,
            const float * inputR,
            float * output,
            size_t nFrame )
```

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

Warning

Buffers must be 64-byte aligned

Parameters

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

12.2 Audio convert helpers (C++)

Helpers to convert audio (C++)

Functions

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

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

12.2.1 Detailed Description

Helpers to convert audio in an optimized way

12.2.2 Function Documentation

12.2.2.1 AudioConvertFrom()

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

Convert the payload to floating-point array

Warning

Output buffer must be 64-byte aligned

Parameters

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

Definition at line 630 of file sound4cl.hpp.

12.2.2.2 AudioConvertTo()

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

Convert floating-point array to payload

Warning

Input buffer must be 64-byte aligned

Parameters

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

Definition at line 643 of file sound4cl.hpp.

12.2.2.3 AudioMonoFromLiveStereo()

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

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

Parameters

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

Definition at line 702 of file sound4cl.hpp.

12.2.2.4 AudioMonoToLiveStereo()

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

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

Parameters

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

Definition at line 711 of file sound4cl.hpp.

12.2.2.5 MonoToStereo()

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

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 667 of file sound4cl.hpp.

12.2.2.6 MonoToStereo_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 693 of file sound4cl.hpp.

12.2.2.7 StereoToMono()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 655 of file sound4cl.hpp.

12.2.2.8 StereoToMono_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 680 of file sound4cl.hpp.

12.3 Audio managing (any audio)

Process audio in any format and size.

Enumerations

- enum cloudx1_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 * cloudx1_GetFormatName (const enum cloudx1_SampleFormat fmt)
- enum cloudx1_SampleFormat cloudx1_GetFormatFromName (const char *name)
- unsigned int cloudx1_GetBytesFromFormat (const enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_GetMaxPacketFrame (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_AddAudio (struct cloudx1_CInstance *instance, const uint8_t *payload, unsigned int nFrame, enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_AddPadAudio (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetOutputCount (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetAudio (struct cloudx1_CInstance *instance, uint8_t *payload, unsigned int max_↩ nFrame, enum cloudx1_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 cloudx1_SampleFormat

`enum cloudx1_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 815 of file sound4.x1.cloud.h.

12.3.3 Function Documentation

12.3.3.1 cloudx1_AddAudio()

```
unsigned int cloudx1_AddAudio (
            struct cloudx1_CInstance * instance,
            const uint8_t * payload,
            unsigned int nFrame,
            enum cloudx1_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 cloudx1_AddPadAudio()

```
unsigned int cloudx1_AddPadAudio (
            struct cloudx1_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 cloudx1_GetAudio()

```
unsigned int cloudx1_GetAudio (
            struct cloudx1_CInstance * instance,
            uint8_t * payload,
            unsigned int max_nFrame,
            enum cloudx1_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 cloudx1_GetBytesFromFormat()

```
unsigned int cloudx1_GetBytesFromFormat (
            const enum cloudx1_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 cloudx1_GetFormatFromName()

```
enum cloudx1_SampleFormat cloudx1_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 cloudx1_GetFormatName()

```
const char * cloudx1_GetFormatName (
            const enum cloudx1_SampleFormat fmt )
```

Convert format to string

Parameters

fmt	the format

Returns

the string name of the format

12.3.3.7 cloudx1_GetMaxPacketFrame()

```
unsigned int cloudx1_GetMaxPacketFrame (
            struct cloudx1_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 cloudx1_GetOutputCount()

```
unsigned int cloudx1_GetOutputCount (
            struct cloudx1_CInstance * instance )
```

Get the available output size

Parameters

instance	the processing instance

Returns

number of available frames at output

12.4 Client's JSon config interfacing

Communicate with the instance.

Classes

- struct cloudx1_CClientInstance

Functions

- struct cloudx1_CClientInstance * cloudx1_NewClient (struct cloudx1_CInstance *instance)
- void cloudx1_DeleteClient (struct cloudx1_CClientInstance *client)
- const char * cloudx1_ProcessJson (struct cloudx1_CClientInstance *client, const char *json_str, int *need↩ _save)
- void cloudx1_FreeJsonAnswer (const char *json_str)
- int cloudx1_SaveState (struct cloudx1_CInstance *instance)

12.4.1 Detailed Description

Allow low-level JSON configuration. see also JSON syntax

12.4.2 Function Documentation

12.4.2.1 cloudx1_DeleteClient()

```
void cloudx1_DeleteClient (
            struct cloudx1_CClientInstance * client )
```

Deletes a client

Parameters

client	the client instance

12.4.2.2 cloudx1_FreeJsonAnswer()

```
void cloudx1_FreeJsonAnswer (
            const char * json_str )
```

Free a JSON answer returned by cloudx1_ProcessJson

Parameters

json_str	the JSON request

12.4.2.3 cloudx1_NewClient()

```
struct cloudx1_CClientInstance * cloudx1_NewClient (
            struct cloudx1_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

12.4.2.4 cloudx1_ProcessJson()

```
const char * cloudx1_ProcessJson (
            struct cloudx1_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 cloudx1_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 cloudx1_FreeJsonAnswer)

12.4.2.5 cloudx1_SaveState()

```
int cloudx1_SaveState (
            struct cloudx1_CInstance * instance )
```

Save the current state

Needed when cloudx1_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

12.5 Custom preset managagement

This allows to provide a customized preset management.

Typedefs

- typedef wchar_t fs_char
- typedef char *(* cloudx1_storage_reader) (const fs_char *filename, void *userdata)
- typedef void(* cloudx1_storage_reader_free) (char *content, void *userdata)
- typedef int(* cloudx1_storage_writer) (const fs_char *filename, const char *content, void *userdata)
- typedef int(* cloudx1_storage_exists) (const fs_char *filename, void *userdata)
- typedef fs_char **(* cloudx1_storage_getall) (void *userdata)
- typedef void(* cloudx1_storage_getall_free) (fs_char **all, void *userdata)
- typedef int(* cloudx1_storage_remove) (const fs_char *filename, void *userdata)
- typedef int(* cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)

Enumerations

- enum cloudx1_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted }

Functions

- void cloudx1_SetPresetManager (struct cloudx1_CParameters *params, cloudx1_storage_reader reader, cloudx1_storage_reader_free, cloudx1_storage_writer writer, cloudx1_storage_exists exists, cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free, cloudx1_storage_remove remove, cloudx1_storage_rename rename, int IsReadOnly, void *userdata)
- void cloudx1_PresetManager_InformChange (struct cloudx1_CInstance *instance, const fs_char *relative↩ _path, enum cloudx1_PresetChange_Kind change_kind)

12.5.1 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 cloudx1_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)

12.5.2 Typedef Documentation

12.5.2.1 cloudx1_storage_exists

```
typedef int(* cloudx1_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 cloudx1_SetPresetManager

Returns

-1 if not exists, 0 if it exists

Definition at line 515 of file sound4.x1.cloud.h.

12.5.2.2 cloudx1_storage_getall

```
typedef fs_char **(* cloudx1_storage_getall) (void *userdata)
```

Function to call to get all stored files' name. Result has to be freed with cloudx1_storage_getall_free callback

Parameters

userdata	the userdata passed to cloudx1_SetPresetManager

Returns

A NULL terminated array of file names

Definition at line 524 of file sound4.x1.cloud.h.

12.5.2.3 cloudx1_storage_getall_free

```
typedef void(* cloudx1_storage_getall_free) (fs_char **all, void *userdata)
```

Function to call to free the result of getall.

Parameters

all	the returned value from cloudx1_storage_getall
userdata	the userdata passed to cloudx1_SetPresetManager

Definition at line 532 of file sound4.x1.cloud.h.

12.5.2.4 cloudx1_storage_reader

```
typedef char *(* cloudx1_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 cloudx1_SetPresetManager

Returns

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

Definition at line 488 of file sound4.x1.cloud.h.

12.5.2.5 cloudx1_storage_reader_free

```
typedef void(* cloudx1_storage_reader_free) (char *content, void *userdata)
```

Function to call to free the result of cloudx1_storage_reader.

Parameters

content	the returned value from cloudx1_storage_reader
userdata	the userdata passed to cloudx1_SetPresetManager

Definition at line 496 of file sound4.x1.cloud.h.

12.5.2.6 cloudx1_storage_remove

```
typedef int(* cloudx1_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 cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 541 of file sound4.x1.cloud.h.

12.5.2.7 cloudx1_storage_rename

```
typedef int(* cloudx1_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 cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 551 of file sound4.x1.cloud.h.

12.5.2.8 cloudx1_storage_writer

```
typedef int(* cloudx1_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 cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 506 of file sound4.x1.cloud.h.

12.5.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 476 of file sound4.x1.cloud.h.

12.5.3 Enumeration Type Documentation

12.5.3.1 cloudx1_PresetChange_Kind

```
enum cloudx1_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.x1.cloud.h.

12.5.4 Function Documentation

12.5.4.1 cloudx1_PresetManager_InformChange()

```
void cloudx1_PresetManager_InformChange (
            struct cloudx1_CInstance * instance,
            const fs_char * relative_path,
            enum cloudx1_PresetChange_Kind change_kind )
```

Inform that a change happened in the storage. This must be used only in conjonction with cloudx1_SetPresetManager

Parameters

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

12.5.4.2 cloudx1_SetPresetManager()

```
void cloudx1_SetPresetManager (
            struct cloudx1_CParameters * params,
            cloudx1_storage_reader reader,
            cloudx1_storage_reader_free ,
            cloudx1_storage_writer writer,
            cloudx1_storage_exists exists,
            cloudx1_storage_getall getall,
            cloudx1_storage_getall_free getall_free,
            cloudx1_storage_remove remove,
            cloudx1_storage_rename rename,
            int IsReadOnly,
            void * userdata )
```

Setup a custom preset manager. This overrides the save_path in cloudx1_InitProcess2

To dynamically update when storage change, see cloudx1_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

12.6 Direct processing

Optimized processing with direct in->out.

Functions

- void cloudx1_ProcessAudio (struct cloudx1_CInstance *instance, const float *input, float *output)
- void cloudx1_ProcessAudio_Planar (struct cloudx1_CInstance *instance, float const *const *input, float *const *output)
- float * cloudx1_GetBufferIn (struct cloudx1_CInstance *instance)
- float * cloudx1_GetBufferOut (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetEstimatedDelay (struct cloudx1_CInstance *instance)

12.6.1 Detailed Description

Optimized processing with direct in->out.

12.6.2 Function Documentation

12.6.2.1 cloudx1_GetBufferIn()

```
float * cloudx1_GetBufferIn (
            struct cloudx1_CInstance * instance )
```

Get preallocated aligned buffer for audio input for cloudx1_ProcessAudio

The buffer is a float[256 * cloudx1_GetAudioInputCount() * cloudx1_GetChannelCount()]

So for this process/version : float[256*1*2]

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.

12.6.2.2 cloudx1_GetBufferOut()

```
float * cloudx1_GetBufferOut (
            struct cloudx1_CInstance * instance )
```

Get preallocated aligned buffer for audio output for cloudx1_ProcessAudio()

The buffer is a float[256 * cloudx1_GetAudioOutputCount() * cloudx1_GetChannelCount()]

So for this process/version : float[256*1*2]

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.

12.6.2.3 cloudx1_GetEstimatedDelay()

```
unsigned int cloudx1_GetEstimatedDelay (
            struct cloudx1_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

12.6.2.4 cloudx1_ProcessAudio()

```
void cloudx1_ProcessAudio (
            struct cloudx1_CInstance * instance,
            const float * input,
            float * output )
```

Process an audio chunk

A chunk contains cloudx1_GetProcessChunkFrames frames, set at init by frames_per_chunk in cloudx1_InitProcess3

Input has cloudx1_GetAudioInputCount() ∗ cloudx1_GetChannelCount() channels

Ouput has cloudx1_GetAudioOutputCount() ∗ cloudx1_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 cloudx1_GetBufferIn
output	audio output buffer (64 byte aligned) or NULL to use internal buffer cloudx1_GetBufferOut

12.6.2.5 cloudx1_ProcessAudio_Planar()

```
void cloudx1_ProcessAudio_Planar (
            struct cloudx1_CInstance * instance,
            float const *const * input,
            float *const * output )
```

Process an audio chunk of planar audio

A chunk contains cloudx1_GetProcessChunkFrames frames, set at init by frames_per_chunk in cloudx1_InitProcess3

Input has cloudx1_GetAudioInputCount() * cloudx1_GetChannelCount() channels

Ouput has cloudx1_GetAudioOutputCount() * cloudx1_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)

12.7 Examples

List all examples.

The C header is sound4.x1.cloud.h.

You can also use the C++ wrappers :

- sound4cl.hpp (using dynamic library loading, compatible with all .CL engines)
- sound4.x1.cloud.hpp
- sound4.x1.cloud_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

12.8 Global setup

Get information on the library.

Classes

- struct SOUND4_ProcessInfo

Macros

- #define CLOUDX1_AUDIOFRAME_COUNT 64
- #define CLOUDX1_CHANNEL_COUNT 2
- #define CLOUDX1_AUDIO_INPUT_COUNT 1
- #define CLOUDX1_AUDIO_OUTPUT_COUNT 1
- #define CLOUDX1_SAMPLE_RATE 48000

Typedefs

- typedef void(∗ cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char ∗)

Enumerations

- enum cloudx1_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 ∗ cloudx1_GetVersion ()
- unsigned int cloudx1_GetChunkSizeInFrames ()
- unsigned int ∗ cloudx1_GetPossibleChunkSizeInFrames ()
- unsigned int cloudx1_GetChannelCount ()
- unsigned int cloudx1_GetAudioInputCount ()
- unsigned int cloudx1_GetAudioOutputCount ()
- unsigned int cloudx1_GetSampleRate ()
- void cloudx1_SetLoggerCallback (cloudx1_loggerfn logger)
- void cloudx1_SetLogSeverity (enum cloudx1_LogSeverity severity)

12.8.1 Detailed Description

Get information on the library. Set logging mode.

12.8.2 Macro Definition Documentation

12.8.2.1 CLOUDX1_AUDIO_INPUT_COUNT

```
#define CLOUDX1_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 cloudx1_GetAudioInputCount()

Definition at line 58 of file sound4.x1.cloud.h.

12.8.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT

```
#define CLOUDX1_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 cloudx1_GetAudioOutputCount()

Definition at line 64 of file sound4.x1.cloud.h.

12.8.2.3 CLOUDX1_AUDIOFRAME_COUNT

```
#define CLOUDX1_AUDIOFRAME_COUNT 64
```

This version number of frames in a process chunk.

Deprecated For binary compatibility, this should be runtime checked to be the same as cloudx1_GetChunkSizeInFrames()

Definition at line 46 of file sound4.x1.cloud.h.

12.8.2.4 CLOUDX1_CHANNEL_COUNT

```
#define CLOUDX1_CHANNEL_COUNT 2
```

This version number of channels.

Deprecated For binary compatibility, this should be runtime checked to be the same as cloudx1_GetChannelCount()

Definition at line 52 of file sound4.x1.cloud.h.

12.8.2.5 CLOUDX1_SAMPLE_RATE

```
#define CLOUDX1_SAMPLE_RATE 48000
```

This input/output sample-rate.

Deprecated For binary compatibility, this should be runtime checked to be the same as cloudx1_GetSampleRate()

Definition at line 70 of file sound4.x1.cloud.h.

12.8.3 Typedef Documentation

12.8.3.1 cloudx1_loggerfn

```
typedef void(* cloudx1_loggerfn) (enum cloudx1_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.x1.cloud.h.

12.8.4 Enumeration Type Documentation

12.8.4.1 cloudx1_LogSeverity

```
enum cloudx1_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.x1.cloud.h.

12.8.5 Function Documentation

12.8.5.1 cloudx1_GetAudioInputCount()

```
unsigned int cloudx1_GetAudioInputCount ( )
```

Get the number of inputs for this process

Remarks

All inputs have cloudx1_GetChannelCount() channels

Fixed to 1, but should be checked for backward compatibility

Returns

The process input count

12.8.5.2 cloudx1_GetAudioOutputCount()

```
unsigned int cloudx1_GetAudioOutputCount ( )
```

Get the number of outputs for this process

Remarks

All outputs have cloudx1_GetChannelCount() channels

Fixed to 1, but should be checked for backward compatibility

Returns

The process outputs count

12.8.5.3 cloudx1_GetChannelCount()

```
unsigned int cloudx1_GetChannelCount ( )
```

Get the number of channels needed by this process

Remarks

Fixed to 2, but should be checked for backward compatibility

Returns

The process channels count

12.8.5.4 cloudx1_GetChunkSizeInFrames()

```
unsigned int cloudx1_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 cloudx1_GetChannelCount samples

Remarks

Fixed to 64, but should be checked for backward compatibility

Returns

The process chunk size in frame

Deprecated Use cloudx1_GetPossibleChunkSizeInFrames

12.8.5.5 cloudx1_GetPossibleChunkSizeInFrames()

```
unsigned int * cloudx1_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 cloudx1_GetChannelCount samples

Returns

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

12.8.5.6 cloudx1_GetSampleRate()

```
unsigned int cloudx1_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

12.8.5.7 cloudx1_GetVersion()

```
const char * cloudx1_GetVersion ( )
```

Get the process version string

Returns

version string

12.8.5.8 cloudx1_SetLoggerCallback()

```
void cloudx1_SetLoggerCallback (
            cloudx1_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.

12.8.5.9 cloudx1_SetLogSeverity()

```
void cloudx1_SetLogSeverity (
            enum cloudx1_LogSeverity severity )
```

Change the global log severity

Logs above this level will not be printed.

Parameters

severity	Log severity

12.8.5.10 SOUND4_GetProcessInfo()

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

Get this processor description

Common to all SOUND4 processors.

Returns

the process description

12.9 Instance tracing (Windows only)

This allows instances to use windows event tracing.

Functions

- void cloudx1_SetInstanceTracing (struct cloudx1_CParameters *params, TraceLoggingHProvider tracing_provider, GUID activity_guid)
- void cloudx1_SetInstanceTracingProcessActivity (struct cloudx1_CInstance *instance, GUID activity_guid)

12.9.1 Detailed Description

This allows instances to use windows event tracing

12.9.2 Function Documentation

12.9.2.1 cloudx1_SetInstanceTracing()

```
void cloudx1_SetInstanceTracing (
            struct cloudx1_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

12.9.2.2 cloudx1_SetInstanceTracingProcessActivity()

```
void cloudx1_SetInstanceTracingProcessActivity (
            struct cloudx1_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

12.10 Metadata Management

Send/update metadata.

Functions

- void cloudx1_SetMetadata (struct cloudx1_CInstance *instance, const char *key, const char *value)
- void cloudx1_SetMetadataMulti (struct cloudx1_CInstance *instance, const char **keyvalue)
- const char ** cloudx1_GetMetadataInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeMetadataInfos (struct cloudx1_CInstance *instance, const char **infos)

12.10.1 Detailed Description

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

12.10.2 Function Documentation

12.10.2.1 cloudx1_FreeMetadataInfos()

```
void cloudx1_FreeMetadataInfos (
            struct cloudx1_CInstance * instance,
            const char ** infos )
```

Frees the information returned by cloudx1_GetMetadataInfos.

Parameters

instance	the instance
infos	the infos

12.10.2.2 cloudx1_GetMetadataInfos()

```
const char ** cloudx1_GetMetadataInfos (
            struct cloudx1_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 cloudx1_FreeMetadataInfos

12.10.2.3 cloudx1_SetMetadata()

```
void cloudx1_SetMetadata (
            struct cloudx1_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

cloudx1_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.

12.10.2.4 cloudx1_SetMetadataMulti()

```
void cloudx1_SetMetadataMulti (
            struct cloudx1_CInstance * instance,
            const char ** keyvalue )
```

Adds multiple metadata to the process.

This works like cloudx1_SetMetadata but with multiple metadata at once.

See also

cloudx1_SetMetadata

Parameters

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

12.11 Process creation/destruction

Create and destroy a process instance.

Classes

- struct cloudx1_CInstance
- struct cloudx1_CParameters

Functions

- struct cloudx1_CParameters * cloudx1_NewParameters ()
- void cloudx1_FreeParameters (struct cloudx1_CParameters *params)
- void cloudx1_SetParameter (struct cloudx1_CParameters *params, const char *name, const char *value)
- const char * cloudx1_GetParameter (struct cloudx1_CParameters *params, const char *name)
- void cloudx1_FreeParameterValue (const char *value)
- struct cloudx1_CInstance * cloudx1_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct cloudx1_CInstance * cloudx1_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters)
- struct cloudx1_CInstance * cloudx1_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters, unsigned int frames_per_chunk)
- unsigned int cloudx1_GetProcessChunkFrames (struct cloudx1_CInstance *instance)
- int cloudx1_TerminateProcess (struct cloudx1_CInstance *instance)
- void cloudx1_ExitProcess (struct cloudx1_CInstance *instance)

12.11.1 Detailed Description

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

12.11.2 Function Documentation

12.11.2.1 cloudx1_ExitProcess()

```
void cloudx1_ExitProcess (
            struct cloudx1_CInstance * instance )
```

Quit and destroy the instance

Parameters

instance	the instance to destroy

12.11.2.2 cloudx1_FreeParameters()

```
void cloudx1_FreeParameters (
            struct cloudx1_CParameters * params )
```

Free the parameters created with cloudx1_NewParameters.

12.11.2.3 cloudx1_FreeParameterValue()

```
void cloudx1_FreeParameterValue (
            const char * value )
```

Free a parameter value returned by cloudx1_GetParameter

Parameters

value	the returned value

12.11.2.4 cloudx1_GetParameter()

```
const char * cloudx1_GetParameter (
            struct cloudx1_CParameters * params,
            const char * name )
```

Get current parameter value.

Returned value must be freed with cloudx1_FreeParameterValue

12.11.2.5 cloudx1_GetProcessChunkFrames()

```
unsigned int cloudx1_GetProcessChunkFrames (
            struct cloudx1_CInstance * instance )
```

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

The value reflects the frames_per_chunk parameter passed to cloudx1_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 cloudx1_GetChannelCount samples

12.11.2.6 cloudx1_InitProcess()

```
struct cloudx1_CInstance * cloudx1_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 cloudx1_InitProcess2, but with default parameters (from env)

Deprecated Use cloudx1_InitProcess3

12.11.2.7 cloudx1_InitProcess2()

```
struct cloudx1_CInstance * cloudx1_InitProcess2 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_Secret,
            const char * save_path,
            const struct cloudx1_CParameters * parameters )
```

Create a process instance with default frames_per_chunk (64)

Same as cloudx1_InitProcess3, but with default frames_per_chunk

Deprecated Use cloudx1_InitProcess3

12.11.2.8 cloudx1_InitProcess3()

```
struct cloudx1_CInstance * cloudx1_InitProcess3 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_Secret,
            const char * save_path,
            const struct cloudx1_CParameters * parameters,
            unsigned int frames_per_chunk )
```

Create a process instance

The output will have to be freed with cloudx1_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 cloudx1_GetPossibleChunkSizeInFrames

Returns

A new process instance

12.11.2.9 cloudx1_NewParameters()

```
struct cloudx1_CParameters * cloudx1_NewParameters ( )
```

Create parameters to use with cloudx1_InitProcess2.

The output will have to be freed with cloudx1_FreeParameters

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

12.11.2.10 cloudx1_SetParameter()

```
void cloudx1_SetParameter (
            struct cloudx1_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

12.11.2.11 cloudx1_TerminateProcess()

```
int cloudx1_TerminateProcess (
            struct cloudx1_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

12.12 Sample using C interface

C example.

See full example cpp/sound4.x1.cloud-pipe.cpp

12.12.1 Sample to create the processing instance

see Process creation/destruction and Update Management

```
    // Setup parameters
    struct cloudx1_CParameters *params = cloudx1_NewParameters();
    // Optional: activate web authentication, set admin password
    // This creates the authentication only for a new storage path
    cloudx1_SetParameter(params,"ADMIN_USER", "admin");
    cloudx1_SetParameter(params,"ADMIN_SECRET", "admin");

    // Create the processing instance
    pInstance=cloudx1_InitProcess2(pLoginKey,pRadioName,pKeyId,pAccessKey,store_path.c_str(), params);
    cloudx1_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 « "cloudx1_StartUpdateThread (no json port)" « std::endl;
       cloudx1_StartUpdateThread(pInstance, 0);
       std::cerr « "cloudx1_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 (cloudx1_WaitUpdateThreadReady(pInstance, 5000)<0) {
        std::cerr « "Timeout to create the process instance"« std::endl;
    }
    std::cerr « "Waiting update thread DONE" « std::endl;

#if (CLOUDX1_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 = cloudx1_Webserver(listenport,listenport_secure,pInstance);
        if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
            std::cerr « "Starting web server FAILED" « std::endl;
        } else {
            int web_status=cloudx1_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 // (CLOUDX1_HAS_WEBSERVER)

     // Force load a factory preset now
     // Could be used instead of setting a store_path
     /*{
         cloudx1_CClientInstance *pClient=cloudx1_NewClient(pInstance);
         const char *answer=cloudx1_ProcessJson(pClient, "{ \"factoryload\": \"All Programs\" }", NULL);
         std::cerr « "Loaded factory preset answer: " « answer« std::endl;
         cloudx1_FreeJsonAnswer(answer);
         cloudx1_DeleteClient(pClient);
     }*/
```

12.12.2 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
        cloudx1_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;
        }
    }
```

12.12.3 Sample to stop the processing instance

see Process creation/destruction

```
#if (CLOUDX1_HAS_WEBSERVER)
    // Stopping the webserver
    std::cerr « "Stopping web server" « std::endl;
    cloudx1_Webserver_Stop(webserver, 1000);
    std::cerr « "Stopping web server DONE" « std::endl;
#endif // (CLOUDX1_HAS_WEBSERVER)

    // Stopping the processing instance
    std::cerr « "Stopping Update thread" « std::endl;
    cloudx1_StopUpdateThread(pInstance); // this will exit the update thread
    std::cerr « "Stopping Update thread DONE" « std::endl;

    updater.join();

    cloudx1_ExitProcess(pInstance);
```

12.13 Sample using C++ interface wrapper

C++ example (static link)

See full example cpp/sound4.x1.cloud-pipe-cxx.cpp

12.13.1 Setup and sanity checks

```
sound4::cloudx1::SetLogSeverity(verbose5);
sound4::cloudx1::SetLoggerCallback(MyLogger);

sound4::cloudx1::CInstance instance;
instance.SetParam("ADMIN_USER","admin");
instance.SetParam("ADMIN_SECRET","admin");
```

12.13.2 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;
}
```

12.13.3 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;
```

12.13.4 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;
}
```

12.13.5 Sample processing loop

```
while (!stopthread)
{
    // Read a chunk from stdin
    if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::cloudx1::InputSampleSize,
  stdin) != sound4::cloudx1::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::cloudx1::OutputSampleSize, stdout)
  != sound4::cloudx1::OutputSampleSize) {
        std::cerr « "Failed to write audio output" « std::endl;
        stopthread=true;
    }
}
```

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

C++ example (dynamic link) See full example cpp/sound4.x1.cloud-pipe-dyn.cpp.

12.14.1 Dynamic loading

```
sound4::cloudx1::dyn::CDynLib dynlib;
if (!dynlib.Load()) {
    std::cerr « "Failed to load processing library" « std::endl;
    return 1;
}
```

12.14.2 Setup and sanity checks

```
sound4::cloudx1::dyn::SetLogSeverity(dynlib, verbose5);
sound4::cloudx1::dyn::SetLoggerCallback(dynlib, MyLogger);

sound4::cloudx1::dyn::CInstance instance(dynlib);
instance.SetParam("ADMIN_USER","admin");
instance.SetParam("ADMIN_SECRET","admin");
```

12.14.3 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;
}
```

12.14.4 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;
```

12.14.5 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;
}
```

12.14.6 Sample processing loop

```
while (!stopthread)
{
    // Read a chunk from stdin
    if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::cloudx1::dyn::InputSampleSize,
  stdin) != sound4::cloudx1::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::cloudx1::dyn::OutputSampleSize,
  stdout) != sound4::cloudx1::dyn::OutputSampleSize) {
        std::cerr « "Failed to write audio output" « std::endl;
        stopthread=true;
    }
}
```

12.14.7 Custom preset manager

```
class MyPresetManager : public sound4::cloudx1::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;
        }
};
```

12.14.8 Custom preset manager loading

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

12.15 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

12.15.1 Loading the library

```
std::string libname =  "libsound4.x1.cloud.so";
sound4::CProcessor processor;
if (!processor.Load(libname)) {
    std::cerr « "Failed to load processing library" « std::endl;
    return 1;
}
```

12.15.2 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");
```

12.15.3 Create the processing instance

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

12.15.4 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;
    }
}
```

12.15.5 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;
}
```

12.15.6 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;
```

12.15.7 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;
        }
};
```

12.15.8 Custom preset manager loading

```
    MyPresetManager preset_manager;
```

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

12.16 Update Management

Manage the process update.

Functions

- void cloudx1_StartUpdateThread (struct cloudx1_CInstance *instance, unsigned int port)
- void cloudx1_StopUpdateThread (struct cloudx1_CInstance *instance)
- int cloudx1_WaitUpdateThreadReady (struct cloudx1_CInstance *instance, int milliseconds)

12.16.1 Detailed Description

Manage the process update and communications.

12.16.2 Function Documentation

12.16.2.1 cloudx1_StartUpdateThread()

```
void cloudx1_StartUpdateThread (
            struct cloudx1_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 cloudx1_StopUpdateThread

Parameters

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

12.16.2.2 cloudx1_StopUpdateThread()

```
void cloudx1_StopUpdateThread (
            struct cloudx1_CInstance * instance )
```

Stop the update thread

Parameters

instance	the processing instance

12.16.2.3 cloudx1_WaitUpdateThreadReady()

```
int cloudx1_WaitUpdateThreadReady (
            struct cloudx1_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

12.17 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 cloudx1_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)
- uint64_t cloudx1_Webserver_tcp2 (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port, const struct cloudx1_CParameters *parameters)
- uint64_t cloudx1_Webserver (unsigned int listenport, unsigned int listenport_secure, struct cloudx1_CInstance *instance)
- int cloudx1_Webserver_Stop (uint64_t id, int timeout_ms)
- int cloudx1_Webserver_Status (uint64_t id)
- void cloudx1_Webserver_SetAppHealth (struct cloudx1_CInstance *instance, int httpcode, const char *contenttype, const char *content)
- void cloudx1_Webserver_GetAppHealth (struct cloudx1_CInstance *instance, int *httpcode, char **contenttype, char **content)
- void cloudx1_Webserver_FreeString (char *str)

12.17.1 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.

See also how to setup authentication with Environment variables and parameters. Their use is documented in cloudx1_InitProcess3.

12.17.2 Macro Definition Documentation

12.17.2.1 SOUND4_INVALID_WEBSERVER_ID

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

This represents an invalid webserver id cloudx1_Webserver and cloudx1_Webserver_tcp will never return this value

Definition at line 1199 of file sound4.x1.cloud.h.

12.17.2.2 SOUND4_WEBSERVER_HTTP_OK

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

If this bit is set in the return value from cloudx1_Webserver_Status, then HTTP server is OK

Definition at line 1204 of file sound4.x1.cloud.h.

12.17.2.3 SOUND4_WEBSERVER_HTTPS_OK

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

If this bit is set in the return value from cloudx1_Webserver_Status, then HTTPS server is OK

Definition at line 1208 of file sound4.x1.cloud.h.

12.17.3 Function Documentation

12.17.3.1 cloudx1_Webserver()

```
uint64_t cloudx1_Webserver (
            unsigned int listenport,
            unsigned int listenport_secure,
            struct cloudx1_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 cloudx1_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 cloudx1_Webserver_Stop

12.17.3.2 cloudx1_Webserver_FreeString()

```
void cloudx1_Webserver_FreeString (
            char * str )
```

Free a a string returned by cloudx1_Webserver_GetAppHealth

Parameters

str	the string to free

12.17.3.3 cloudx1_Webserver_GetAppHealth()

```
void cloudx1_Webserver_GetAppHealth (
            struct cloudx1_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 cloudx1_Webserver_FreeString)
content	The http body content pointer (to be freed with cloudx1_Webserver_FreeString)

12.17.3.4 cloudx1_Webserver_SetAppHealth()

```
void cloudx1_Webserver_SetAppHealth (
            struct cloudx1_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

12.17.3.5 cloudx1_Webserver_Status()

```
int cloudx1_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 cloudx1_Webserver_tcp or cloudx1_Webserver

Returns

a combinaison of SOUND4_WEBSERVER_HTTP_OK and SOUND4_WEBSERVER_HTTPS_OK depending on the server status

12.17.3.6 cloudx1_Webserver_Stop()

```
int cloudx1_Webserver_Stop (
            uint64_t id,
            int timeout_ms )
```

Stop a web server started with cloudx1_Webserver_tcp

Warning

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

Parameters

id	The web server Id, returned by cloudx1_Webserver_tcp or cloudx1_Webserver
timeout_ms	The timeout to wait gracefull shutdown

Returns

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

12.17.3.7 cloudx1_Webserver_tcp()

```
uint64_t cloudx1_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 cloudx1_Webserver_tcp2, but with default parameters (from env)

Deprecated Use cloudx1_Webserver_tcp2

12.17.3.8 cloudx1_Webserver_tcp2()

```
uint64_t cloudx1_Webserver_tcp2 (
            unsigned int listenport,
            unsigned int listenport_secure,
            const char * socket_ip,
            unsigned int socket_port,
            const struct cloudx1_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 cloudx1_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 cloudx1_NewParameters

Returns

id of webserver (starting at 0), to use with cloudx1_Webserver_Stop

Chapter 13

Namespace Documentation

13.1 sound4 Namespace Reference

Namespaces

- namespace cloudx1
- namespace helper

Classes

- class CBus

  Shared Bus.
- class CInstance

  Instance handling class.
- class CPresetLoader

  Custom preset handler helper.
- class CProcessor

  Dynamic library interface.

Typedefs

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

Enumerations

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

Functions

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

Variables

- static log_cb_t _log_cb

13.1.1 Detailed Description

SOUND4 namespace

13.1.2 Typedef Documentation

13.1.2.1 log_cb_t

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

Log callback type

Definition at line 284 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 268 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 883 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 291 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 289 of file sound4cl.hpp.

13.2 sound4::cloudx1 Namespace Reference

Namespaces

- namespace dyn
- namespace helper

Classes

- class CInstance

  Instance handling class.
- class CPresetLoader

  Custom preset handler helper.

Typedefs

- using SampleFormat = cloudx1_SampleFormat
- using LogSeverity = cloudx1_LogSeverity
- using log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using PresetChange_Kind = cloudx1_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 (cloudx1_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 x1.CLOUD"
- static constexpr const char * process_shortname = "x1"
- const size_t ChunkFrames = 64
- const size_t InputChannels = 2 * 1
- const size_t OutputChannels = 2 * 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

cloudx1 namespace

13.2.2 Typedef Documentation

13.2.2.1 log_cb_t

```
using sound4::cloudx1::log_cb_t = typedef std::function<void(LogSeverity,const std::string&)>
```

Log callback type

Definition at line 398 of file sound4.x1.cloud.hpp.

13.2.2.2 LogSeverity

```
using sound4::cloudx1::LogSeverity = typedef cloudx1_LogSeverity
```

Log Severity levels

Definition at line 394 of file sound4.x1.cloud.hpp.

13.2.2.3 PresetChange_Kind

```
using sound4::cloudx1::PresetChange_Kind = typedef cloudx1_PresetChange_Kind
```

Definition at line 464 of file sound4.x1.cloud.hpp.

13.2.2.4 SampleFormat

```
using sound4::cloudx1::SampleFormat = typedef cloudx1_SampleFormat
```

Definition at line 56 of file sound4.x1.cloud.hpp.

13.2.3 Function Documentation

13.2.3.1 _log_cb_c()

```
static void sound4::cloudx1::_log_cb_c (
            cloudx1_LogSeverity severity,
            const char * c_msg )  [inline], [static]
```

Definition at line 405 of file sound4.x1.cloud.hpp.

13.2.3.2 GetAudioInputCount()

```
static unsigned int sound4::cloudx1::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 304 of file sound4.x1.cloud.hpp.

13.2.3.3 GetAudioOutputCount()

```
static unsigned int sound4::cloudx1::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 316 of file sound4.x1.cloud.hpp.

13.2.3.4 GetBytesFromFormat()

```
static unsigned int sound4::cloudx1::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 386 of file sound4.x1.cloud.hpp.

13.2.3.5 GetChannelCount()

```
static unsigned int sound4::cloudx1::GetChannelCount ( )  [inline], [static]
```

Get the number of channels needed by this process

Returns

The process channels count

Definition at line 292 of file sound4.x1.cloud.hpp.

13.2.3.6 GetChunkSizeInFrames()

```
static unsigned int sound4::cloudx1::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 265 of file sound4.x1.cloud.hpp.

13.2.3.7 GetFormatFromName()

```
static SampleFormat sound4::cloudx1::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 375 of file sound4.x1.cloud.hpp.

13.2.3.8 GetFormatName()

```
static std::string sound4::cloudx1::GetFormatName (
            const SampleFormat fmt )  [inline], [static]
```

Convert format to string

Parameters

fmt	the format

Returns

the string name of the format

Definition at line 364 of file sound4.x1.cloud.hpp.

13.2.3.9 GetPossibleChunkSizeInFrames()

```
static std::vector< unsigned int > sound4::cloudx1::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 278 of file sound4.x1.cloud.hpp.

13.2.3.10 GetSampleRate()

```
static unsigned int sound4::cloudx1::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 328 of file sound4.x1.cloud.hpp.

13.2.3.11 GetVersion()

```
static std::string sound4::cloudx1::GetVersion ( )  [inline], [static]
```

Get the process version string

Returns

version string

Definition at line 250 of file sound4.x1.cloud.hpp.

13.2.3.12 SanityCheck()

```
static void sound4::cloudx1::SanityCheck (
            bool a_bCheckFrames = true )  [inline], [static]
```

Checks for library compatibility with const/defines

Throws std::runtime_error on failure

Definition at line 338 of file sound4.x1.cloud.hpp.

13.2.3.13 SetLoggerCallback()

```
static void sound4::cloudx1::SetLoggerCallback (
            log_cb_t cb )  [inline], [static]
```

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

Definition at line 419 of file sound4.x1.cloud.hpp.

13.2.3.14 SetLogSeverity()

```
static void sound4::cloudx1::SetLogSeverity (
            LogSeverity severity )  [inline], [static]
```

Sets the log severity (global)

Definition at line 412 of file sound4.x1.cloud.hpp.

13.2.4 Variable Documentation

13.2.4.1 _log_cb

```
log_cb_t sound4::cloudx1::_log_cb  [static]
```

internal helpers for C log callbacks

Definition at line 403 of file sound4.x1.cloud.hpp.

13.2.4.2 ChunkFrames

```
const size_t sound4::cloudx1::ChunkFrames = 64
```

Number of frames in a chunk

Deprecated use GetPossibleChunkSizeInFrames

Definition at line 61 of file sound4.x1.cloud.hpp.

13.2.4.3 InputChannels

```
const size_t sound4::cloudx1::InputChannels = 2 * 1
```

Number of float values in an input frame

Deprecated Use GetChannelCount and GetAudioInputCount

Definition at line 66 of file sound4.x1.cloud.hpp.

13.2.4.4 InputSampleSize

```
const size_t sound4::cloudx1::InputSampleSize =ChunkFrames*InputChannels
```

Number of float values for an input chunk

Deprecated Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Definition at line 76 of file sound4.x1.cloud.hpp.

13.2.4.5 OutputChannels

```
const size_t sound4::cloudx1::OutputChannels = 2 * 1
```

Number of float values in an output frame

Deprecated Use GetChannelCount and GetAudioOutputCount

Definition at line 71 of file sound4.x1.cloud.hpp.

13.2.4.6 OutputSampleSize

```
const size_t sound4::cloudx1::OutputSampleSize =ChunkFrames*OutputChannels
```

Number of float values for an output chunk

Deprecated Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Definition at line 81 of file sound4.x1.cloud.hpp.

13.2.4.7 process_name

```
constexpr const char* sound4::cloudx1::process_name = "SOUND4 x1.CLOUD"  [static], [constexpr]
```

Definition at line 54 of file sound4.x1.cloud.hpp.

13.2.4.8 process_shortname

```
constexpr const char* sound4::cloudx1::process_shortname = "x1"  [static], [constexpr]
```

Definition at line 55 of file sound4.x1.cloud.hpp.

13.2.4.9 SampleRate

```
const size_t sound4::cloudx1::SampleRate = 48000
```

Sample rate of library

Deprecated Use GetSampleRate

Definition at line 86 of file sound4.x1.cloud.hpp.

13.3 sound4::cloudx1::dyn Namespace Reference

Namespaces

- namespace helper

Classes

- class CDynLib

  Dynamic library interface.
- class CInstance

  Instance handling class.
- class CPresetLoader

  Custom preset handler helper.

Typedefs

- using SampleFormat = cloudx1_SampleFormat
- using LogSeverity = cloudx1_LogSeverity
- using log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using PresetChange_Kind = cloudx1_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 (cloudx1_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 x1.CLOUD"
- static constexpr const char * process_shortname = "x1"
- const size_t ChunkFrames = 64
- const size_t InputChannels = 2 * 1
- const size_t OutputChannels = 2 * 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::cloudx1::dyn::log_cb_t = typedef std::function<void(LogSeverity,const std↩
::string&)>
```

Log callback type

Definition at line 856 of file sound4.x1.cloud_dyn.hpp.

13.3.2.2 LogSeverity

```
using sound4::cloudx1::dyn::LogSeverity = typedef cloudx1_LogSeverity
```

Log Severity levels

Definition at line 852 of file sound4.x1.cloud_dyn.hpp.

13.3.2.3 PresetChange_Kind

```
using sound4::cloudx1::dyn::PresetChange_Kind = typedef cloudx1_PresetChange_Kind
```

Definition at line 930 of file sound4.x1.cloud_dyn.hpp.

13.3.2.4 SampleFormat

```
using sound4::cloudx1::dyn::SampleFormat = typedef cloudx1_SampleFormat
```

Definition at line 490 of file sound4.x1.cloud_dyn.hpp.

13.3.3 Function Documentation

13.3.3.1 _log_cb_c()

```
static void sound4::cloudx1::dyn::_log_cb_c (
            cloudx1_LogSeverity severity,
            const char * c_msg )  [inline], [static]
```

Definition at line 863 of file sound4.x1.cloud_dyn.hpp.

13.3.3.2 GetAudioInputCount()

```
static unsigned int sound4::cloudx1::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 755 of file sound4.x1.cloud_dyn.hpp.

13.3.3.3 GetAudioOutputCount()

```
static unsigned int sound4::cloudx1::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 768 of file sound4.x1.cloud_dyn.hpp.

13.3.3.4 GetBytesFromFormat()

```
static unsigned int sound4::cloudx1::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 844 of file sound4.x1.cloud_dyn.hpp.

13.3.3.5 GetChannelCount()

```
static unsigned int sound4::cloudx1::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 742 of file sound4.x1.cloud_dyn.hpp.

13.3.3.6 GetChunkSizeInFrames()

```
static unsigned int sound4::cloudx1::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 709 of file sound4.x1.cloud_dyn.hpp.

13.3.3.7 GetFormatFromName()

```
static SampleFormat sound4::cloudx1::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 832 of file sound4.x1.cloud_dyn.hpp.

13.3.3.8 GetFormatName()

```
static std::string sound4::cloudx1::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 820 of file sound4.x1.cloud_dyn.hpp.

13.3.3.9 GetPossibleChunkSizeInFrames()

```
static std::vector< unsigned int > sound4::cloudx1::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 723 of file sound4.x1.cloud_dyn.hpp.

13.3.3.10 GetSampleRate()

```
static unsigned int sound4::cloudx1::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 781 of file sound4.x1.cloud_dyn.hpp.

13.3.3.11 GetVersion()

```
static std::string sound4::cloudx1::dyn::GetVersion (
            CDynLib & dynlib )  [inline], [static]
```

Get the process version string

Parameters

dynlib	the dynamic library

Returns

version string

Definition at line 693 of file sound4.x1.cloud_dyn.hpp.

13.3.3.12 SanityCheck()

```
static void sound4::cloudx1::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 793 of file sound4.x1.cloud_dyn.hpp.

13.3.3.13 SetLoggerCallback()

```
static void sound4::cloudx1::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 877 of file sound4.x1.cloud_dyn.hpp.

13.3.3.14 SetLogSeverity()

```
static void sound4::cloudx1::dyn::SetLogSeverity (
            CDynLib & dynlib,
            LogSeverity severity )  [inline], [static]
```

Sets the log severity (global)

Definition at line 870 of file sound4.x1.cloud_dyn.hpp.

13.3.4 Variable Documentation

13.3.4.1 _log_cb

```
log_cb_t sound4::cloudx1::dyn::_log_cb  [static]
```

internal helpers for C log callbacks

Definition at line 861 of file sound4.x1.cloud_dyn.hpp.

13.3.4.2 ChunkFrames

```
const size_t sound4::cloudx1::dyn::ChunkFrames = 64
```

Number of frames in a chunk

Deprecated use GetPossibleChunkSizeInFrames()

Definition at line 495 of file sound4.x1.cloud_dyn.hpp.

13.3.4.3 InputChannels

```
const size_t sound4::cloudx1::dyn::InputChannels = 2 * 1
```

Number of float values in an input frame

Deprecated Use GetChannelCount and GetAudioInputCount

Definition at line 500 of file sound4.x1.cloud_dyn.hpp.

13.3.4.4 InputSampleSize

```
const size_t sound4::cloudx1::dyn::InputSampleSize =ChunkFrames*InputChannels
```

Number of float values for an input chunk

Deprecated Use GetPossibleChunkSizeInFrames and GetAudioInputCount

Definition at line 510 of file sound4.x1.cloud_dyn.hpp.

13.3.4.5 OutputChannels

```
const size_t sound4::cloudx1::dyn::OutputChannels = 2 * 1
```

Number of float values in an output frame

Deprecated Use GetChannelCount and GetAudioOutputCount

Definition at line 505 of file sound4.x1.cloud_dyn.hpp.

13.3.4.6 OutputSampleSize

```
const size_t sound4::cloudx1::dyn::OutputSampleSize =ChunkFrames*OutputChannels
```

Number of float values for an output chunk

Deprecated Use GetPossibleChunkSizeInFrames and GetAudioOutputCount

Definition at line 515 of file sound4.x1.cloud_dyn.hpp.

13.3.4.7 process_name

```
constexpr const char* sound4::cloudx1::dyn::process_name = "SOUND4 x1.CLOUD"  [static], [constexpr]
```

Definition at line 488 of file sound4.x1.cloud_dyn.hpp.

13.3.4.8 process_shortname

```
constexpr const char* sound4::cloudx1::dyn::process_shortname = "x1"  [static], [constexpr]
```

Definition at line 489 of file sound4.x1.cloud_dyn.hpp.

13.3.4.9 SampleRate

```
const size_t sound4::cloudx1::dyn::SampleRate = 48000
```

Sample rate of library

Deprecated Use GetSampleRate

Definition at line 520 of file sound4.x1.cloud_dyn.hpp.

13.4 sound4::cloudx1::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, cloudx1_SampleFormat fmt)
- static void AudioConvertTo (CDynLib &dynlib, const float *input, uint8_t *payload, size_t nSpl, cloudx1_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::cloudx1::dyn::helper::AudioConvertFrom (
            CDynLib & dynlib,
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            cloudx1_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 586 of file sound4.x1.cloud_dyn.hpp.

13.4.2.2 AudioConvertTo()

```
static void sound4::cloudx1::dyn::helper::AudioConvertTo (
            CDynLib & dynlib,
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            cloudx1_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 601 of file sound4.x1.cloud_dyn.hpp.

13.4.2.3 AudioMonoFromLiveStereo()

```
static void sound4::cloudx1::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 670 of file sound4.x1.cloud_dyn.hpp.

13.4.2.4 AudioMonoToLiveStereo()

```
static void sound4::cloudx1::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 681 of file sound4.x1.cloud_dyn.hpp.

13.4.2.5 MonoToStereo()

```
static void sound4::cloudx1::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 629 of file sound4.x1.cloud_dyn.hpp.

13.4.2.6 MonoToStereo_Planar()

```
static void sound4::cloudx1::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 659 of file sound4.x1.cloud_dyn.hpp.

13.4.2.7 StereoToMono()

```
static void sound4::cloudx1::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 615 of file sound4.x1.cloud_dyn.hpp.

13.4.2.8 StereoToMono_Planar()

```
static void sound4::cloudx1::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 644 of file sound4.x1.cloud_dyn.hpp.

13.4.2.9 UTF8ToWString()

```
static std::wstring sound4::cloudx1::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 544 of file sound4.x1.cloud_dyn.hpp.

13.4.2.10 WStringToUTF8()

```
static std::string sound4::cloudx1::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 532 of file sound4.x1.cloud_dyn.hpp.

13.5 sound4::cloudx1::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, cloudx1_SampleFormat fmt)
- static void AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, cloudx1_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::cloudx1::helper::AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            cloudx1_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 151 of file sound4.x1.cloud.hpp.

13.5.2.2 AudioConvertTo()

```
static void sound4::cloudx1::helper::AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            cloudx1_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 165 of file sound4.x1.cloud.hpp.

13.5.2.3 AudioMonoFromLiveStereo()

```
static void sound4::cloudx1::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 229 of file sound4.x1.cloud.hpp.

13.5.2.4 AudioMonoToLiveStereo()

```
static void sound4::cloudx1::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 239 of file sound4.x1.cloud.hpp.

13.5.2.5 MonoToStereo()

```
static void sound4::cloudx1::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 191 of file sound4.x1.cloud.hpp.

13.5.2.6 MonoToStereo_Planar()

```
static void sound4::cloudx1::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 219 of file sound4.x1.cloud.hpp.

13.5.2.7 StereoToMono()

```
static void sound4::cloudx1::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 178 of file sound4.x1.cloud.hpp.

13.5.2.8 StereoToMono_Planar()

```
static void sound4::cloudx1::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 205 of file sound4.x1.cloud.hpp.

13.5.2.9 UTF8ToWString()

```
static std::wstring sound4::cloudx1::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 110 of file sound4.x1.cloud.hpp.

13.5.2.10 WStringToUTF8()

```
static std::string sound4::cloudx1::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 98 of file sound4.x1.cloud.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::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

14.1.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 841 of file sound4cl.hpp.

14.1.2 Constructor & Destructor Documentation

14.1.2.1 CBus()

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

Definition at line 843 of file sound4cl.hpp.

14.1.2.2 ~CBus()

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

Definition at line 850 of file sound4cl.hpp.

14.1.3 Member Function Documentation

14.1.3.1 Get()

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

Definition at line 859 of file sound4cl.hpp.

14.1.4 Friends And Related Symbol Documentation

14.1.4.1 CInstance

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

Definition at line 858 of file sound4cl.hpp.

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

- sound4cl.hpp

14.2 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.

14.2.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1413 of file sound4cl.hpp.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 CClient() [1/2]

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

Definition at line 1415 of file sound4cl.hpp.

14.2.2.2 CClient() [2/2]

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

14.2.2.3 ~CClient()

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

Definition at line 1425 of file sound4cl.hpp.

14.2.3 Member Function Documentation

14.2.3.1 operator=()

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

14.2.3.2 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 1439 of file sound4cl.hpp.

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

- sound4cl.hpp

14.3 sound4::cloudx1::CInstance::CClient Class Reference

`#include <sound4.x1.cloud.hpp>`

Public Member Functions

- CClient (cloudx1_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.

14.3.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 947 of file sound4.x1.cloud.hpp.

14.3.2 Constructor & Destructor Documentation

14.3.2.1 CClient() [1/2]

```
sound4::cloudx1::CInstance::CClient::CClient (
            cloudx1_CInstance * instance )  [inline]
```

Definition at line 949 of file sound4.x1.cloud.hpp.

14.3.2.2 CClient() [2/2]

```
sound4::cloudx1::CInstance::CClient::CClient (
            const CInstance &  )  [delete]
```

14.3.2.3 ~CClient()

```
sound4::cloudx1::CInstance::CClient::~CClient ( )  [inline]
```

Definition at line 958 of file sound4.x1.cloud.hpp.

14.3.3 Member Function Documentation

14.3.3.1 operator=()

```
CClient & sound4::cloudx1::CInstance::CClient::operator= (
            const CInstance &  )  [delete]
```

14.3.3.2 ProcessJson()

```
std::string sound4::cloudx1::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 972 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.4 sound4::cloudx1::dyn::CInstance::CClient Class Reference

`#include <sound4.x1.cloud_dyn.hpp>`

Public Member Functions

- CClient (cloudx1_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.

14.4.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1437 of file sound4.x1.cloud_dyn.hpp.

14.4.2 Constructor & Destructor Documentation

14.4.2.1 CClient() [1/2]

```
sound4::cloudx1::dyn::CInstance::CClient::CClient (
            cloudx1_CInstance * instance,
            CDynLib & dynlib )  [inline]
```

Definition at line 1439 of file sound4.x1.cloud_dyn.hpp.

14.4.2.2 CClient() [2/2]

```
sound4::cloudx1::dyn::CInstance::CClient::CClient (
            const CInstance &  )  [delete]
```

14.4.2.3 ~CClient()

```
sound4::cloudx1::dyn::CInstance::CClient::~CClient ( )  [inline]
```

Definition at line 1449 of file sound4.x1.cloud_dyn.hpp.

14.4.3 Member Function Documentation

14.4.3.1 operator=()

```
CClient & sound4::cloudx1::dyn::CInstance::CClient::operator= (
            const CInstance &  )  [delete]
```

14.4.3.2 ProcessJson()

```
std::string sound4::cloudx1::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 1463 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.5 sound4::cloudx1::dyn::CDynLib Class Reference

Dynamic library interface.

```
#include <sound4.x1.cloud_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(cloudx1_GetVersion) > GetVersion
- helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames) > GetChunkSizeInFrames
- helper::DynFuncHolder< decltype(cloudx1_GetChannelCount) > GetChannelCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount) > GetAudioInputCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount) > GetAudioOutputCount
- helper::DynFuncHolder< decltype(cloudx1_GetSampleRate) > GetSampleRate
- helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback) > SetLoggerCallback
- helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity) > SetLogSeverity
- helper::DynFuncHolder< decltype(cloudx1_NewParameters) > NewParameters
- helper::DynFuncHolder< decltype(cloudx1_FreeParameters) > FreeParameters
- helper::DynFuncHolder< decltype(cloudx1_SetParameter) > SetParameter
- helper::DynFuncHolder< decltype(cloudx1_GetParameter) > GetParameter
- helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue) > FreeParameterValue
- helper::DynFuncHolder< decltype(cloudx1_InitProcess) > InitProcess
- helper::DynFuncHolder< decltype(cloudx1_InitProcess2) > InitProcess2
- helper::DynFuncHolder< decltype(cloudx1_TerminateProcess) > TerminateProcess
- helper::DynFuncHolder< decltype(cloudx1_ExitProcess) > ExitProcess
- helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread) > StartUpdateThread
- helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread) > StopUpdateThread
- helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady) > WaitUpdateThreadReady
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio) > ProcessAudio
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar) > ProcessAudio_Planar
- helper::DynFuncHolder< decltype(cloudx1_GetBufferIn) > GetBufferIn
- helper::DynFuncHolder< decltype(cloudx1_GetBufferOut) > GetBufferOut
- helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay) > GetEstimatedDelay
- helper::DynFuncHolder< decltype(cloudx1_GetFormatName) > GetFormatName
- helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName) > GetFormatFromName
- helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat) > GetBytesFromFormat
- helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame) > GetMaxPacketFrame
- helper::DynFuncHolder< decltype(cloudx1_AddAudio) > AddAudio
- helper::DynFuncHolder< decltype(cloudx1_AddPadAudio) > AddPadAudio
- helper::DynFuncHolder< decltype(cloudx1_GetOutputCount) > GetOutputCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudio) > GetAudio
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom) > AudioConvertFrom
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo) > AudioConvertTo
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono) > StereoToMono
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo) > MonoToStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo) > AudioMonoToLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_NewClient) > NewClient
- helper::DynFuncHolder< decltype(cloudx1_DeleteClient) > DeleteClient
- helper::DynFuncHolder< decltype(cloudx1_ProcessJson) > ProcessJson
- helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer) > FreeJsonAnswer
- helper::DynFuncHolder< decltype(cloudx1_SaveState) > SaveState

- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp) > Webserver_tcp
- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2) > Webserver_tcp2
- helper::DynFuncHolder< decltype(cloudx1_Webserver) > Webserver
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop) > Webserver_Stop
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Status) > Webserver_Status
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar) > StereoToMono_Planar
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar) > MonoToStereo_Planar
- helper::DynFuncHolder< decltype(cloudx1_SetMetadata) > SetMetadata
- helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos) > GetMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos) > FreeMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_SetPresetManager) > SetPresetManager
- helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange) > PresetManager_InformChange
- helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames) > GetPossibleChunkSizeInFrames
- helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames) > GetProcessChunkFrames
- helper::DynFuncHolder< decltype(cloudx1_InitProcess3) > InitProcess3
- helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti) > SetMetadataMulti
- helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth) > Webserver_SetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth) > Webserver_GetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString) > Webserver_FreeString
- helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracing) > SetInstanceTracing
- helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracingProcessActivity) > SetInstanceTracingProcessActivity

14.5.1 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 223 of file sound4.x1.cloud_dyn.hpp.

14.5.2 Constructor & Destructor Documentation

14.5.2.1 CDynLib()

```
sound4::cloudx1::dyn::CDynLib::CDynLib ( )  [default]
```

14.5.3 Member Function Documentation

14.5.3.1 IsOk()

```
bool sound4::cloudx1::dyn::CDynLib::IsOk ( ) const  [inline]
```

Returns

true if OK.

false if not OK.

Definition at line 480 of file sound4.x1.cloud_dyn.hpp.

14.5.3.2 Load()

```
bool sound4::cloudx1::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 331 of file sound4.x1.cloud_dyn.hpp.

14.5.4 Member Data Documentation

14.5.4.1 AddAudio

```
helper::DynFuncHolder< decltype(cloudx1_AddAudio ) > sound4::cloudx1::dyn::CDynLib::AddAudio
```

Definition at line 260 of file sound4.x1.cloud_dyn.hpp.

14.5.4.2 AddPadAudio

```
helper::DynFuncHolder< decltype(cloudx1_AddPadAudio ) > sound4::cloudx1::dyn::CDynLib::Add↩
PadAudio
```

Definition at line 261 of file sound4.x1.cloud_dyn.hpp.

14.5.4.3 AudioConvertFrom

```
helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom ) > sound4::cloudx1::dyn::CDynLib::↩
AudioConvertFrom
```

Definition at line 264 of file sound4.x1.cloud_dyn.hpp.

14.5.4.4 AudioConvertTo

```
helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo ) > sound4::cloudx1::dyn::CDynLib::↩
AudioConvertTo
```

Definition at line 265 of file sound4.x1.cloud_dyn.hpp.

14.5.4.5 AudioMonoFromLiveStereo

```
helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > sound4::cloudx1::dyn::↩
CDynLib::AudioMonoFromLiveStereo
```

Definition at line 268 of file sound4.x1.cloud_dyn.hpp.

14.5.4.6 AudioMonoToLiveStereo

```
helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo ) > sound4::cloudx1::dyn::CDyn↩
Lib::AudioMonoToLiveStereo
```

Definition at line 269 of file sound4.x1.cloud_dyn.hpp.

14.5.4.7 DeleteClient

```
helper::DynFuncHolder< decltype(cloudx1_DeleteClient ) > sound4::cloudx1::dyn::CDynLib::↩
DeleteClient
```

Definition at line 271 of file sound4.x1.cloud_dyn.hpp.

14.5.4.8 ExitProcess

```
helper::DynFuncHolder< decltype(cloudx1_ExitProcess ) > sound4::cloudx1::dyn::CDynLib::Exit↩
Process
```

Definition at line 247 of file sound4.x1.cloud_dyn.hpp.

14.5.4.9 FreeJsonAnswer

```
helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer ) > sound4::cloudx1::dyn::CDynLib::↩
FreeJsonAnswer
```

Definition at line 273 of file sound4.x1.cloud_dyn.hpp.

14.5.4.10 FreeMetadataInfos

```
helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos ) > sound4::cloudx1::dyn::CDynLib↩
::FreeMetadataInfos
```

Definition at line 290 of file sound4.x1.cloud_dyn.hpp.

14.5.4.11 FreeParameters

```
helper::DynFuncHolder< decltype(cloudx1_FreeParameters ) > sound4::cloudx1::dyn::CDynLib::↩
FreeParameters
```

Definition at line 240 of file sound4.x1.cloud_dyn.hpp.

14.5.4.12 FreeParameterValue

```
helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue ) > sound4::cloudx1::dyn::CDynLib↩
::FreeParameterValue
```

Definition at line 243 of file sound4.x1.cloud_dyn.hpp.

14.5.4.13 GetAudio

```
helper::DynFuncHolder< decltype(cloudx1_GetAudio ) > sound4::cloudx1::dyn::CDynLib::GetAudio
```

Definition at line 263 of file sound4.x1.cloud_dyn.hpp.

14.5.4.14 GetAudioInputCount

```
helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount ) > sound4::cloudx1::dyn::CDynLib↩
::GetAudioInputCount
```

Definition at line 234 of file sound4.x1.cloud_dyn.hpp.

14.5.4.15 GetAudioOutputCount

```
helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount ) > sound4::cloudx1::dyn::CDyn↩
Lib::GetAudioOutputCount
```

Definition at line 235 of file sound4.x1.cloud_dyn.hpp.

14.5.4.16 GetBufferIn

```
helper::DynFuncHolder< decltype(cloudx1_GetBufferIn ) > sound4::cloudx1::dyn::CDynLib::Get↩
BufferIn
```

Definition at line 253 of file sound4.x1.cloud_dyn.hpp.

14.5.4.17 GetBufferOut

```
helper::DynFuncHolder< decltype(cloudx1_GetBufferOut ) > sound4::cloudx1::dyn::CDynLib::Get↩
BufferOut
```

Definition at line 254 of file sound4.x1.cloud_dyn.hpp.

14.5.4.18 GetBytesFromFormat

```
helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat ) > sound4::cloudx1::dyn::CDynLib↩
::GetBytesFromFormat
```

Definition at line 258 of file sound4.x1.cloud_dyn.hpp.

14.5.4.19 GetChannelCount

```
helper::DynFuncHolder< decltype(cloudx1_GetChannelCount ) > sound4::cloudx1::dyn::CDynLib::↩
GetChannelCount
```

Definition at line 233 of file sound4.x1.cloud_dyn.hpp.

14.5.4.20 GetChunkSizeInFrames

```
helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames ) > sound4::cloudx1::dyn::CDyn↩
Lib::GetChunkSizeInFrames
```

Definition at line 232 of file sound4.x1.cloud_dyn.hpp.

14.5.4.21 GetEstimatedDelay

```
helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay ) > sound4::cloudx1::dyn::CDynLib↩
::GetEstimatedDelay
```

Definition at line 255 of file sound4.x1.cloud_dyn.hpp.

14.5.4.22 GetFormatFromName

```
helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName ) > sound4::cloudx1::dyn::CDynLib↩
::GetFormatFromName
```

Definition at line 257 of file sound4.x1.cloud_dyn.hpp.

14.5.4.23 GetFormatName

```
helper::DynFuncHolder< decltype(cloudx1_GetFormatName ) > sound4::cloudx1::dyn::CDynLib::Get↩
FormatName
```

Definition at line 256 of file sound4.x1.cloud_dyn.hpp.

14.5.4.24 GetMaxPacketFrame

```
helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame ) > sound4::cloudx1::dyn::CDynLib↩
::GetMaxPacketFrame
```

Definition at line 259 of file sound4.x1.cloud_dyn.hpp.

14.5.4.25 GetMetadataInfos

```
helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos ) > sound4::cloudx1::dyn::CDynLib::↩
GetMetadataInfos
```

Definition at line 289 of file sound4.x1.cloud_dyn.hpp.

14.5.4.26 GetOutputCount

```
helper::DynFuncHolder< decltype(cloudx1_GetOutputCount ) > sound4::cloudx1::dyn::CDynLib::↩
GetOutputCount
```

Definition at line 262 of file sound4.x1.cloud_dyn.hpp.

14.5.4.27 GetParameter

```
helper::DynFuncHolder< decltype(cloudx1_GetParameter ) > sound4::cloudx1::dyn::CDynLib::Get↩
Parameter
```

Definition at line 242 of file sound4.x1.cloud_dyn.hpp.

14.5.4.28 GetPossibleChunkSizeInFrames

```
helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames ) > sound4::cloudx1↩
::dyn::CDynLib::GetPossibleChunkSizeInFrames
```

Definition at line 297 of file sound4.x1.cloud_dyn.hpp.

14.5.4.29 GetProcessChunkFrames

```
helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames ) > sound4::cloudx1::dyn::CDyn↩
Lib::GetProcessChunkFrames
```

Definition at line 298 of file sound4.x1.cloud_dyn.hpp.

14.5.4.30 GetSampleRate

```
helper::DynFuncHolder< decltype(cloudx1_GetSampleRate ) > sound4::cloudx1::dyn::CDynLib::Get↩
SampleRate
```

Definition at line 236 of file sound4.x1.cloud_dyn.hpp.

14.5.4.31 GetVersion

```
helper::DynFuncHolder< decltype(cloudx1_GetVersion ) > sound4::cloudx1::dyn::CDynLib::Get↩
Version
```

Definition at line 231 of file sound4.x1.cloud_dyn.hpp.

14.5.4.32 InitProcess

```
helper::DynFuncHolder< decltype(cloudx1_InitProcess ) > sound4::cloudx1::dyn::CDynLib::Init↩
Process
```

Definition at line 244 of file sound4.x1.cloud_dyn.hpp.

14.5.4.33 InitProcess2

```
helper::DynFuncHolder< decltype(cloudx1_InitProcess2 ) > sound4::cloudx1::dyn::CDynLib::Init↩
Process2
```

Definition at line 245 of file sound4.x1.cloud_dyn.hpp.

14.5.4.34 InitProcess3

```
helper::DynFuncHolder< decltype(cloudx1_InitProcess3 ) > sound4::cloudx1::dyn::CDynLib::Init↩
Process3
```

Definition at line 299 of file sound4.x1.cloud_dyn.hpp.

14.5.4.35 MonoToStereo

```
helper::DynFuncHolder< decltype(cloudx1_MonoToStereo ) > sound4::cloudx1::dyn::CDynLib::Mono↩
ToStereo
```

Definition at line 267 of file sound4.x1.cloud_dyn.hpp.

14.5.4.36 MonoToStereo_Planar

```
helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar ) > sound4::cloudx1::dyn::CDyn↩
Lib::MonoToStereo_Planar
```

Definition at line 285 of file sound4.x1.cloud_dyn.hpp.

14.5.4.37 NewClient

```
helper::DynFuncHolder< decltype(cloudx1_NewClient ) > sound4::cloudx1::dyn::CDynLib::NewClient
```

Definition at line 270 of file sound4.x1.cloud_dyn.hpp.

14.5.4.38 NewParameters

```
helper::DynFuncHolder< decltype(cloudx1_NewParameters ) > sound4::cloudx1::dyn::CDynLib::New↩
Parameters
```

Definition at line 239 of file sound4.x1.cloud_dyn.hpp.

14.5.4.39 PresetManager_InformChange

```
helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange ) > sound4::cloudx1::dyn↩
::CDynLib::PresetManager_InformChange
```

Definition at line 294 of file sound4.x1.cloud_dyn.hpp.

14.5.4.40 ProcessAudio

```
helper::DynFuncHolder< decltype(cloudx1_ProcessAudio ) > sound4::cloudx1::dyn::CDynLib::↩
ProcessAudio
```

Definition at line 251 of file sound4.x1.cloud_dyn.hpp.

14.5.4.41 ProcessAudio_Planar

helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar) > sound4::cloudx1::dyn::CDyn↩
Lib::ProcessAudio_Planar

Definition at line 252 of file sound4.x1.cloud_dyn.hpp.

14.5.4.42 ProcessJson

helper::DynFuncHolder< decltype(cloudx1_ProcessJson) > sound4::cloudx1::dyn::CDynLib::↩
ProcessJson

Definition at line 272 of file sound4.x1.cloud_dyn.hpp.

14.5.4.43 SaveState

helper::DynFuncHolder< decltype(cloudx1_SaveState) > sound4::cloudx1::dyn::CDynLib::SaveState

Definition at line 274 of file sound4.x1.cloud_dyn.hpp.

14.5.4.44 SetInstanceTracing

helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracing) > sound4::cloudx1::dyn::CDynLib↩
::SetInstanceTracing

Definition at line 320 of file sound4.x1.cloud_dyn.hpp.

14.5.4.45 SetInstanceTracingProcessActivity

helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracingProcessActivity) > sound4::cloudx1↩
::dyn::CDynLib::SetInstanceTracingProcessActivity

Definition at line 321 of file sound4.x1.cloud_dyn.hpp.

14.5.4.46 SetLoggerCallback

helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback) > sound4::cloudx1::dyn::CDynLib↩
::SetLoggerCallback

Definition at line 237 of file sound4.x1.cloud_dyn.hpp.

14.5.4.47 SetLogSeverity

helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity) > sound4::cloudx1::dyn::CDynLib::↩
SetLogSeverity

Definition at line 238 of file sound4.x1.cloud_dyn.hpp.

14.5.4.48 SetMetadata

```
helper::DynFuncHolder< decltype(cloudx1_SetMetadata ) > sound4::cloudx1::dyn::CDynLib::Set↩
Metadata
```

Definition at line 288 of file sound4.x1.cloud_dyn.hpp.

14.5.4.49 SetMetadataMulti

```
helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti ) > sound4::cloudx1::dyn::CDynLib::↩
SetMetadataMulti
```

Definition at line 309 of file sound4.x1.cloud_dyn.hpp.

14.5.4.50 SetParameter

```
helper::DynFuncHolder< decltype(cloudx1_SetParameter ) > sound4::cloudx1::dyn::CDynLib::Set↩
Parameter
```

Definition at line 241 of file sound4.x1.cloud_dyn.hpp.

14.5.4.51 SetPresetManager

```
helper::DynFuncHolder< decltype(cloudx1_SetPresetManager ) > sound4::cloudx1::dyn::CDynLib::↩
SetPresetManager
```

Definition at line 293 of file sound4.x1.cloud_dyn.hpp.

14.5.4.52 StartUpdateThread

```
helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread ) > sound4::cloudx1::dyn::CDynLib↩
::StartUpdateThread
```

Definition at line 248 of file sound4.x1.cloud_dyn.hpp.

14.5.4.53 StereoToMono

```
helper::DynFuncHolder< decltype(cloudx1_StereoToMono ) > sound4::cloudx1::dyn::CDynLib::↩
StereoToMono
```

Definition at line 266 of file sound4.x1.cloud_dyn.hpp.

14.5.4.54 StereoToMono_Planar

```
helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar ) > sound4::cloudx1::dyn::CDyn↩
Lib::StereoToMono_Planar
```

Definition at line 284 of file sound4.x1.cloud_dyn.hpp.

14.5.4.55 StopUpdateThread

helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread) > sound4::cloudx1::dyn::CDynLib::↩
StopUpdateThread

Definition at line 249 of file sound4.x1.cloud_dyn.hpp.

14.5.4.56 TerminateProcess

helper::DynFuncHolder< decltype(cloudx1_TerminateProcess) > sound4::cloudx1::dyn::CDynLib::↩
TerminateProcess

Definition at line 246 of file sound4.x1.cloud_dyn.hpp.

14.5.4.57 WaitUpdateThreadReady

helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady) > sound4::cloudx1::dyn::CDyn↩
Lib::WaitUpdateThreadReady

Definition at line 250 of file sound4.x1.cloud_dyn.hpp.

14.5.4.58 Webserver

helper::DynFuncHolder< decltype(cloudx1_Webserver) > sound4::cloudx1::dyn::CDynLib::Webserver

Definition at line 278 of file sound4.x1.cloud_dyn.hpp.

14.5.4.59 Webserver_FreeString

helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString) > sound4::cloudx1::dyn::CDyn↩
Lib::Webserver_FreeString

Definition at line 315 of file sound4.x1.cloud_dyn.hpp.

14.5.4.60 Webserver_GetAppHealth

helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth) > sound4::cloudx1::dyn::↩
CDynLib::Webserver_GetAppHealth

Definition at line 314 of file sound4.x1.cloud_dyn.hpp.

14.5.4.61 Webserver_SetAppHealth

helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth) > sound4::cloudx1::dyn::↩
CDynLib::Webserver_SetAppHealth

Definition at line 313 of file sound4.x1.cloud_dyn.hpp.

14.5.4.62 Webserver_Status

helper::DynFuncHolder< decltype(cloudx1_Webserver_Status) > sound4::cloudx1::dyn::CDynLib::↩
Webserver_Status

Definition at line 280 of file sound4.x1.cloud_dyn.hpp.

14.5.4.63 Webserver_Stop

helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop) > sound4::cloudx1::dyn::CDynLib::↩
Webserver_Stop

Definition at line 279 of file sound4.x1.cloud_dyn.hpp.

14.5.4.64 Webserver_tcp

helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp) > sound4::cloudx1::dyn::CDynLib::↩
Webserver_tcp

Definition at line 276 of file sound4.x1.cloud_dyn.hpp.

14.5.4.65 Webserver_tcp2

helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2) > sound4::cloudx1::dyn::CDynLib::↩
Webserver_tcp2

Definition at line 277 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.6 sound4::cloudx1::dyn::helper::CDynLoader Class Reference

Helper to load dynamic library.

#include <sound4.x1.cloud_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)

14.6.1 Detailed Description

Definition at line 108 of file sound4.x1.cloud_dyn.hpp.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 CDynLoader() [1/3]

```
sound4::cloudx1::dyn::helper::CDynLoader::CDynLoader ( )  [default]
```

14.6.2.2 CDynLoader() [2/3]

```
sound4::cloudx1::dyn::helper::CDynLoader::CDynLoader (
            CDynLoader &&  )  [default]
```

14.6.2.3 CDynLoader() [3/3]

```
sound4::cloudx1::dyn::helper::CDynLoader::CDynLoader (
            const CDynLoader &  )  [delete]
```

14.6.2.4 ~CDynLoader()

```
virtual sound4::cloudx1::dyn::helper::CDynLoader::~CDynLoader ( )  [inline], [virtual]
```

Definition at line 211 of file sound4.x1.cloud_dyn.hpp.

14.6.3 Member Function Documentation

14.6.3.1 Close()

```
void sound4::cloudx1::dyn::helper::CDynLoader::Close ( )  [inline]
```

Definition at line 172 of file sound4.x1.cloud_dyn.hpp.

14.6.3.2 GetSymbol()

```
template<typename T >
DynFuncHolder< T > sound4::cloudx1::dyn::helper::CDynLoader::GetSymbol (
            const std::string & name )  [inline]
```

Definition at line 185 of file sound4.x1.cloud_dyn.hpp.

14.6.3.3 GetSymbol_safe()

```
template<typename T >
DynFuncHolder< T > sound4::cloudx1::dyn::helper::CDynLoader::GetSymbol_safe (
            const std::string & name )  [inline]
```

Definition at line 193 of file sound4.x1.cloud_dyn.hpp.

14.6.3.4 GetThisLibraryPath()

```
static std::filesystem::path sound4::cloudx1::dyn::helper::CDynLoader::GetThisLibraryPath (
            void  )  [inline], [static]
```

Definition at line 197 of file sound4.x1.cloud_dyn.hpp.

14.6.3.5 IsOk()

```
bool sound4::cloudx1::dyn::helper::CDynLoader::IsOk ( ) const  [inline]
```

Definition at line 126 of file sound4.x1.cloud_dyn.hpp.

14.6.3.6 Load()

```
bool sound4::cloudx1::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 166 of file sound4.x1.cloud_dyn.hpp.

14.6.3.7 operator bool()

```
sound4::cloudx1::dyn::helper::CDynLoader::operator bool ( ) const  [inline]
```

Definition at line 123 of file sound4.x1.cloud_dyn.hpp.

14.6.3.8 operator=()

```
CDynLoader & sound4::cloudx1::dyn::helper::CDynLoader::operator= (
            CDynLoader const &  )  [delete]
```

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

- sound4.x1.cloud_dyn.hpp

14.7 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)

14.7.1 Detailed Description

Definition at line 155 of file sound4cl.hpp.

14.7.2 Constructor & Destructor Documentation

14.7.2.1 CDynLoader() [1/3]

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

14.7.2.2 CDynLoader() [2/3]

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

14.7.2.3 CDynLoader() [3/3]

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

14.7.2.4 ~CDynLoader()

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

Definition at line 258 of file sound4cl.hpp.

14.7.3 Member Function Documentation

14.7.3.1 Close()

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

Definition at line 219 of file sound4cl.hpp.

14.7.3.2 GetSymbol()

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

Definition at line 232 of file sound4cl.hpp.

14.7.3.3 GetSymbol_safe()

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

Definition at line 240 of file sound4cl.hpp.

14.7.3.4 GetThisLibraryPath()

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

Definition at line 244 of file sound4cl.hpp.

14.7.3.5 IsOk()

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

Definition at line 173 of file sound4cl.hpp.

14.7.3.6 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 213 of file sound4cl.hpp.

14.7.3.7 operator bool()

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

Definition at line 170 of file sound4cl.hpp.

14.7.3.8 operator=()

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

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

- sound4cl.hpp

14.8 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)

14.8.1 Detailed Description

Holds an instance of processing

Definition at line 893 of file sound4cl.hpp.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 CInstance() [1/3]

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

Creates the instance, with default parameters

Parameters

dynlib	the processor dynamic library

Definition at line 900 of file sound4cl.hpp.

14.8.2.2 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 915 of file sound4cl.hpp.

14.8.2.3 CInstance() [3/3]

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

14.8.2.4 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 927 of file sound4cl.hpp.

14.8.3 Member Function Documentation

14.8.3.1 AddAudio()

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

Definition at line 1513 of file sound4cl.hpp.

14.8.3.2 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 1031 of file sound4cl.hpp.

14.8.3.3 GetAudio()

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

Definition at line 1519 of file sound4cl.hpp.

14.8.3.4 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 1330 of file sound4cl.hpp.

14.8.3.5 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 1347 of file sound4cl.hpp.

14.8.3.6 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 1077 of file sound4cl.hpp.

14.8.3.7 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 1312 of file sound4cl.hpp.

14.8.3.8 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 1165 of file sound4cl.hpp.

14.8.3.9 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 956 of file sound4cl.hpp.

14.8.3.10 GetWebServerAppHealth()

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

Get the web server /apphealth status info

Definition at line 1258 of file sound4cl.hpp.

14.8.3.11 IsOk()

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

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

Definition at line 935 of file sound4cl.hpp.

14.8.3.12 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 1463 of file sound4cl.hpp.

14.8.3.13 OnUpdateThreadStart()

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

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

Definition at line 1189 of file sound4cl.hpp.

14.8.3.14 OnUpdateThreadStop()

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

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

Definition at line 1193 of file sound4cl.hpp.

14.8.3.15 operator=()

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

14.8.3.16 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 1091 of file sound4cl.hpp.

14.8.3.17 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 1389 of file sound4cl.hpp.

14.8.3.18 ProcessAudio()

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

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

Definition at line 1355 of file sound4cl.hpp.

14.8.3.19 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 1371 of file sound4cl.hpp.

14.8.3.20 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 1479 of file sound4cl.hpp.

14.8.3.21 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 973 of file sound4cl.hpp.

14.8.3.22 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 1493 of file sound4cl.hpp.

14.8.3.23 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 1504 of file sound4cl.hpp.

14.8.3.24 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 1109 of file sound4cl.hpp.

14.8.3.25 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 1129 of file sound4cl.hpp.

14.8.3.26 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 945 of file sound4cl.hpp.

14.8.3.27 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 990 of file sound4cl.hpp.

14.8.3.28 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 1249 of file sound4cl.hpp.

14.8.3.29 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 1209 of file sound4cl.hpp.

14.8.3.30 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 1289 of file sound4cl.hpp.

14.8.3.31 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 1237 of file sound4cl.hpp.

14.8.3.32 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 1278 of file sound4cl.hpp.

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

- sound4cl.hpp

14.9 sound4::cloudx1::CInstance Class Reference

Instance handling class.

`#include <sound4.x1.cloud.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 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=64)
- 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)

14.9.1 Detailed Description

Holds an instance of processing

Definition at line 471 of file sound4.x1.cloud.hpp.

14.9.2 Constructor & Destructor Documentation

14.9.2.1 CInstance() [1/2]

```
sound4::cloudx1::CInstance::CInstance ( )  [inline]
```

Creates the instance, with default parameters

Definition at line 476 of file sound4.x1.cloud.hpp.

14.9.2.2 CInstance() [2/2]

```
sound4::cloudx1::CInstance::CInstance (
            const CInstance &  )  [delete]
```

14.9.2.3 ~CInstance()

```
virtual sound4::cloudx1::CInstance::~CInstance ( )  [inline], [virtual]
```

Destructor, stopping any running tasks

Definition at line 487 of file sound4.x1.cloud.hpp.

14.9.3 Member Function Documentation

14.9.3.1 AddAudio()

```
template<typename T >
unsigned int sound4::cloudx1::CInstance::AddAudio (
            const T * payload,
            unsigned int nFrame )  [inline], [protected]
```

Definition at line 1041 of file sound4.x1.cloud.hpp.

14.9.3.2 Create()

```
bool sound4::cloudx1::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 = 64 )  [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 cloudx1_GetPossibleChunkSizeInFrames, or 0 to set default

Returns

false if failed

Definition at line 590 of file sound4.x1.cloud.hpp.

14.9.3.3 GetAudio()

```
template<typename T >
unsigned int sound4::cloudx1::CInstance::GetAudio (
            T * payload,
            unsigned int max_nFrame )  [inline], [protected]
```

Definition at line 1047 of file sound4.x1.cloud.hpp.

14.9.3.4 GetBufferIn()

```
std::array< float, InputSampleSize > & sound4::cloudx1::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 862 of file sound4.x1.cloud.hpp.

14.9.3.5 GetBufferOut()

```
std::array< float, OutputSampleSize > & sound4::cloudx1::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 880 of file sound4.x1.cloud.hpp.

14.9.3.6 GetChunkFrames()

```
unsigned int sound4::cloudx1::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 627 of file sound4.x1.cloud.hpp.

14.9.3.7 GetEstimatedDelay()

```
unsigned int sound4::cloudx1::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 844 of file sound4.x1.cloud.hpp.

14.9.3.8 GetMetadataInfos()

```
std::vector< std::tuple< std::string, std::string > > sound4::cloudx1::CInstance::GetMetadata↩
Infos ( )  [inline]
```

Get information on known metadata keys.

Returns

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

Definition at line 705 of file sound4.x1.cloud.hpp.

14.9.3.9 GetParam()

```
std::string sound4::cloudx1::CInstance::GetParam (
            const std::string & name )  [inline]
```

Gets a parameter value

Parameters

name	the parameter name

Returns

the parameter value

Definition at line 516 of file sound4.x1.cloud.hpp.

14.9.3.10 GetWebServerAppHealth()

```
void sound4::cloudx1::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 791 of file sound4.x1.cloud.hpp.

14.9.3.11 IsOk()

```
bool sound4::cloudx1::CInstance::IsOk ( ) const  [inline]
```

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

Definition at line 495 of file sound4.x1.cloud.hpp.

14.9.3.12 NewClient()

```
std::shared_ptr< CClient > sound4::cloudx1::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 995 of file sound4.x1.cloud.hpp.

14.9.3.13 OnUpdateThreadStart()

```
virtual void sound4::cloudx1::CInstance::OnUpdateThreadStart ( )  [inline], [virtual]
```

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

Definition at line 727 of file sound4.x1.cloud.hpp.

14.9.3.14 OnUpdateThreadStop()

```
virtual void sound4::cloudx1::CInstance::OnUpdateThreadStop ( )  [inline], [virtual]
```

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

Definition at line 731 of file sound4.x1.cloud.hpp.

14.9.3.15 operator=()

```
CInstance & sound4::cloudx1::CInstance::operator= (
            const CInstance &  )  [delete]
```

14.9.3.16 PresetManager_InformChange()

```
void sound4::cloudx1::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 641 of file sound4.x1.cloud.hpp.

14.9.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector< T > sound4::cloudx1::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 CLOUDX1_AUDIOFRAME_COUNT

Template Parameters

T	the C type for one audio sample

Parameters

input	audio to process

Returns

processed audio

Definition at line 923 of file sound4.x1.cloud.hpp.

14.9.3.18 ProcessAudio()

```
void sound4::cloudx1::CInstance::ProcessAudio ( )  [inline]
```

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

Definition at line 889 of file sound4.x1.cloud.hpp.

14.9.3.19 ProcessAudio_Planar()

```
void sound4::cloudx1::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 905 of file sound4.x1.cloud.hpp.

14.9.3.20 SaveState()

```
bool sound4::cloudx1::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 1011 of file sound4.x1.cloud.hpp.

14.9.3.21 SetInstanceTracing()

```
void sound4::cloudx1::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 1025 of file sound4.x1.cloud.hpp.

14.9.3.22 SetInstanceTracingProcessActivity()

```
void sound4::cloudx1::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 1034 of file sound4.x1.cloud.hpp.

14.9.3.23 SetMetadata()

```
void sound4::cloudx1::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 657 of file sound4.x1.cloud.hpp.

14.9.3.24 SetMetadataMulti()

```
void sound4::cloudx1::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 675 of file sound4.x1.cloud.hpp.

14.9.3.25 SetParam()

```
void sound4::cloudx1::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 505 of file sound4.x1.cloud.hpp.

14.9.3.26 SetPresetManager()

```
void sound4::cloudx1::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 550 of file sound4.x1.cloud.hpp.

14.9.3.27 SetWebServerAppHealth()

```
void sound4::cloudx1::CInstance::SetWebServerAppHealth (
            int httpcode,
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 783 of file sound4.x1.cloud.hpp.

14.9.3.28 StartWebServer()

```
bool sound4::cloudx1::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 747 of file sound4.x1.cloud.hpp.

14.9.3.29 Stop()

```
void sound4::cloudx1::CInstance::Stop ( )  [inline]
```

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

Definition at line 821 of file sound4.x1.cloud.hpp.

14.9.3.30 StopWebServer()

```
void sound4::cloudx1::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 772 of file sound4.x1.cloud.hpp.

14.9.3.31 Terminate()

```
int sound4::cloudx1::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 812 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.10 sound4::cloudx1::dyn::CInstance Class Reference

Instance handling class.

`#include <sound4.x1.cloud_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 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=64)
- 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)

14.10.1 Detailed Description

Holds an instance of processing

Definition at line 937 of file sound4.x1.cloud_dyn.hpp.

14.10.2 Constructor & Destructor Documentation

14.10.2.1 CInstance() [1/2]

```
sound4::cloudx1::dyn::CInstance::CInstance (
            CDynLib & dynlib )  [inline]
```

Creates the instance, with default parameters

Definition at line 942 of file sound4.x1.cloud_dyn.hpp.

14.10.2.2 CInstance() [2/2]

```
sound4::cloudx1::dyn::CInstance::CInstance (
            const CInstance &  )  [delete]
```

14.10.2.3 ~CInstance()

```
virtual sound4::cloudx1::dyn::CInstance::~CInstance ( )  [inline], [virtual]
```

Destructor, stopping any running tasks

Definition at line 954 of file sound4.x1.cloud_dyn.hpp.

14.10.3 Member Function Documentation

14.10.3.1 AddAudio()

```
template<typename T >
unsigned int sound4::cloudx1::dyn::CInstance::AddAudio (
            const T * payload,
            unsigned int nFrame )  [inline], [protected]
```

Definition at line 1537 of file sound4.x1.cloud_dyn.hpp.

14.10.3.2 Create()

```
bool sound4::cloudx1::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 = 64 )  [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 cloudx1_GetPossibleChunkSizeInFrames, or 0 to set default

Returns

false if failed

Definition at line 1059 of file sound4.x1.cloud_dyn.hpp.

14.10.3.3 GetAudio()

```
template<typename T >
unsigned int sound4::cloudx1::dyn::CInstance::GetAudio (
            T * payload,
            unsigned int max_nFrame )  [inline], [protected]
```

Definition at line 1543 of file sound4.x1.cloud_dyn.hpp.

14.10.3.4 GetBufferIn()

```
std::array< float, InputSampleSize > & sound4::cloudx1::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 1352 of file sound4.x1.cloud_dyn.hpp.

14.10.3.5 GetBufferOut()

```
std::array< float, OutputSampleSize > & sound4::cloudx1::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 1370 of file sound4.x1.cloud_dyn.hpp.

14.10.3.6 GetChunkFrames()

```
unsigned int sound4::cloudx1::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 1102 of file sound4.x1.cloud_dyn.hpp.

14.10.3.7 GetEstimatedDelay()

```
unsigned int sound4::cloudx1::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 1334 of file sound4.x1.cloud_dyn.hpp.

14.10.3.8 GetMetadataInfos()

```
std::vector< std::tuple< std::string, std::string > > sound4::cloudx1::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 1191 of file sound4.x1.cloud_dyn.hpp.

14.10.3.9 GetParam()

```
std::string sound4::cloudx1::dyn::CInstance::GetParam (
            const std::string & name )  [inline]
```

Gets a parameter value

Parameters

name	the parameter name

Returns

the parameter value

Definition at line 983 of file sound4.x1.cloud_dyn.hpp.

14.10.3.10 GetWebServerAppHealth()

```
void sound4::cloudx1::dyn::CInstance::GetWebServerAppHealth (
            int & httpcode,
            std::string & contenttype,
            std::string & content )  [inline]
```

Get the web server /apphealth status info

Definition at line 1280 of file sound4.x1.cloud_dyn.hpp.

14.10.3.11 IsOk()

```
bool sound4::cloudx1::dyn::CInstance::IsOk ( ) const  [inline]
```

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

Definition at line 962 of file sound4.x1.cloud_dyn.hpp.

14.10.3.12 NewClient()

```
std::shared_ptr< CClient > sound4::cloudx1::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 1487 of file sound4.x1.cloud_dyn.hpp.

14.10.3.13 OnUpdateThreadStart()

```
virtual void sound4::cloudx1::dyn::CInstance::OnUpdateThreadStart ( )  [inline], [virtual]
```

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

Definition at line 1215 of file sound4.x1.cloud_dyn.hpp.

14.10.3.14 OnUpdateThreadStop()

```
virtual void sound4::cloudx1::dyn::CInstance::OnUpdateThreadStop ( )  [inline], [virtual]
```

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

Definition at line 1219 of file sound4.x1.cloud_dyn.hpp.

14.10.3.15 operator=()

```
CInstance & sound4::cloudx1::dyn::CInstance::operator= (
            const CInstance &  )  [delete]
```

14.10.3.16 PresetManager_InformChange()

```
void sound4::cloudx1::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 1116 of file sound4.x1.cloud_dyn.hpp.

14.10.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector< T > sound4::cloudx1::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 CLOUDX1_AUDIOFRAME_COUNT

Template Parameters

T	the C type for one audio sample

Parameters

input	audio to process

Returns

processed audio

Definition at line 1413 of file sound4.x1.cloud_dyn.hpp.

14.10.3.18 ProcessAudio()

```
void sound4::cloudx1::dyn::CInstance::ProcessAudio ( )  [inline]
```

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

Definition at line 1379 of file sound4.x1.cloud_dyn.hpp.

14.10.3.19 ProcessAudio_Planar()

```
void sound4::cloudx1::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 1395 of file sound4.x1.cloud_dyn.hpp.

14.10.3.20 SaveState()

```
bool sound4::cloudx1::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 1503 of file sound4.x1.cloud_dyn.hpp.

14.10.3.21 SetInstanceTracing()

```
void sound4::cloudx1::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 1517 of file sound4.x1.cloud_dyn.hpp.

14.10.3.22 SetInstanceTracingProcessActivity()

```
void sound4::cloudx1::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 1528 of file sound4.x1.cloud_dyn.hpp.

14.10.3.23 SetMetadata()

```
void sound4::cloudx1::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 1134 of file sound4.x1.cloud_dyn.hpp.

14.10.3.24 SetMetadataMulti()

```
void sound4::cloudx1::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 1155 of file sound4.x1.cloud_dyn.hpp.

14.10.3.25 SetParam()

```
void sound4::cloudx1::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 972 of file sound4.x1.cloud_dyn.hpp.

14.10.3.26 SetPresetManager()

```
void sound4::cloudx1::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 1018 of file sound4.x1.cloud_dyn.hpp.

14.10.3.27 SetWebServerAppHealth()

```
void sound4::cloudx1::dyn::CInstance::SetWebServerAppHealth (
            int httpcode,
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 1271 of file sound4.x1.cloud_dyn.hpp.

14.10.3.28 StartWebServer()

```
bool sound4::cloudx1::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 1235 of file sound4.x1.cloud_dyn.hpp.

14.10.3.29 Stop()

```
void sound4::cloudx1::dyn::CInstance::Stop ( )  [inline]
```

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

Definition at line 1311 of file sound4.x1.cloud_dyn.hpp.

14.10.3.30 StopWebServer()

```
void sound4::cloudx1::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 1260 of file sound4.x1.cloud_dyn.hpp.

14.10.3.31 Terminate()

```
int sound4::cloudx1::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 1302 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.11 cloudx1_CClientInstance Struct Reference

`#include <sound4.x1.cloud.h>`

14.11.1 Detailed Description

Abstract pointer for a client instance

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

- sound4.x1.cloud.h

14.12 cloudx1_CInstance Struct Reference

`#include <sound4.x1.cloud.h>`

14.12.1 Detailed Description

Abstract holder for a process instance

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

- sound4.x1.cloud.h

14.13 cloudx1_CParameters Struct Reference

`#include <sound4.x1.cloud.h>`

14.13.1 Detailed Description

Abstract holder for a process instance parameters

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

- sound4.x1.cloud.h

14.14 sound4::cloudx1::CPresetLoader Class Reference

Custom preset handler helper.

`#include <sound4.x1.cloud.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

14.14.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 451 of file sound4.x1.cloud.hpp.

14.14.2 Constructor & Destructor Documentation

14.14.2.1 CPresetLoader()

```
sound4::cloudx1::CPresetLoader::CPresetLoader ( )  [default]
```

14.14.2.2 ~CPresetLoader()

```
virtual sound4::cloudx1::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

14.14.3 Member Function Documentation

14.14.3.1 Exists()

```
virtual bool sound4::cloudx1::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

14.14.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::cloudx1::CPresetLoader::GetAll ( )  [pure
virtual]
```

14.14.3.3 IsReadOnly()

```
virtual bool sound4::cloudx1::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

14.14.3.4 Read()

```
virtual std::string sound4::cloudx1::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

14.14.3.5 Remove()

```
virtual bool sound4::cloudx1::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

14.14.3.6 Rename()

```
virtual bool sound4::cloudx1::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

14.14.3.7 Write()

```
virtual bool sound4::cloudx1::CPresetLoader::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [pure virtual]
```

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

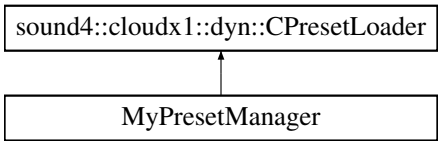
- sound4.x1.cloud.hpp

14.15 sound4::cloudx1::dyn::CPresetLoader Class Reference

Custom preset handler helper.

`#include <sound4.x1.cloud_dyn.hpp>`

Inheritance diagram for sound4::cloudx1::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

14.15.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 917 of file sound4.x1.cloud_dyn.hpp.

14.15.2 Constructor & Destructor Documentation

14.15.2.1 CPresetLoader()

```
sound4::cloudx1::dyn::CPresetLoader::CPresetLoader ( )  [default]
```

14.15.2.2 ~CPresetLoader()

```
virtual sound4::cloudx1::dyn::CPresetLoader::~CPresetLoader ( )  [virtual], [default]
```

14.15.3 Member Function Documentation

14.15.3.1 Exists()

```
virtual bool sound4::cloudx1::dyn::CPresetLoader::Exists (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.2 GetAll()

```
virtual std::vector< std::filesystem::path > sound4::cloudx1::dyn::CPresetLoader::GetAll ( )
[pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.3 IsReadOnly()

```
virtual bool sound4::cloudx1::dyn::CPresetLoader::IsReadOnly ( )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.4 Read()

```
virtual std::string sound4::cloudx1::dyn::CPresetLoader::Read (
            const std::filesystem::path & filename )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.5 Remove()

```
virtual bool sound4::cloudx1::dyn::CPresetLoader::Remove (
            const std::filesystem::path & name )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.6 Rename()

```
virtual bool sound4::cloudx1::dyn::CPresetLoader::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [pure virtual]
```

Implemented in MyPresetManager, and MyPresetManager.

14.15.3.7 Write()

```
virtual bool sound4::cloudx1::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:

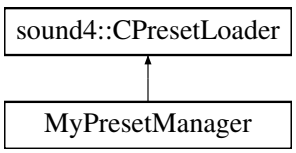
- sound4.x1.cloud_dyn.hpp

14.16 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

14.16.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 870 of file sound4cl.hpp.

14.16.2 Constructor & Destructor Documentation

14.16.2.1 CPresetLoader()

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

14.16.2.2 ~CPresetLoader()

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

14.16.3 Member Function Documentation

14.16.3.1 Exists()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.2 GetAll()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.3 IsReadOnly()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.4 Read()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.5 Remove()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.6 Rename()

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

Implemented in MyPresetManager, and MyPresetManager.

14.16.3.7 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

14.17 sound4::CProcessor Class Reference

Dynamic library interface.

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

Public Member Functions

- CProcessor ()=default
- bool Load (const std::filesystem::path &filepath)

  Loads the library.
- bool IsOk () const

  Check if the library was loaded correctly.
- void AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
- void AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
- void StereoToMono (const float *input, float *output, size_t nFrame)
- void MonoToStereo (const float *input, float *output, size_t nFrame)
- void StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- void AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- std::string GetVersion ()

- unsigned int GetChunkSizeInFrames ()
- std::vector< unsigned int > GetPossibleChunkSizeInFrames ()
- unsigned int GetChannelCount ()
- unsigned int GetAudioInputCount ()
- unsigned int GetAudioOutputCount ()
- unsigned int GetSampleRate ()
- std::string GetFormatName (const SampleFormat fmt)
- SampleFormat GetFormatFromName (const std::string &name)
- unsigned int GetBytesFromFormat (const SampleFormat fmt)
- void SetLogSeverity (LogSeverity severity)
- void SetLoggerCallback (log_cb_t cb)

Protected Member Functions

- template<typename T >
  helper::DynFuncHolder< T > GetPrefixSymbol (const std::string &name)

Friends

- class CBus
- class CInstance

14.17.1 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 302 of file sound4cl.hpp.

14.17.2 Constructor & Destructor Documentation

14.17.2.1 CProcessor()

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

14.17.3 Member Function Documentation

14.17.3.1 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 773 of file sound4cl.hpp.

14.17.3.2 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 783 of file sound4cl.hpp.

14.17.3.3 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 820 of file sound4cl.hpp.

14.17.3.4 GetChannelCount()

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

Get the number of channels needed by this process

Returns

The process channels count

Definition at line 763 of file sound4cl.hpp.

14.17.3.5 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 734 of file sound4cl.hpp.

14.17.3.6 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 811 of file sound4cl.hpp.

14.17.3.7 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 802 of file sound4cl.hpp.

14.17.3.8 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 745 of file sound4cl.hpp.

14.17.3.9 GetPrefixSymbol()

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

Definition at line 402 of file sound4cl.hpp.

14.17.3.10 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 793 of file sound4cl.hpp.

14.17.3.11 GetVersion()

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

Get the process version string

Returns

version string

Definition at line 721 of file sound4cl.hpp.

14.17.3.12 IsOk()

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

Returns

true if OK.

false if not OK.

Definition at line 608 of file sound4cl.hpp.

14.17.3.13 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 417 of file sound4cl.hpp.

14.17.3.14 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 832 of file sound4cl.hpp.

14.17.3.15 SetLogSeverity()

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

Sets the log severity (global)

Definition at line 826 of file sound4cl.hpp.

14.17.4 Friends And Related Symbol Documentation

14.17.4.1 CBus

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

Definition at line 308 of file sound4cl.hpp.

14.17.4.2 CInstance

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

Definition at line 309 of file sound4cl.hpp.

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

- sound4cl.hpp

14.18 sound4::cloudx1::dyn::helper::DynFuncHolder< T > Class Template Reference

Helper to wrap a function pointer.

```
#include <sound4.x1.cloud_dyn.hpp>
```

Public Member Functions

- DynFuncHolder ()=default
- DynFuncHolder (DynFunc_t a_ptr)
- operator bool () const
- bool IsOk () const
- operator T* () const

14.18.1 Detailed Description

template<typename T>
class sound4::cloudx1::dyn::helper::DynFuncHolder< T >

Parameters

T	is the function type, you may use decltype(fn)

Definition at line 87 of file sound4.x1.cloud_dyn.hpp.

14.18.2 Constructor & Destructor Documentation

14.18.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::cloudx1::dyn::helper::DynFuncHolder< T >::DynFuncHolder ( )  [default]
```

14.18.2.2 DynFuncHolder() [2/2]

```
template<typename T >
sound4::cloudx1::dyn::helper::DynFuncHolder< T >::DynFuncHolder (
            DynFunc_t a_ptr )  [inline]
```

Definition at line 98 of file sound4.x1.cloud_dyn.hpp.

14.18.3 Member Function Documentation

14.18.3.1 IsOk()

```
template<typename T >
bool sound4::cloudx1::dyn::helper::DynFuncHolder< T >::IsOk ( ) const  [inline]
```

Definition at line 100 of file sound4.x1.cloud_dyn.hpp.

14.18.3.2 operator bool()

```
template<typename T >
sound4::cloudx1::dyn::helper::DynFuncHolder< T >::operator bool ( ) const  [inline]
```

Definition at line 99 of file sound4.x1.cloud_dyn.hpp.

14.18.3.3 operator T*()

```
template<typename T >
sound4::cloudx1::dyn::helper::DynFuncHolder< T >::operator T* ( ) const  [inline]
```

Definition at line 101 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.19 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

14.19.1 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.

14.19.2 Constructor & Destructor Documentation

14.19.2.1 DynFuncHolder() [1/2]

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

14.19.2.2 DynFuncHolder() [2/2]

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

Definition at line 145 of file sound4cl.hpp.

14.19.3 Member Function Documentation

14.19.3.1 IsOk()

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

Definition at line 147 of file sound4cl.hpp.

14.19.3.2 operator bool()

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

Definition at line 146 of file sound4cl.hpp.

14.19.3.3 operator T*()

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

Definition at line 148 of file sound4cl.hpp.

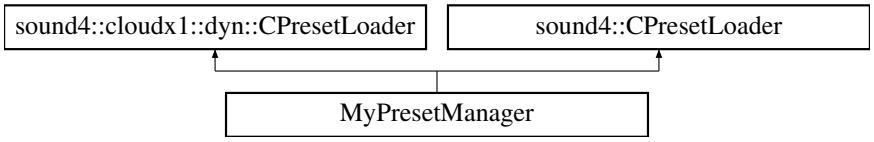
The documentation for this class was generated from the following file:

- sound4cl.hpp

14.20 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::cloudx1::dyn::CPresetLoader

- CPresetLoader ()=default
- virtual ~CPresetLoader ()=default

Public Member Functions inherited from sound4::CPresetLoader

- CPresetLoader ()=default
- virtual ~CPresetLoader ()=default

14.20.1 Detailed Description

Definition at line 120 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.2 Constructor & Destructor Documentation

14.20.2.1 MyPresetManager() [1/2]

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

14.20.2.2 ~MyPresetManager() [1/2]

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

14.20.2.3 MyPresetManager() [2/2]

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

14.20.2.4 ~MyPresetManager() [2/2]

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

14.20.3 Member Function Documentation

14.20.3.1 Exists() [1/2]

```
virtual bool MyPresetManager::Exists (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 138 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.2 Exists() [2/2]

```
virtual bool MyPresetManager::Exists (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 122 of file sound4cl-pipe.cpp.

14.20.3.3 GetAll() [1/2]

```
virtual std::vector< std::filesystem::path > MyPresetManager::GetAll ( )  [inline], [override],
[virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 154 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.4 GetAll() [2/2]

```
virtual std::vector< std::filesystem::path > MyPresetManager::GetAll ( )  [inline], [override],
[virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 138 of file sound4cl-pipe.cpp.

14.20.3.5 IsReadOnly() [1/2]

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

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 137 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.6 IsReadOnly() [2/2]

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

Implements sound4::cloudx1::dyn::CPresetLoader.

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

14.20.3.7 Read() [1/2]

```
virtual std::string MyPresetManager::Read (
            const std::filesystem::path & filename )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 168 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.8 Read() [2/2]

```
virtual std::string MyPresetManager::Read (
            const std::filesystem::path & filename )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 152 of file sound4cl-pipe.cpp.

14.20.3.9 Remove() [1/2]

```
virtual bool MyPresetManager::Remove (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 143 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.10 Remove() [2/2]

```
virtual bool MyPresetManager::Remove (
            const std::filesystem::path & name )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 127 of file sound4cl-pipe.cpp.

14.20.3.11 Rename() [1/2]

```
virtual bool MyPresetManager::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 148 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.12 Rename() [2/2]

```
virtual bool MyPresetManager::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 132 of file sound4cl-pipe.cpp.

14.20.3.13 SetupPath() [1/2]

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

Definition at line 125 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.14 SetupPath() [2/2]

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

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

14.20.3.15 Write() [1/2]

```
virtual bool MyPresetManager::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 178 of file sound4.x1.cloud-pipe-dyn.cpp.

14.20.3.16 Write() [2/2]

```
virtual bool MyPresetManager::Write (
            const std::filesystem::path & filename,
            const std::string & content )  [inline], [override], [virtual]
```

Implements sound4::cloudx1::dyn::CPresetLoader.

Definition at line 162 of file sound4cl-pipe.cpp.

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

- sound4.x1.cloud-pipe-dyn.cpp
- sound4cl-pipe.cpp

14.21 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

```
#include <sound4.x1.cloud_dyn.hpp>
```

14.21.1 Detailed Description

template<typename T>
struct sound4::cloudx1::dyn::helper::SampleFormat< T >

Template Parameters

T	the C type for one audio sample

Definition at line 559 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.22 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

`#include <sound4.x1.cloud.hpp>`

14.22.1 Detailed Description

template<typename T>
struct sound4::cloudx1::helper::SampleFormat< T >

Template Parameters

T	the C type for one audio sample

Definition at line 125 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.23 sound4::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

`#include <sound4cl.hpp>`

14.23.1 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

14.24 SOUND4_ProcessInfo Struct Reference

`#include <sound4.x1.cloud.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.

14.24.1 Detailed Description

Information on this processor.

See also

SOUND4_GetProcessInfo

Definition at line 77 of file sound4.x1.cloud.h.

14.24.2 Member Data Documentation

14.24.2.1 name

`const char* SOUND4_ProcessInfo::name`

Definition at line 79 of file sound4.x1.cloud.h.

14.24.2.2 prefix

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

Definition at line 80 of file sound4.x1.cloud.h.

14.24.2.3 shortname

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

Definition at line 81 of file sound4.x1.cloud.h.

14.24.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

Definition at line 78 of file sound4.x1.cloud.h.

14.24.2.5 version

```
const char* SOUND4_ProcessInfo::version
```

Definition at line 82 of file sound4.x1.cloud.h.

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

- sound4.x1.cloud.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 sound4.x1.cloud-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.x1.cloud.hpp"
```

Functions

- std::string GetEnv (const std::string &name)
- void ProcessLoop (sound4::cloudx1::CInstance &instance)
- void Usage (const char *progname)
- void MyLogger (sound4::cloudx1::LogSeverity severity, const std::string &msg)
- int main (int argc, const char **argv)

15.11.1 Function Documentation

15.11.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 41 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.2 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.x1.cloud-pipe-cxx.cpp.

15.11.1.3 MyLogger()

```
void MyLogger (
            sound4::cloudx1::LogSeverity severity,
            const std::string & msg )
```

Definition at line 101 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::cloudx1::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 50 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.5 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 81 of file sound4.x1.cloud-pipe-cxx.cpp.

15.12 sound4.x1.cloud-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.x1.cloud.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::cloudx1::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::cloudx1::InputSampleSize,
      stdin) != sound4::cloudx1::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::cloudx1::OutputSampleSize,
      stdout) != sound4::cloudx1::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::cloudx1::GetSampleRate()«" -ac
      "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00094     #if defined(_WIN32) || defined(WIN32)
00095         std::cerr « "ffmpeg -f dshow -i audio=\"Microphone\" -ar "«sound4::cloudx1::GetSampleRate()«"
      -ac "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00096     #else
00097         std::cerr « "ffmpeg -f pulse -i default -ar "«sound4::cloudx1::GetSampleRate()«" -ac
      "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00098     #endif
00099 }
00100
00101 void MyLogger(sound4::cloudx1::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 (CLOUDX1_HAS_WEBSERVER)
00109     int listen_port = 8080;
00110 #endif // (CLOUDX1_HAS_WEBSERVER)
00111     auto store_path_str=GetEnv("S4STORE_PATH");
00112
00113     // Get credentials from environment
00114     auto LoginKey=GetEnv("S4LOGINKEY");
00115     if (LoginKey.empty()) {
00116         LoginKey=GetEnv("S4USERID");
00117     }
00118     auto RadioName=GetEnv("RADIO_NAME");
00119     auto KeyId=GetEnv("ACCESS_KEY_ID");
00120     auto AccessKey=GetEnv("ACCESS_KEY_SECRET");
00121
00122     const char *progname=argv[0];
00123
00124     if (argc>1) {
00125         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00126             Usage(progname);
00127             return 0;
00128         }
00129     }
00130     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00131         std::cerr « "Missing credentials in environment" « std::endl;
00132         Usage(progname);
00133         return 1;
00134     }
00135     std::filesystem::path store_path;
00136     if (!store_path_str.empty()) {
00137         std::error_code ec;
00138         store_path = std::filesystem::path(store_path_str);
00139         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
      {
00140             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
      std::endl;
00141             Usage(progname);
00142             return 1;
00143         }
00144     }
00145
00147     sound4::cloudx1::SetLogSeverity(verbose5);
00148     sound4::cloudx1::SetLoggerCallback(MyLogger);
00149
00150     sound4::cloudx1::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 (CLOUDX1_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 // (CLOUDX1_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 }
```

15.13 sound4.x1.cloud-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.x1.cloud_dyn.hpp"
```

Classes

- class MyPresetManager

  [Custom preset manager]

Functions

- std::string GetEnv (const std::string &name)
- void ProcessLoop (sound4::cloudx1::dyn::CInstance &instance)
- void Usage (const char *progname, sound4::cloudx1::dyn::CDynLib &dynlib)
- void MyLogger (sound4::cloudx1::dyn::LogSeverity severity, const std::string &msg)
- int main (int argc, const char **argv)

  [Custom preset manager]

15.13.1 Function Documentation

15.13.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 52 of file sound4.x1.cloud-pipe-dyn.cpp.

15.13.1.2 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.x1.cloud-pipe-dyn.cpp.

15.13.1.3 MyLogger()

```
void MyLogger (
            sound4::cloudx1::dyn::LogSeverity severity,
            const std::string & msg )
```

Definition at line 114 of file sound4.x1.cloud-pipe-dyn.cpp.

15.13.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::cloudx1::dyn::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 61 of file sound4.x1.cloud-pipe-dyn.cpp.

15.13.1.5 Usage()

```
void Usage (
            const char * progname,
            sound4::cloudx1::dyn::CDynLib & dynlib )
```

Definition at line 92 of file sound4.x1.cloud-pipe-dyn.cpp.

15.14 sound4.x1.cloud-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.x1.cloud_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::cloudx1::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::cloudx1::dyn::InputSampleSize, stdin) != sound4::cloudx1::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::cloudx1::dyn::OutputSampleSize, stdout) != sound4::cloudx1::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::cloudx1::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::cloudx1::dyn::LogSeverity severity,const std::string& msg)
00115 {
00116     std::cerr « msg « std::endl;
00117 }
00118
00120 class MyPresetManager : public sound4::cloudx1::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 (CLOUDX1_HAS_WEBSERVER)
00202     int listen_port = 8080;
00203 #endif // (CLOUDX1_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::cloudx1::dyn::CDynLib dynlib;
00220     if (!dynlib.Load()) {
00221         std::cerr « "Failed to load processing library" « std::endl;
00222         return 1;
00223     }
00225 
00226     if (argc>1) {
00227         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00228             Usage(progname,dynlib);
00229             return 0;
00230         }
00231     }
00232     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00233         std::cerr « "Missing credentials in environment" « std::endl;
00234         Usage(progname,dynlib);
00235         return 1;
00236     }
00237     std::filesystem::path store_path;
00238     if (!store_path_str.empty()) {
00239         std::error_code ec;
00240         store_path = std::filesystem::path(store_path_str);
00241         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
    {
00242             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
    std::endl;
00243             Usage(progname,dynlib);
00244             return 1;
00245         }
00246     }
00247 
00249     sound4::cloudx1::dyn::SetLogSeverity(dynlib, verbose5);
00250     sound4::cloudx1::dyn::SetLoggerCallback(dynlib, MyLogger);
00251 
00252     sound4::cloudx1::dyn::CInstance instance(dynlib);
00253     instance.SetParam("ADMIN_USER","admin");
00254     instance.SetParam("ADMIN_SECRET","admin");
00256 
00258     MyPresetManager preset_manager;
00259     if (!store_path.empty()) {
00260         if (!preset_manager.SetupPath(store_path)) {
00261             std::cerr « "Failed to setup the custom preset manager on path " « store_path.c_str() «
    std::endl;
00262             return 1;
00263         }
00264         instance.SetPresetManager(&preset_manager);
00265     }
00267 
00268     std::cerr « "Creating" « std::endl;
00270     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00271         std::cerr « "Failed to create the processing instance" « std::endl;
00272         return 1;
00273     }
00275 
00276 #if (CLOUDX1_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 // (CLOUDX1_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 }
```

15.15 sound4.x1.cloud-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.x1.cloud.h>
```

Macros

- #define SAMPLE_RATE 48000
- #define CHUNK_SIZE 64
- #define CHANNEL_COUNT 2

Functions

- void ProcessLoop (struct cloudx1_CInstance *pInstance)
- void Usage (const char *progname)
- void MyLogger (cloudx1_LogSeverity severe, const char *str)
- int main (int argc, char *const argv[])

15.15.1 Macro Definition Documentation

15.15.1.1 CHANNEL_COUNT

```
#define CHANNEL_COUNT 2
```

Definition at line 22 of file sound4.x1.cloud-pipe.cpp.

15.15.1.2 CHUNK_SIZE

```
#define CHUNK_SIZE 64
```

Definition at line 21 of file sound4.x1.cloud-pipe.cpp.

15.15.1.3 SAMPLE_RATE

```
#define SAMPLE_RATE 48000
```

Definition at line 20 of file sound4.x1.cloud-pipe.cpp.

15.15.2 Function Documentation

15.15.2.1 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.x1.cloud-pipe.cpp.

15.15.2.2 MyLogger()

```
void MyLogger (
            cloudx1_LogSeverity severe,
            const char * str )
```

Definition at line 78 of file sound4.x1.cloud-pipe.cpp.

15.15.2.3 ProcessLoop()

```
void ProcessLoop (
            struct cloudx1_CInstance * pInstance )
```

[Processing Loop]

[Processing Loop]

Definition at line 25 of file sound4.x1.cloud-pipe.cpp.

15.15.2.4 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 58 of file sound4.x1.cloud-pipe.cpp.

15.16 sound4.x1.cloud-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.x1.cloud.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 cloudx1_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         cloudx1_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(cloudx1_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 cloudx1_CInstance *pInstance = NULL;
00087     const char *store_path_str=getenv("S4STORE_PATH");
00088     // Get credentials from environment
00089     char *pLoginKey=getenv("S4LOGINKEY");
00090     if (!pLoginKey) {
00091         pLoginKey=getenv("S4USERID");
00092     }
00093     char *pRadioName=getenv("RADIO_NAME");
00094     char *pKeyId=getenv("ACCESS_KEY_ID");
00095     char *pAccessKey=getenv("ACCESS_KEY_SECRET");
00096
00097     const char *progname=argv[0];
00098
00099     if (argc>1) {
00100         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00101             Usage(progname);
00102             return 0;
00103         }
00104     }
00105     if (!pLoginKey || !pRadioName || !pKeyId || !pAccessKey) {
00106         std::cerr « "Missing credentials in environment" « std::endl;
00107         Usage(progname);
00108         return 1;
00109     }
00110     std::error_code ec;
00111     std::filesystem::path fs_store_path;
00112     if (!store_path_str || store_path_str[0]==0) {
00113         fs_store_path=std::filesystem::current_path(ec) / "state";
00114     } else {
00115         fs_store_path = std::filesystem::path(store_path_str);
00116     }
00117     if (!std::filesystem::exists(fs_store_path,ec) ||
    !std::filesystem::is_directory(fs_store_path,ec)) {
00118         std::cerr « "Storage folder '"«fs_store_path.c_str()«"' is not a valid directory" « std::endl;
00119         Usage(progname);
00120         return 1;
00121     }
00122     std::string store_path;
00123     #ifdef _WIN32
00124     {
00125         // Convert path to UTF-8
00126         std::wstring wstr(fs_store_path.wstring());
00127         if (wstr.empty()) return {};
00128         int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
00129         std::string str(size, 0);
00130         WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00131         store_path=str;
00132     }
00133     #else // !_WIN32
00134         store_path=fs_store_path;
00135     #endif // !_WIN32
00136
00138     // Optional: Use our own logging, instead of default color output to stderr.
00139     //cloudx1_SetLoggerCallback(&MyLogger);
00140
00141     //cloudx1_SetLogSeverity(cloudx1_LogSeverity::info);
00142     cloudx1_SetLogSeverity(cloudx1_LogSeverity::verbose5);
00143
00144     std::cerr « "Process version: " « cloudx1_GetVersion() « std::endl;
00145     // Do some sanity checks
00146     if (cloudx1_GetSampleRate()!=SAMPLE_RATE) {
00147       std::cerr « "Bad library sampling rate, expecting '"«SAMPLE_RATE«"' but has "«
    cloudx1_GetSampleRate() « std::endl;
00148       return 1;
00149     }
00150     if (cloudx1_GetChunkSizeInFrames()!=CHUNK_SIZE) {
00151       std::cerr « "Bad library frame size, expecting '"«CHUNK_SIZE«"' but has "«
    cloudx1_GetChunkSizeInFrames() « std::endl;
00152       return 1;
00153     }
00154     if (cloudx1_GetChannelCount()!=CHANNEL_COUNT) {
00155       std::cerr « "Bad library channel count, expecting '"«CHANNEL_COUNT«"' but has "«
    cloudx1_GetChannelCount() « std::endl;
00156       return 1;
00157     }
00159
00161     // Setup parameters
00162     struct cloudx1_CParameters *params = cloudx1_NewParameters();
00163     // Optional: activate web authentication, set admin password
00164     // This creates the authentication only for a new storage path
00165     cloudx1_SetParameter(params,"ADMIN_USER", "admin");
00166     cloudx1_SetParameter(params,"ADMIN_SECRET", "admin");
```

```
00167
00168     // Create the processing instance
00169     pInstance=cloudx1_InitProcess2(pLoginKey,pRadioName,pKeyId,pAccessKey,store_path.c_str(), params);
00170     cloudx1_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 « "cloudx1_StartUpdateThread (no json port)" « std::endl;
00180        cloudx1_StartUpdateThread(pInstance, 0);
00181        std::cerr « "cloudx1_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 (cloudx1_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 (CLOUDX1_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 = cloudx1_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=cloudx1_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 // (CLOUDX1_HAS_WEBSERVER)
00223
00224      // Force load a factory preset now
00225      // Could be used instead of setting a store_path
00226      /*{
00227          cloudx1_CClientInstance *pClient=cloudx1_NewClient(pInstance);
00228          const char *answer=cloudx1_ProcessJson(pClient, "{ \"factoryload\": \"All Programs\" }",
      NULL);
00229          std::cerr « "Loaded factory preset answer: " « answer« std::endl;
00230          cloudx1_FreeJsonAnswer(answer);
00231          cloudx1_DeleteClient(pClient);
00232      }*/
00234
00235     std::cerr « "Estimated delay " « ( cloudx1_GetEstimatedDelay(pInstance)*1000.0/SAMPLE_RATE ) « "
      ms" « std::endl;
00236
00237     ProcessLoop(pInstance);
00238
00240 #if (CLOUDX1_HAS_WEBSERVER)
00241     // Stopping the webserver
00242     std::cerr « "Stopping web server" « std::endl;
00243     cloudx1_Webserver_Stop(webserver, 1000);
00244     std::cerr « "Stopping web server DONE" « std::endl;
00245 #endif // (CLOUDX1_HAS_WEBSERVER)
00246
00247     // Stopping the processing instance
00248     std::cerr « "Stopping Update thread" « std::endl;
00249     cloudx1_StopUpdateThread(pInstance); // this will exit the update thread
00250     std::cerr « "Stopping Update thread DONE" « std::endl;
00251
00252     updater.join();
00253
00254     cloudx1_ExitProcess(pInstance);
00256
```

```
00257     return 0;
00258 }
```

15.17 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]

15.17.1 Function Documentation

15.17.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 61 of file sound4cl-pipe.cpp.

15.17.1.2 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.

15.17.1.3 ProcessLoop()

```
void ProcessLoop (
             sound4::CInstance & instance,
             sound4::CProcessor & processor )
```

[Processing Loop]

[Processing Loop]

Definition at line 70 of file sound4cl-pipe.cpp.

15.18 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.x1.cloud.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 2 -f f32le - | "«progname«" | ffplay
      -f f32le -ar 48000 -ac 2 -i -" « std::endl;;
00233             #if defined(_WIN32) || defined(WIN32)
00234             std::cerr « "  ffmpeg -f dshow -i audio=\"Microphone\" -ar 48000 -ac 2 -f f32le - |
      "«progname«" | ffplay -f f32le -ar 48000 -ac 2 -i -" « std::endl;;
00235             #else
00236             std::cerr « "  ffmpeg -f pulse -i default -ar 48000 -ac 2 -f f32le - | "«progname«" |
      ffplay -f f32le -ar 48000 -ac 2 -i -" « std::endl;;
00237             #endif
00238             return 0;
00239         } else {
00240             std::cerr « "unknown argument '"«arg«"'"« std::endl;
00241             return 1;
00242         }
00243     }
00244
00245
00247     sound4::CProcessor processor;
00248     if (!processor.Load(libname)) {
00249         std::cerr « "Failed to load processing library" « std::endl;
00250         return 1;
00251     }
00253
00254     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00255         std::cerr « "Missing credentials in environment" « std::endl;
00256         return 1;
00257     }
00258     std::filesystem::path store_path;
00259     if (!store_path_str.empty()) {
00260         store_path = std::filesystem::path(store_path_str);
00261         std::error_code ec;
00262         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
      {
00263             std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
      std::endl;
00264             return 1;
00265         }
00266     }
00267
00269     processor.SetLogSeverity(sound4::verbose5);
00270     processor.SetLoggerCallback([](sound4::LogSeverity severity,const std::string& msg) {
00271             std::cerr « msg « std::endl;
00272         });
00273
00274     sound4::CInstance instance(processor);
00275     instance.SetParam("ADMIN_USER","admin");
00276     instance.SetParam("ADMIN_SECRET","admin");
00278
00280     MyPresetManager preset_manager;
00281     if (!store_path.empty()) {
00282         if (!preset_manager.SetupPath(store_path)) {
00283             std::cerr « "Failed to setup the custom preset manager on path " « store_path.c_str() «
      std::endl;
00284             return 1;
00285         }
00286         instance.SetPresetManager(&preset_manager);
00287     }
00289
00290     std::cerr « "Creating" « std::endl;
00292     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00293         std::cerr « "Failed to create the processing instance" « std::endl;
00294         return 1;
00295     }
00297
00298     std::cerr « "Starting web server" « std::endl;
00300     if (!instance.StartWebServer(listen_port,0)) {
00301         std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
00302         return 1;
00303     }
00305
00306     {
```

```
00308         auto client = instance.NewClient();
00309         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00310         std::cerr « "Request for processor name returned "«answer«std::endl;
00312     }
00313
00314     ProcessLoop(instance, processor);
00315
00316     std::cerr « "Stopping now" « std::endl;
00317     instance.Stop();
00318
00319     std::cerr « "End" « std::endl;
00320     return 0;
00321 }
```

15.19 sound4.x1.cloud.h File Reference

C interface for library.

```
#include <windows.h>
#include <stdint.h>
```

Classes

- struct SOUND4_ProcessInfo

Macros

- #define CLOUDX1_DECL __declspec(dllimport)
- #define CLOUDX1_HAS_WEBSERVER 1
- #define CLOUDX1_HAS_CLOUDBUS 0
- #define CLOUDX1_AUDIOFRAME_COUNT 64
- #define CLOUDX1_CHANNEL_COUNT 2
- #define CLOUDX1_AUDIO_INPUT_COUNT 1
- #define CLOUDX1_AUDIO_OUTPUT_COUNT 1
- #define CLOUDX1_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(* cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *)
- typedef struct _tlgProvider_t const * TraceLoggingHProvider
- typedef wchar_t fs_char
- typedef char *(* cloudx1_storage_reader) (const fs_char *filename, void *userdata)
- typedef void(* cloudx1_storage_reader_free) (char *content, void *userdata)
- typedef int(* cloudx1_storage_writer) (const fs_char *filename, const char *content, void *userdata)
- typedef int(* cloudx1_storage_exists) (const fs_char *filename, void *userdata)
- typedef fs_char **(* cloudx1_storage_getall) (void *userdata)
- typedef void(* cloudx1_storage_getall_free) (fs_char **all, void *userdata)
- typedef int(* cloudx1_storage_remove) (const fs_char *filename, void *userdata)
- typedef int(* cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)

Enumerations

- enum cloudx1_LogSeverity {
  none = 0 , fatal = 1 , error = 2 , warning = 3 ,
  info = 4 , verbose = 5 , verbose2 = 6 , verbose3 = 7 ,
  verbose4 = 8 , verbose5 = 9 }
- enum cloudx1_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted
  }
- enum cloudx1_SampleFormat {
  INVALID_FORMAT =0 , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE =S16_LE , S24_NATIVE =S24_LE , S32_NATIVE =S32_LE ,
  F32_NATIVE =F32_LE }

Functions

- const struct SOUND4_ProcessInfo * SOUND4_GetProcessInfo ()
- const char * cloudx1_GetVersion ()
- unsigned int cloudx1_GetChunkSizeInFrames ()
- unsigned int * cloudx1_GetPossibleChunkSizeInFrames ()
- unsigned int cloudx1_GetChannelCount ()
- unsigned int cloudx1_GetAudioInputCount ()
- unsigned int cloudx1_GetAudioOutputCount ()
- unsigned int cloudx1_GetSampleRate ()
- void cloudx1_SetLoggerCallback (cloudx1_loggerfn logger)
- void cloudx1_SetLogSeverity (enum cloudx1_LogSeverity severity)
- struct cloudx1_CParameters * cloudx1_NewParameters ()
- void cloudx1_FreeParameters (struct cloudx1_CParameters *params)
- void cloudx1_SetParameter (struct cloudx1_CParameters *params, const char *name, const char *value)
- const char * cloudx1_GetParameter (struct cloudx1_CParameters *params, const char *name)
- void cloudx1_FreeParameterValue (const char *value)
- struct cloudx1_CInstance * cloudx1_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct cloudx1_CInstance * cloudx1_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters)
- struct cloudx1_CInstance * cloudx1_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters, unsigned int frames_per_chunk)
- unsigned int cloudx1_GetProcessChunkFrames (struct cloudx1_CInstance *instance)
- int cloudx1_TerminateProcess (struct cloudx1_CInstance *instance)
- void cloudx1_ExitProcess (struct cloudx1_CInstance *instance)
- void cloudx1_SetInstanceTracing (struct cloudx1_CParameters *params, TraceLoggingHProvider tracing_↩ provider, GUID activity_guid)
- void cloudx1_SetInstanceTracingProcessActivity (struct cloudx1_CInstance *instance, GUID activity_guid)
- void cloudx1_SetPresetManager (struct cloudx1_CParameters *params, cloudx1_storage_reader reader, cloudx1_storage_reader_free, cloudx1_storage_writer writer, cloudx1_storage_exists exists, cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free, cloudx1_storage_remove remove, cloudx1_storage_rename rename, int IsReadOnly, void *userdata)
- void cloudx1_PresetManager_InformChange (struct cloudx1_CInstance *instance, const fs_char *relative↩ _path, enum cloudx1_PresetChange_Kind change_kind)
- void cloudx1_SetMetadata (struct cloudx1_CInstance *instance, const char *key, const char *value)
- void cloudx1_SetMetadataMulti (struct cloudx1_CInstance *instance, const char **keyvalue)
- const char ** cloudx1_GetMetadataInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeMetadataInfos (struct cloudx1_CInstance *instance, const char **infos)

- void cloudx1_StartUpdateThread (struct cloudx1_CInstance *instance, unsigned int port)
- void cloudx1_StopUpdateThread (struct cloudx1_CInstance *instance)
- int cloudx1_WaitUpdateThreadReady (struct cloudx1_CInstance *instance, int milliseconds)
- void cloudx1_ProcessAudio (struct cloudx1_CInstance *instance, const float *input, float *output)
- void cloudx1_ProcessAudio_Planar (struct cloudx1_CInstance *instance, float const *const *input, float *const *output)
- float * cloudx1_GetBufferIn (struct cloudx1_CInstance *instance)
- float * cloudx1_GetBufferOut (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetEstimatedDelay (struct cloudx1_CInstance *instance)
- const char * cloudx1_GetFormatName (const enum cloudx1_SampleFormat fmt)
- enum cloudx1_SampleFormat cloudx1_GetFormatFromName (const char *name)
- unsigned int cloudx1_GetBytesFromFormat (const enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_GetMaxPacketFrame (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_AddAudio (struct cloudx1_CInstance *instance, const uint8_t *payload, unsigned int nFrame, enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_AddPadAudio (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetOutputCount (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetAudio (struct cloudx1_CInstance *instance, uint8_t *payload, unsigned int max_↩ nFrame, enum cloudx1_SampleFormat fmt)
- void cloudx1_AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_StereoToMono (const float *input, float *output, size_t nFrame)
- void cloudx1_MonoToStereo (const float *input, float *output, size_t nFrame)
- void cloudx1_StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void cloudx1_MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void cloudx1_AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- void cloudx1_AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- struct cloudx1_CClientInstance * cloudx1_NewClient (struct cloudx1_CInstance *instance)
- void cloudx1_DeleteClient (struct cloudx1_CClientInstance *client)
- const char * cloudx1_ProcessJson (struct cloudx1_CClientInstance *client, const char *json_str, int *need↩ _save)
- void cloudx1_FreeJsonAnswer (const char *json_str)
- int cloudx1_SaveState (struct cloudx1_CInstance *instance)
- uint64_t cloudx1_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)
- uint64_t cloudx1_Webserver_tcp2 (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port, const struct cloudx1_CParameters *parameters)
- uint64_t cloudx1_Webserver (unsigned int listenport, unsigned int listenport_secure, struct cloudx1_CInstance *instance)
- int cloudx1_Webserver_Stop (uint64_t id, int timeout_ms)
- int cloudx1_Webserver_Status (uint64_t id)
- void cloudx1_Webserver_SetAppHealth (struct cloudx1_CInstance *instance, int httpcode, const char *contenttype, const char *content)
- void cloudx1_Webserver_GetAppHealth (struct cloudx1_CInstance *instance, int *httpcode, char **contenttype, char **content)
- void cloudx1_Webserver_FreeString (char *str)

15.19.1 Macro Definition Documentation

15.19.1.1 CLOUDX1_DECL

```
#define CLOUDX1_DECL __declspec(dllimport)
```

Definition at line 14 of file sound4.x1.cloud.h.

15.19.1.2 CLOUDX1_HAS_CLOUDBUS

```
#define CLOUDX1_HAS_CLOUDBUS 0
```

Definition at line 26 of file sound4.x1.cloud.h.

15.19.1.3 CLOUDX1_HAS_WEBSERVER

```
#define CLOUDX1_HAS_WEBSERVER 1
```

Definition at line 25 of file sound4.x1.cloud.h.

15.19.2 Typedef Documentation

15.19.2.1 TraceLoggingHProvider

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

Definition at line 418 of file sound4.x1.cloud.h.

15.20 sound4.x1.cloud.h

Go to the documentation of this file.

```
00001
00004 #pragma once
00005
00006 #ifdef __unix__
00007     #include <sys/types.h>
00008     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00009 #elif defined(_WIN32) || defined(WIN32)
00010     #include <windows.h>
00011     #ifdef BUILDING_CLOUDX1_DLL
00012         #define CLOUDX1_DECL    __declspec(dllexport)
00013     #else
00014         #define CLOUDX1_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 CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00022 #endif
00023 #include <stdint.h>
00024
00025 #define CLOUDX1_HAS_WEBSERVER 1
00026 #define CLOUDX1_HAS_CLOUDBUS 0
00027
00028 #ifdef __cplusplus
00029 extern "C" {
00030 #endif // __cplusplus
00031
00032 // ------------------------------------------------------------
00046 #define CLOUDX1_AUDIOFRAME_COUNT      64
00052 #define CLOUDX1_CHANNEL_COUNT          2
00058 #define CLOUDX1_AUDIO_INPUT_COUNT      1
00064 #define CLOUDX1_AUDIO_OUTPUT_COUNT     1
00070 #define CLOUDX1_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 CLOUDX1_DECL const struct SOUND4_ProcessInfo* SOUND4_GetProcessInfo();
00093
00099 CLOUDX1_DECL const char *cloudx1_GetVersion();
00100
00113 CLOUDX1_DECL unsigned int cloudx1_GetChunkSizeInFrames();
00114
00123 CLOUDX1_DECL unsigned int* cloudx1_GetPossibleChunkSizeInFrames();
00124
00132 CLOUDX1_DECL unsigned int cloudx1_GetChannelCount();
00133
00142 CLOUDX1_DECL unsigned int cloudx1_GetAudioInputCount();
00143
00152 CLOUDX1_DECL unsigned int cloudx1_GetAudioOutputCount();
00153
00161 CLOUDX1_DECL unsigned int cloudx1_GetSampleRate();
00162
00166 enum cloudx1_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 (*cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *);
00186
00196 CLOUDX1_DECL void cloudx1_SetLoggerCallback( cloudx1_loggerfn logger );
00197
00205 CLOUDX1_DECL void cloudx1_SetLogSeverity( enum cloudx1_LogSeverity severity );
00206
00209 // ------------------------------------------------------------
00217 // ------------------------------------------------------------
00221 struct cloudx1_CInstance;
00222
00226 struct cloudx1_CParameters;
00227
00235 CLOUDX1_DECL struct cloudx1_CParameters *cloudx1_NewParameters();
00236
00240 CLOUDX1_DECL void cloudx1_FreeParameters(struct cloudx1_CParameters *params);
00241
00253 CLOUDX1_DECL void cloudx1_SetParameter(struct cloudx1_CParameters *params, const char *name, const
      char *value);
00254
00260 CLOUDX1_DECL const char *cloudx1_GetParameter(struct cloudx1_CParameters *params, const char *name);
00261
00267 CLOUDX1_DECL void cloudx1_FreeParameterValue(const char *value);
00268
00276 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_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 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_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 cloudx1_CParameters *parameters
00298     );
00299
00327 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_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 cloudx1_CParameters *parameters,
00334         unsigned int frames_per_chunk
00335     );
00336
00348 CLOUDX1_DECL unsigned int cloudx1_GetProcessChunkFrames(struct cloudx1_CInstance *instance);
00349
00359 CLOUDX1_DECL int cloudx1_TerminateProcess(struct cloudx1_CInstance *instance);
00360
00366 CLOUDX1_DECL void cloudx1_ExitProcess(struct cloudx1_CInstance *instance);
00367
```

```
00370 #if CLOUDX1_HAS_CLOUDBUS
00371 // ---------------------------------------------------------------
00381 struct cloudx1_CBus;
00382
00388 CLOUDX1_DECL struct cloudx1_CBus* cloudx1_NewBus();
00389
00395 CLOUDX1_DECL void cloudx1_FreeBus(struct cloudx1_CBus *bus);
00396
00406 CLOUDX1_DECL void cloudx1_SetInstanceBus(struct cloudx1_CParameters *params, struct cloudx1_CBus
      *bus);
00407
00408
00410 #endif // CLOUDX1_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 CLOUDX1_DECL void cloudx1_SetInstanceTracing(struct cloudx1_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00436
00444 CLOUDX1_DECL void cloudx1_SetInstanceTracingProcessActivity(struct cloudx1_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 * (*cloudx1_storage_reader) (const fs_char *filename, void* userdata);
00489
00496 typedef void (*cloudx1_storage_reader_free) (char *content, void* userdata);
00497
00506 typedef int (*cloudx1_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00507
00515 typedef int (*cloudx1_storage_exists) (const fs_char *filename, void* userdata);
00516
00524 typedef fs_char** (*cloudx1_storage_getall) (void* userdata);
00525
00532 typedef void (*cloudx1_storage_getall_free) (fs_char**all, void* userdata);
00533
00541 typedef int (*cloudx1_storage_remove) (const fs_char *filename, void* userdata);
00542
00551 typedef int (*cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00552
00570 CLOUDX1_DECL void cloudx1_SetPresetManager(struct cloudx1_CParameters *params
00571     , cloudx1_storage_reader reader, cloudx1_storage_reader_free
00572     , cloudx1_storage_writer writer
00573     , cloudx1_storage_exists exists
00574     , cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free
00575     , cloudx1_storage_remove remove
00576     , cloudx1_storage_rename rename
00577     , int IsReadOnly
00578     , void* userdata
00579     );
00580
00584 enum cloudx1_PresetChange_Kind {
00585     change_kind_created = 1,
00586     change_kind_modified   ,
00587     change_kind_deleted    ,
00588 };
00589
00598 CLOUDX1_DECL void cloudx1_PresetManager_InformChange(struct cloudx1_CInstance *instance, const fs_char
      *relative_path, enum cloudx1_PresetChange_Kind change_kind);
00599
00602 // ---------------------------------------------------------------
00610 // ---------------------------------------------------------------
00611
00628 CLOUDX1_DECL void cloudx1_SetMetadata(struct cloudx1_CInstance *instance, const char* key, const char*
      value);
00629
00640 CLOUDX1_DECL void cloudx1_SetMetadataMulti(struct cloudx1_CInstance *instance, const char** keyvalue);
00641
00649 CLOUDX1_DECL const char** cloudx1_GetMetadataInfos(struct cloudx1_CInstance *instance);
00650
```

```
00657 CLOUDX1_DECL void cloudx1_FreeMetadataInfos(struct cloudx1_CInstance *instance, const char** infos);
00658
00661 // ---------------------------------------------------------------
00668 // ---------------------------------------------------------------
00669
00682 CLOUDX1_DECL void cloudx1_StartUpdateThread(struct cloudx1_CInstance *instance, unsigned int port);
00683
00689 CLOUDX1_DECL void cloudx1_StopUpdateThread(struct cloudx1_CInstance *instance);
00690
00698 CLOUDX1_DECL int cloudx1_WaitUpdateThreadReady(struct cloudx1_CInstance *instance, int milliseconds);
00699
00702 // ---------------------------------------------------------------
00709 // ---------------------------------------------------------------
00731 CLOUDX1_DECL void cloudx1_ProcessAudio(struct cloudx1_CInstance *instance, const float *input, float
      *output );
00732
00751 CLOUDX1_DECL void cloudx1_ProcessAudio_Planar(struct cloudx1_CInstance *instance
00752     , float const * const *input
00753     , float * const *output);
00754
00767 CLOUDX1_DECL float *cloudx1_GetBufferIn(struct cloudx1_CInstance *instance);
00768
00781 CLOUDX1_DECL float *cloudx1_GetBufferOut(struct cloudx1_CInstance *instance);
00782
00792 CLOUDX1_DECL unsigned int cloudx1_GetEstimatedDelay(struct cloudx1_CInstance *instance);
00793
00796 // ---------------------------------------------------------------
00805 // ---------------------------------------------------------------
00806     #ifdef _MSC_VER
00807         #define __BYTE_ORDER__  REG_DWORD
00808         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00809         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00810     #endif
00811
00815 enum cloudx1_SampleFormat {
00816     INVALID_FORMAT=0,
00817     S16_LE,
00818     S16_BE,
00819     S24_LE,
00820     S24_BE,
00821     S32_LE,
00822     S32_BE,
00823     F32_LE,
00824     F32_BE,
00825
00826     // Native definitions, only links to the physical ones
00827     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00828         S16_NATIVE=S16_LE,
00829         S24_NATIVE=S24_LE,
00830         S32_NATIVE=S32_LE,
00831         F32_NATIVE=F32_LE,
00832     #else
00833         S16_NATIVE=S16_BE,
00834         S24_NATIVE=S24_BE,
00835         S32_NATIVE=S32_BE,
00836         F32_NATIVE=F32_BE,
00837     #endif
00838 };
00839
00846 CLOUDX1_DECL const char *cloudx1_GetFormatName(const enum cloudx1_SampleFormat fmt);
00847
00854 CLOUDX1_DECL enum cloudx1_SampleFormat cloudx1_GetFormatFromName(const char *name);
00855
00862 CLOUDX1_DECL unsigned int cloudx1_GetBytesFromFormat(const enum cloudx1_SampleFormat fmt);
00863
00864
00871 CLOUDX1_DECL unsigned int cloudx1_GetMaxPacketFrame(struct cloudx1_CInstance *instance);
00872
00883 CLOUDX1_DECL unsigned int cloudx1_AddAudio(struct cloudx1_CInstance *instance, const uint8_t *payload,
      unsigned int nFrame, enum cloudx1_SampleFormat fmt );
00884
00891 CLOUDX1_DECL unsigned int cloudx1_AddPadAudio(struct cloudx1_CInstance *instance);
00892
00899 CLOUDX1_DECL unsigned int cloudx1_GetOutputCount(struct cloudx1_CInstance *instance);
00900
00910 CLOUDX1_DECL unsigned int cloudx1_GetAudio(struct cloudx1_CInstance *instance, uint8_t *payload,
      unsigned int max_nFrame, enum cloudx1_SampleFormat fmt );
00911
00915 // ---------------------------------------------------------------
00922 // ---------------------------------------------------------------
00933 CLOUDX1_DECL void cloudx1_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
00934
00945 CLOUDX1_DECL void cloudx1_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
00946
00956 CLOUDX1_DECL void cloudx1_StereoToMono(const float *input, float *output, size_t nFrame);
```

```
00957
00967 CLOUDX1_DECL void cloudx1_MonoToStereo(const float *input, float *output, size_t nFrame);
00968
00979 CLOUDX1_DECL void cloudx1_StereoToMono_Planar(const float *inputL, const float *inputR, float *output,
      size_t nFrame);
00980
00991 CLOUDX1_DECL void cloudx1_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame);
00992
00999 CLOUDX1_DECL void cloudx1_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
01000
01007 CLOUDX1_DECL void cloudx1_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
01008
01011 // ------------------------------------------------------------------
01018 // ------------------------------------------------------------------
01022 struct cloudx1_CClientInstance;
01023
01032 CLOUDX1_DECL struct cloudx1_CClientInstance *cloudx1_NewClient(struct cloudx1_CInstance *instance);
01033
01039 CLOUDX1_DECL void cloudx1_DeleteClient(struct cloudx1_CClientInstance *client);
01040
01054 CLOUDX1_DECL const char *cloudx1_ProcessJson(struct cloudx1_CClientInstance *client, const char
      *json_str, int *need_save);
01055
01061 CLOUDX1_DECL void cloudx1_FreeJsonAnswer(const char *json_str);
01062
01074 CLOUDX1_DECL int cloudx1_SaveState(struct cloudx1_CInstance *instance);
01075
01078 #if CLOUDX1_HAS_WEBSERVER
01079 // ------------------------------------------------------------------
01094 // ------------------------------------------------------------------
01102 CLOUDX1_DECL uint64_t cloudx1_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port);
01103
01104 // ------------------------------------------------------------------
01124 CLOUDX1_DECL uint64_t cloudx1_Webserver_tcp2(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port, const struct cloudx1_CParameters *parameters);
01125
01143 CLOUDX1_DECL uint64_t cloudx1_Webserver(unsigned int listenport, unsigned int listenport_secure,
      struct cloudx1_CInstance *instance);
01144
01154 CLOUDX1_DECL int cloudx1_Webserver_Stop(uint64_t id, int timeout_ms);
01155
01164 CLOUDX1_DECL int cloudx1_Webserver_Status(uint64_t id);
01165
01176 CLOUDX1_DECL void cloudx1_Webserver_SetAppHealth(struct cloudx1_CInstance *instance, int httpcode,
      const char *contenttype, const char *content);
01177
01186 CLOUDX1_DECL void cloudx1_Webserver_GetAppHealth(struct cloudx1_CInstance *instance, int* httpcode,
      char** contenttype, char** content);
01187
01193 CLOUDX1_DECL void cloudx1_Webserver_FreeString(char *str);
01194
01199 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
01200
01204 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
01208 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
01209
01211 #endif // CLOUDX1_HAS_WEBSERVER
01212
01213 #ifdef __cplusplus
01214 } // extern "C"
01215 #endif // __cplusplus
```

15.21 sound4.x1.cloud.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.x1.cloud.h"
#include <stringapiset.h>
#include <wchar.h>
```

Classes

- struct sound4::cloudx1::helper::SampleFormat< T >

  Helper for Sample format types.
- class sound4::cloudx1::CPresetLoader

  Custom preset handler helper.
- class sound4::cloudx1::CInstance

  Instance handling class.
- class sound4::cloudx1::CInstance::CClient

Namespaces

- namespace sound4
- namespace sound4::cloudx1
- namespace sound4::cloudx1::helper

Typedefs

- using sound4::cloudx1::SampleFormat = cloudx1_SampleFormat
- using sound4::cloudx1::LogSeverity = cloudx1_LogSeverity
- using sound4::cloudx1::log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using sound4::cloudx1::PresetChange_Kind = cloudx1_PresetChange_Kind

Functions

- static char * cloudx1_custom_reader (const fs_char *filename, void *userdata)
- static void cloudx1_custom_reader_free (char *content, void *userdata)
- static int cloudx1_custom_writer (const fs_char *filename, const char *content, void *userdata)
- static int cloudx1_custom_exists (const fs_char *filename, void *userdata)
- static fs_char ** cloudx1_custom_getall (void *userdata)
- static void cloudx1_custom_getall_free (fs_char **all, void *userdata)
- static int cloudx1_custom_remove (const fs_char *filename, void *userdata)
- static int cloudx1_custom_rename (const fs_char *from, const fs_char *to, void *userdata)
- static std::string sound4::cloudx1::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::cloudx1::helper::UTF8ToWString (const std::string &str)
- static void sound4::cloudx1::helper::AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, cloudx1_SampleFormat fmt)
- static void sound4::cloudx1::helper::AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, cloudx1_SampleFormat fmt)
- static void sound4::cloudx1::helper::StereoToMono (const float *input, float *output, size_t nFrame)
- static void sound4::cloudx1::helper::MonoToStereo (const float *input, float *output, size_t nFrame)
- static void sound4::cloudx1::helper::StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)

- static void sound4::cloudx1::helper::MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- static void sound4::cloudx1::helper::AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- static void sound4::cloudx1::helper::AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- static std::string sound4::cloudx1::GetVersion ()
- static unsigned int sound4::cloudx1::GetChunkSizeInFrames ()
- static std::vector< unsigned int > sound4::cloudx1::GetPossibleChunkSizeInFrames ()
- static unsigned int sound4::cloudx1::GetChannelCount ()
- static unsigned int sound4::cloudx1::GetAudioInputCount ()
- static unsigned int sound4::cloudx1::GetAudioOutputCount ()
- static unsigned int sound4::cloudx1::GetSampleRate ()
- static void sound4::cloudx1::SanityCheck (bool a_bCheckFrames=true)
- static std::string sound4::cloudx1::GetFormatName (const SampleFormat fmt)
- static SampleFormat sound4::cloudx1::GetFormatFromName (const std::string &name)
- static unsigned int sound4::cloudx1::GetBytesFromFormat (const SampleFormat fmt)
- static void sound4::cloudx1::_log_cb_c (cloudx1_LogSeverity severity, const char *c_msg)
- static void sound4::cloudx1::SetLogSeverity (LogSeverity severity)
- static void sound4::cloudx1::SetLoggerCallback (log_cb_t cb)

Variables

- static constexpr const char * sound4::cloudx1::process_name = "SOUND4 x1.CLOUD"
- static constexpr const char * sound4::cloudx1::process_shortname = "x1"
- const size_t sound4::cloudx1::ChunkFrames = 64
- const size_t sound4::cloudx1::InputChannels = 2 * 1
- const size_t sound4::cloudx1::OutputChannels = 2 * 1
- const size_t sound4::cloudx1::InputSampleSize =ChunkFrames*InputChannels
- const size_t sound4::cloudx1::OutputSampleSize =ChunkFrames*OutputChannels
- const size_t sound4::cloudx1::SampleRate = 48000
- static log_cb_t sound4::cloudx1::_log_cb

15.21.1 Function Documentation

15.21.1.1 cloudx1_custom_exists()

```
static int cloudx1_custom_exists (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1082 of file sound4.x1.cloud.hpp.

15.21.1.2 cloudx1_custom_getall()

```
static fs_char ** cloudx1_custom_getall (
            void * userdata )  [static]
```

Definition at line 1087 of file sound4.x1.cloud.hpp.

15.21.1.3 cloudx1_custom_getall_free()

```
static void cloudx1_custom_getall_free (
            fs_char ** all,
            void * userdata )  [static]
```

Definition at line 1097 of file sound4.x1.cloud.hpp.

15.21.1.4 cloudx1_custom_reader()

```
static char * cloudx1_custom_reader (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1068 of file sound4.x1.cloud.hpp.

15.21.1.5 cloudx1_custom_reader_free()

```
static void cloudx1_custom_reader_free (
            char * content,
            void * userdata )  [static]
```

Definition at line 1073 of file sound4.x1.cloud.hpp.

15.21.1.6 cloudx1_custom_remove()

```
static int cloudx1_custom_remove (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1104 of file sound4.x1.cloud.hpp.

15.21.1.7 cloudx1_custom_rename()

```
static int cloudx1_custom_rename (
            const fs_char * from,
            const fs_char * to,
            void * userdata )  [static]
```

Definition at line 1109 of file sound4.x1.cloud.hpp.

15.21.1.8 cloudx1_custom_writer()

```
static int cloudx1_custom_writer (
            const fs_char * filename,
            const char * content,
            void * userdata )  [static]
```

Definition at line 1077 of file sound4.x1.cloud.hpp.

15.22 sound4.x1.cloud.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.x1.cloud.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 *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00037     static void cloudx1_custom_reader_free(char *content, void* userdata);
00038     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00039     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00040     static fs_char** cloudx1_custom_getall(void* userdata);
00041     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00042     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00043     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00044 };
00045
00049 namespace sound4 {
00053 namespace cloudx1 {
00054     static constexpr const char *process_name = "SOUND4 x1.CLOUD";
00055     static constexpr const char *process_shortname = "x1";
00056     using SampleFormat = cloudx1_SampleFormat;
00061     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00066     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00071     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00076     const size_t InputSampleSize=ChunkFrames*InputChannels;
00081     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00086     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
00087
00091     namespace helper {
00092         #ifdef _WIN32
00098         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00099             if (wstr.empty()) return {};
00100             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00101             std::string str(size, 0);
00102             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00103             return str;
00104         }
00110         static inline std::wstring UTF8ToWString(const std::string& str) {
00111             if (str.empty()) return std::wstring();
00112             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00113             std::wstring wstr(size, 0);
00114             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00115             return wstr;
00116         }
00117         #endif // _WIN32
00118
00124             template<typename T>
00125         struct SampleFormat {
00126         };
00127             template<>
00128         struct SampleFormat<int16_t> {
00129             const cloudx1_SampleFormat format = S16_NATIVE;
00130         };
00131             template<>
00132         struct SampleFormat<int32_t> {
00133             const cloudx1_SampleFormat format = S32_NATIVE;
```

```
00134         };
00135             template<>
00136         struct SampleFormat<float> {
00137             const cloudx1_SampleFormat format = F32_NATIVE;
00138         };
00139 
00140         // ------------------------------------------------------------
00151         static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00152             cloudx1_AudioConvertFrom(payload, output, nSpl, fmt);
00153         }
00154 
00165         static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00166             cloudx1_AudioConvertTo(input, payload, nSpl, fmt);
00167         }
00168 
00178         static inline void StereoToMono(const float *input, float *output, size_t nFrame) {
00179             cloudx1_StereoToMono(input, output, nFrame);
00180         }
00181 
00191         static inline void MonoToStereo(const float *input, float *output, size_t nFrame) {
00192             cloudx1_MonoToStereo(input, output, nFrame);
00193         }
00194 
00205         static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float
      *output, size_t nFrame) {
00206             cloudx1_StereoToMono_Planar(inputL, inputR, output, nFrame);
00207         }
00208 
00219         static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame) {
00220             cloudx1_MonoToStereo_Planar(input, outputL, outputR, nFrame);
00221         }
00222 
00229         static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
00230             cloudx1_AudioMonoFromLiveStereo(payload, output);
00231         }
00232 
00239         static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
00240             cloudx1_AudioMonoToLiveStereo(input, payload);
00241         }
00242 
00243     }; // namespace helper
00244 
00250     static inline std::string GetVersion()
00251         {
00252             return std::string(cloudx1_GetVersion());
00253         }
00254 
00265     static inline unsigned int GetChunkSizeInFrames()
00266         {
00267             return cloudx1_GetChunkSizeInFrames();
00268         }
00269 
00278     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00279         {
00280             std::vector<unsigned int> list;
00281             for (unsigned int* src=cloudx1_GetPossibleChunkSizeInFrames();*src;src++) {
00282                 list.push_back(*src);
00283             }
00284             return list;
00285         }
00286 
00292     static inline unsigned int GetChannelCount()
00293         {
00294             return cloudx1_GetChannelCount();
00295         }
00296 
00304     static inline unsigned int GetAudioInputCount()
00305         {
00306             return cloudx1_GetAudioInputCount();
00307         }
00308 
00316     static inline unsigned int GetAudioOutputCount()
00317         {
00318             return cloudx1_GetAudioOutputCount();
00319         }
00320 
00328     static inline unsigned int GetSampleRate()
00329         {
00330             return cloudx1_GetSampleRate();
00331         }
00332 
00338     static inline void SanityCheck(bool a_bCheckFrames = true)
00339     {
00340         // Do some sanity checks
```

```
00341         if (cloudx1_GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00342             throw std::runtime_error("Bad library sampling rate");
00343         }
00344         if (a_bCheckFrames && cloudx1_GetChunkSizeInFrames()!=ChunkFrames) {
00345             throw std::runtime_error("Bad library frame size");
00346         }
00347         if (cloudx1_GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00348             throw std::runtime_error("Bad library channel count");
00349         }
00350         if (cloudx1_GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00351             throw std::runtime_error("Bad library input count");
00352         }
00353         if (cloudx1_GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00354             throw std::runtime_error("Bad library output count");
00355         }
00356     }
00357
00364     static inline std::string GetFormatName(const SampleFormat fmt)
00365         {
00366             return std::string(cloudx1_GetFormatName(fmt));
00367         }
00368
00375     static inline SampleFormat GetFormatFromName(const std::string& name)
00376         {
00377             return cloudx1_GetFormatFromName(name.c_str());
00378         }
00379
00386     static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
00387         {
00388             return cloudx1_GetBytesFromFormat(fmt);
00389         }
00390
00394     using LogSeverity = cloudx1_LogSeverity;
00398     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00399
00403     static log_cb_t _log_cb;
00404     extern "C" {
00405         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00406             _log_cb(severity, std::string(c_msg));
00407         }
00408     }
00412     static inline void SetLogSeverity(LogSeverity severity)
00413         {
00414             cloudx1_SetLogSeverity(severity);
00415         }
00419     static inline void SetLoggerCallback(log_cb_t cb)
00420         {
00421             _log_cb=cb;
00422             cloudx1_SetLoggerCallback(&_log_cb_c);
00423         }
00424
00425 #if CLOUDX1_HAS_CLOUDBUS
00431     class CBus {
00432         public:
00433             CBus()
00434                 : m_bus(cloudx1_NewBus()) {}
00435             ~CBus()
00436                 {
00437                     cloudx1_FreeBus(m_bus);
00438                     m_bus = nullptr;
00439                 }
00440             cloudx1_CBus *Get() const { return m_bus; }
00441         private:
00442             cloudx1_CBus *m_bus;
00443     };
00444 #endif // CLOUDX1_HAS_CLOUDBUS
00445
00451     class CPresetLoader {
00452     public:
00453         CPresetLoader() = default;
00454         virtual ~CPresetLoader() = default;
00455
00456         virtual bool IsReadOnly() = 0;
00457         virtual bool Exists(const std::filesystem::path &name) = 0;
00458         virtual bool Remove(const std::filesystem::path &name) = 0;
00459         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00460         virtual std::vector<std::filesystem::path> GetAll() = 0;
00461         virtual std::string Read(const std::filesystem::path &filename) = 0;
00462         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00463     };
00464     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00465
00471     class CInstance {
00472     public:
00476         CInstance()
00477             {
00478                 params = cloudx1_NewParameters();
```

```
00479                 if (!params) throw std::bad_alloc();
00480             }
00481         // non-copyable
00482         CInstance( const CInstance& ) = delete;
00483         CInstance& operator=( const CInstance& ) = delete;
00487         virtual ~CInstance()
00488             {
00489                 Stop();
00490                 if (params) cloudx1_FreeParameters(params);
00491             }
00495        bool IsOk() const { return instance!=nullptr; }
00505         void SetParam(const std::string& name, const std::string& value)
00506             {
00507                 assert(params);
00508                 cloudx1_SetParameter(params, name.c_str(), value.c_str());
00509             }
00516         std::string GetParam(const std::string& name)
00517             {
00518                 assert(params);
00519                 auto c_value=cloudx1_GetParameter(params, name.c_str());
00520                 std::string ret(c_value);
00521                 cloudx1_FreeParameterValue(c_value);
00522                 return ret;
00523             }
00524
00525 #if CLOUDX1_HAS_CLOUDBUS
00534             void SetBus(const CBus& bus) {
00535                 assert(params);
00536                 cloudx1_SetInstanceBus(params,bus.Get());
00537             }
00538 #endif // CLOUDX1_HAS_CLOUDBUS
00539
00550         void SetPresetManager(CPresetLoader *preset_manager)
00551             {
00552                 if (!preset_manager) return;
00553                 assert(params);
00554                 cloudx1_SetPresetManager(params,
00555                     cloudx1_custom_reader,
00556                     cloudx1_custom_reader_free,
00557                     cloudx1_custom_writer,
00558                     cloudx1_custom_exists,
00559                     cloudx1_custom_getall,
00560                     cloudx1_custom_getall_free,
00561                     cloudx1_custom_remove,
00562                     cloudx1_custom_rename,
00563                     preset_manager->IsReadOnly(),
00564                     preset_manager
00565                 );
00566             }
00567
00590         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=64)
00591             {
00592                 assert(!instance);
00593                 std::string l_save_path;
00594                 if (!save_path.empty()) {
00595                     l_save_path = save_path.u8string();
00596                 }
00597                 instance = cloudx1_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);
00598                 if (!instance) return false;
00599                 if (!init_metadata.empty()) {
00600                     for (auto&& [key,value]: init_metadata) {
00601                         cloudx1_SetMetadata(instance, key.c_str(), value.c_str());
00602                     }
00603                     init_metadata.clear();
00604                 }
00605                 update_thread = std::thread([this, json_port](){
00606                     OnUpdateThreadStart();
00607                     cloudx1_StartUpdateThread(instance, json_port);
00608                     OnUpdateThreadStop();
00609                 });
00610                 int timeout=5000;
00611                 #ifdef DEBUG
00612                     timeout=60*1000; // For debugger pauses
00613                 #endif
00614                 if (cloudx1_WaitUpdateThreadReady(instance, timeout)<0) {
00615                     return false;
00616                 }
00617                 return true;
00618             }
00619
00627         unsigned int GetChunkFrames() {
00628             if (instance) {
00629                 return cloudx1_GetProcessChunkFrames(instance);
00630             }
```

```
00631             return 0;
00632         }
00633
00641         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
00642             {
00643                 cloudx1_PresetManager_InformChange(instance, relative_path.c_str(),
      (cloudx1_PresetChange_Kind) change_kind);
00644             }
00645
00657         void SetMetadata(const std::string& key, const char* value)
00658             {
00659                 if (instance) {
00660                     cloudx1_SetMetadata(instance, key.c_str(), value);
00661                 } else {
00662                     init_metadata.push_back( {key, value} );
00663                 }
00664             }
00665
00675         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
00676             {
00677                 if (instance) {
00678                     typedef const char* pchar;
00679                     pchar* keyvalue=new pchar[2*list.size()+1];
00680                     size_t n=0;
00681                     for (auto&& [key,value]: list) {
00682                         keyvalue[2*n+0]=key.c_str();
00683                         keyvalue[2*n+1]=value;
00684                         n++;
00685                     }
00686                     keyvalue[2*n+0]=nullptr;
00687
00688                     cloudx1_SetMetadataMulti(instance, keyvalue);
00689
00690                     delete[] keyvalue;
00691                 } else {
00692                     for (auto&& [key,value]: list) {
00693                         if (value) {
00694                             init_metadata.push_back( {key, value} );
00695                         }
00696                     }
00697                 }
00698             }
00699
00705         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
00706             {
00707                 std::vector< std::tuple<std::string,std::string> > values;
00708                 const char** c_values = cloudx1_GetMetadataInfos(instance);
00709                 if (c_values) {
00710                     for (const char** c_value=c_values; *c_value; ) {
00711                         std::string key(*c_value);
00712                         c_value++;
00713                         if (*c_value) {
00714                             std::string descr(*c_value);
00715                             c_value++;
00716                             values.push_back( {key,descr} );
00717                         }
00718                     }
00719                     cloudx1_FreeMetadataInfos(instance, c_values);
00720                 }
00721                 return values;
00722             }
00723
00727         virtual void OnUpdateThreadStart() {}
00731         virtual void OnUpdateThreadStop() {}
00732
00733 #if CLOUDX1_HAS_WEBSERVER
00747         bool StartWebServer(int http_port, int https_port=0)
00748             {
00749                 assert(instance);
00750                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00751                 webserver = cloudx1_Webserver(http_port,https_port,instance);
00752                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00753                     return false;
00754                 }
00755                 int web_status=cloudx1_Webserver_Status(webserver);
00756                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00757                     return false;
00758                 }
00759                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00760                     return false;
00761                 }
00762                 return true;
00763             }
00764
00772             void StopWebServer(int timeout_ms = 1000)
00773             {
```

```
00774                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00775                     cloudx1_Webserver_Stop(webserver, timeout_ms);
00776                     webserver = SOUND4_INVALID_WEBSERVER_ID;
00777                 }
00778             }
00779
00783         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
00784             {
00785                 assert(instance);
00786                 cloudx1_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
00787             }
00791         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
00792             {
00793                 assert(instance);
00794                 char* c_contenttype=nullptr;
00795                 char* c_content=nullptr;
00796                 cloudx1_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
00797                 contenttype=c_contenttype;
00798                 content=c_content;
00799                 cloudx1_Webserver_FreeString(c_contenttype);
00800                 cloudx1_Webserver_FreeString(c_content);
00801         }
00802 #endif // CLOUDX1_HAS_WEBSERVER
00803
00812         int Terminate()
00813             {
00814                 if (!instance) return 0;
00815                 return cloudx1_TerminateProcess(instance);
00816             }
00817
00821         void Stop()
00822             {
00823                 if (!instance) return;
00824 #if CLOUDX1_HAS_WEBSERVER
00825                 StopWebServer();
00826 #endif // CLOUDX1_HAS_WEBSERVER
00827                 if (update_thread.joinable()) {
00828                     cloudx1_StopUpdateThread(instance); // this will exit the update thread
00829                     update_thread.join();
00830                 }
00831                 cloudx1_ExitProcess(instance);
00832                 instance=nullptr;
00833             }
00834
00844         unsigned int GetEstimatedDelay()
00845             {
00846                 assert(instance);
00847                 return cloudx1_GetEstimatedDelay(instance);
00848             }
00849
00862         std::array<float,InputSampleSize>& GetBufferIn()
00863             {
00864                 assert(instance);
00865                 float *buf=cloudx1_GetBufferIn(instance);
00866                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
00867
00868             }
00880         std::array<float,OutputSampleSize>& GetBufferOut()
00881             {
00882                 assert(instance);
00883                 float *buf= cloudx1_GetBufferOut(instance);
00884                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00885             }
00889         void ProcessAudio()
00890             {
00891                 assert(instance);
00892                 cloudx1_ProcessAudio(instance, cloudx1_GetBufferIn(instance),
      cloudx1_GetBufferOut(instance));
00893             }
00905         void ProcessAudio_Planar(float const * const *input, float * const *output)
00906             {
00907                 assert(instance);
00908                 cloudx1_ProcessAudio_Planar(instance, input, output);
00909             }
00910
00922             template<typename T>
00923         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00924             {
00925                 assert(instance);
00926                 std::vector<T> output;
00927                 unsigned int out_offset=0;
00928                 unsigned int in_offset=0;
00929                 unsigned int todo = input.size();
00930                 while (todo>0) {
00931                     unsigned int left = AddAudio(&input[in_offset], todo);
```

```
00932                     unsigned int out_avail = cloudx1_GetOutputCount(instance);
00933                     output.resize(out_offset + out_avail);
00934                     GetAudio(&output[out_offset], out_avail);
00935                     out_offset+=out_avail;
00936                     in_offset += todo-left;
00937                     todo=left;
00938                 }
00939                 return output;
00940             }
00941
00947         class CClient {
00948             public:
00949                 CClient(cloudx1_CInstance* instance)
00950                     {
00951                         assert(instance);
00952                         client=cloudx1_NewClient(instance);
00953                         if (!client) throw std::bad_alloc();
00954                     }
00955                 // non-copyable
00956                 CClient( const CInstance& ) = delete;
00957                 CClient& operator=( const CInstance& ) = delete;
00958                 ~CClient()
00959                     {
00960                         if (client) {
00961                             cloudx1_DeleteClient(client);
00962                             client=nullptr;
00963                         }
00964                     }
00972                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
00973                     {
00974                         assert(client);
00975                         int need_save=0;
00976                         const char *canswer = cloudx1_ProcessJson(client, request.c_str(),
    &need_save);
00977                         if (!canswer) return {};
00978                         std::string answer(canswer);
00979                         cloudx1_FreeJsonAnswer (canswer);
00980                         if (NeedSave) {
00981                             *NeedSave=(need_save!=0);
00982                         }
00983                         return answer;
00984                     }
00985             private:
00986                 cloudx1_CClientInstance *client = nullptr;
00987         };
00995         std::shared_ptr<CClient> NewClient()
00996             {
00997                 assert(instance);
00998                 return std::make_shared<CClient>(instance);
00999             }
01000
01011         bool SaveState() {
01012             assert(instance);
01013             if (cloudx1_SaveState(instance)!=0) {
01014                 return false;
01015             }
01016             return true;
01017         }
01018 #ifdef _WIN32
01025         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01026             cloudx1_SetInstanceTracing(params, tracing_provider, activity_guid);
01027         }
01034         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01035             assert(instance);
01036             cloudx1_SetInstanceTracingProcessActivity(instance, activity_guid);
01037         }
01038 #endif // _WIN32
01039     protected:
01040             template<typename T>
01041         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01042             {
01043                 assert(instance);
01044                 return cloudx1_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
    helper::SampleFormat<T>::format);
01045             }
01046             template<typename T>
01047         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01048             {
01049                 assert(instance);
01050                 return cloudx1_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
    helper::SampleFormat<T>::format);
01051             }
01052
01053     private:
01054         cloudx1_CParameters* params = nullptr;
01055         cloudx1_CInstance* instance = nullptr;
01056         std::thread update_thread;
```

```
01057         std::vector< std::pair<std::string,std::string> > init_metadata;
01058 #if CLOUDX1_HAS_WEBSERVER
01059         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01060 #endif // CLOUDX1_HAS_WEBSERVER
01061     };
01062
01063 }; // namespace cloudx1
01064 }; // namespace sound4
01065
01066 // bridge C callbacks to CPresetLoader
01067 extern "C" {
01068     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01069         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01070         std::string content=preset_loader->Read(filename);
01071         return strdup(content.c_str());
01072     }
01073     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01074         if (!content) return;
01075         free(content);
01076     }
01077     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01078         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01079         auto res=preset_loader->Write(filename,content);
01080         return res?0:-1;
01081     }
01082     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01083         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01084         auto res=preset_loader->Exists(filename);
01085         return res?0:-1;
01086     }
01087     static fs_char** cloudx1_custom_getall(void* userdata) {
01088         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01089         auto res=preset_loader->GetAll();
01090         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01091         for (size_t n=0;n<res.size();n++) {
01092             all[n]=fs_strdup(res[n].c_str());
01093         }
01094         all[res.size()]=nullptr;
01095         return all;
01096     }
01097     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01098         if (!all) return;
01099         for (fs_char** one=all;*one;one++) {
01100             free(*one);
01101         }
01102         free(all);
01103     }
01104     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01105         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01106         auto res=preset_loader->Remove(filename);
01107         return res?0:-1;
01108     }
01109     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01110         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01111         auto res=preset_loader->Rename(from,to);
01112         return res?0:-1;
01113     }
01114
01115 };
```

15.23 sound4.x1.cloud_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.x1.cloud.h"
#include <stringapiset.h>
#include <wchar.h>
```

Classes

- class sound4::cloudx1::dyn::helper::DynFuncHolder< T >

  Helper to wrap a function pointer.
- class sound4::cloudx1::dyn::helper::CDynLoader

  Helper to load dynamic library.
- class sound4::cloudx1::dyn::CDynLib

  Dynamic library interface.
- struct sound4::cloudx1::dyn::helper::SampleFormat< T >

  Helper for Sample format types.
- class sound4::cloudx1::dyn::CPresetLoader

  Custom preset handler helper.
- class sound4::cloudx1::dyn::CInstance

  Instance handling class.
- class sound4::cloudx1::dyn::CInstance::CClient

Namespaces

- namespace sound4
- namespace sound4::cloudx1
- namespace sound4::cloudx1::dyn
- namespace sound4::cloudx1::dyn::helper

Macros

- #define UNIXLIKE 0

Typedefs

- using sound4::cloudx1::dyn::SampleFormat = cloudx1_SampleFormat
- using sound4::cloudx1::dyn::LogSeverity = cloudx1_LogSeverity
- using sound4::cloudx1::dyn::log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using sound4::cloudx1::dyn::PresetChange_Kind = cloudx1_PresetChange_Kind

Functions

- static char * cloudx1_custom_reader (const fs_char *filename, void *userdata)
- static void cloudx1_custom_reader_free (char *content, void *userdata)
- static int cloudx1_custom_writer (const fs_char *filename, const char *content, void *userdata)
- static int cloudx1_custom_exists (const fs_char *filename, void *userdata)
- static fs_char ** cloudx1_custom_getall (void *userdata)
- static void cloudx1_custom_getall_free (fs_char **all, void *userdata)
- static int cloudx1_custom_remove (const fs_char *filename, void *userdata)
- static int cloudx1_custom_rename (const fs_char *from, const fs_char *to, void *userdata)
- static std::string sound4::cloudx1::dyn::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::cloudx1::dyn::helper::UTF8ToWString (const std::string &str)
- static void sound4::cloudx1::dyn::helper::AudioConvertFrom (CDynLib &dynlib, const uint8_t *payload, float *output, size_t nSpl, cloudx1_SampleFormat fmt)
- static void sound4::cloudx1::dyn::helper::AudioConvertTo (CDynLib &dynlib, const float *input, uint8_t *payload, size_t nSpl, cloudx1_SampleFormat fmt)
- static void sound4::cloudx1::dyn::helper::StereoToMono (CDynLib &dynlib, const float *input, float *output, size_t nFrame)
- static void sound4::cloudx1::dyn::helper::MonoToStereo (CDynLib &dynlib, const float *input, float *output, size_t nFrame)
- static void sound4::cloudx1::dyn::helper::StereoToMono_Planar (CDynLib &dynlib, const float *inputL, const float *inputR, float *output, size_t nFrame)
- static void sound4::cloudx1::dyn::helper::MonoToStereo_Planar (CDynLib &dynlib, const float *input, float *outputL, float *outputR, size_t nFrame)
- static void sound4::cloudx1::dyn::helper::AudioMonoFromLiveStereo (CDynLib &dynlib, const uint8_t *payload, float *output)
- static void sound4::cloudx1::dyn::helper::AudioMonoToLiveStereo (CDynLib &dynlib, const float *input, uint8_t *payload)
- static std::string sound4::cloudx1::dyn::GetVersion (CDynLib &dynlib)
- static unsigned int sound4::cloudx1::dyn::GetChunkSizeInFrames (CDynLib &dynlib)
- static std::vector< unsigned int > sound4::cloudx1::dyn::GetPossibleChunkSizeInFrames (CDynLib &dynlib)
- static unsigned int sound4::cloudx1::dyn::GetChannelCount (CDynLib &dynlib)
- static unsigned int sound4::cloudx1::dyn::GetAudioInputCount (CDynLib &dynlib)
- static unsigned int sound4::cloudx1::dyn::GetAudioOutputCount (CDynLib &dynlib)
- static unsigned int sound4::cloudx1::dyn::GetSampleRate (CDynLib &dynlib)
- static void sound4::cloudx1::dyn::SanityCheck (CDynLib &dynlib, bool a_bCheckFrames=true)
- static std::string sound4::cloudx1::dyn::GetFormatName (CDynLib &dynlib, const SampleFormat fmt)
- static SampleFormat sound4::cloudx1::dyn::GetFormatFromName (CDynLib &dynlib, const std::string &name)
- static unsigned int sound4::cloudx1::dyn::GetBytesFromFormat (CDynLib &dynlib, const SampleFormat fmt)
- static void sound4::cloudx1::dyn::_log_cb_c (cloudx1_LogSeverity severity, const char *c_msg)
- static void sound4::cloudx1::dyn::SetLogSeverity (CDynLib &dynlib, LogSeverity severity)
- static void sound4::cloudx1::dyn::SetLoggerCallback (CDynLib &dynlib, log_cb_t cb)

Variables

- static constexpr const char * sound4::cloudx1::dyn::process_name = "SOUND4 x1.CLOUD"
- static constexpr const char * sound4::cloudx1::dyn::process_shortname = "x1"
- const size_t sound4::cloudx1::dyn::ChunkFrames = 64
- const size_t sound4::cloudx1::dyn::InputChannels = 2 * 1
- const size_t sound4::cloudx1::dyn::OutputChannels = 2 * 1
- const size_t sound4::cloudx1::dyn::InputSampleSize =ChunkFrames*InputChannels
- const size_t sound4::cloudx1::dyn::OutputSampleSize =ChunkFrames*OutputChannels
- const size_t sound4::cloudx1::dyn::SampleRate = 48000
- static log_cb_t sound4::cloudx1::dyn::_log_cb

15.23.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file sound4.x1.cloud_dyn.hpp.

15.23.2 Macro Definition Documentation

15.23.2.1 UNIXLIKE

```
#define UNIXLIKE 0
```

Definition at line 23 of file sound4.x1.cloud_dyn.hpp.

15.23.3 Function Documentation

15.23.3.1 cloudx1_custom_exists()

```
static int cloudx1_custom_exists (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1580 of file sound4.x1.cloud_dyn.hpp.

15.23.3.2 cloudx1_custom_getall()

```
static fs_char ** cloudx1_custom_getall (
            void * userdata )  [static]
```

Definition at line 1585 of file sound4.x1.cloud_dyn.hpp.

15.23.3.3 cloudx1_custom_getall_free()

```
static void cloudx1_custom_getall_free (
            fs_char ** all,
            void * userdata )  [static]
```

Definition at line 1595 of file sound4.x1.cloud_dyn.hpp.

15.23.3.4 cloudx1_custom_reader()

```
static char * cloudx1_custom_reader (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1566 of file sound4.x1.cloud_dyn.hpp.

15.23.3.5 cloudx1_custom_reader_free()

```
static void cloudx1_custom_reader_free (
            char * content,
            void * userdata )  [static]
```

Definition at line 1571 of file sound4.x1.cloud_dyn.hpp.

15.23.3.6 cloudx1_custom_remove()

```
static int cloudx1_custom_remove (
            const fs_char * filename,
            void * userdata )  [static]
```

Definition at line 1602 of file sound4.x1.cloud_dyn.hpp.

15.23.3.7 cloudx1_custom_rename()

```
static int cloudx1_custom_rename (
            const fs_char * from,
            const fs_char * to,
            void * userdata )  [static]
```

Definition at line 1607 of file sound4.x1.cloud_dyn.hpp.

15.23.3.8 cloudx1_custom_writer()

```
static int cloudx1_custom_writer (
            const fs_char * filename,
            const char * content,
            void * userdata )  [static]
```

Definition at line 1575 of file sound4.x1.cloud_dyn.hpp.

15.24 sound4.x1.cloud_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.x1.cloud.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 *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00054     static void cloudx1_custom_reader_free(char *content, void* userdata);
00055     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00056     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00057     static fs_char** cloudx1_custom_getall(void* userdata);
00058     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00059     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00060     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00061 };
00062
00066 namespace sound4 {
00070 namespace cloudx1 {
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                       std::filesystem::path p(rp);
00162                       free(rp);
00163                       return p;
00164                   }
00165               #else
00166                   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
00167                   {
00168                       Close();
00169                       m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00170                       return IsOk();
00171                   }
00172                   void Close() {
00173                       if (IsOk()) {
00174                           // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00175                           // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00176                           #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00177                               FreeLibraryAndExitThread(m_lib, 0);
00178                           #elif defined(SOUND4_CALL_FREELIBRARY)
00179                               FreeLibrary(m_lib);
00180                           #endif // SOUND4_CALL_FREELIBRARY
00181                           m_lib=NULL;
00182                       }
00183                   }
00184                       template <typename T>
00185                   DynFuncHolder<T> GetSymbol(const std::string& name) {
00186                       auto ptr=GetProcAddress(m_lib, name.c_str());
00187                       if (!ptr) {
00188                           throw std::runtime_error("Missing function in library");
00189                       }
00190                       return DynFuncHolder<T>(ptr);
00191                   }
00192                       template <typename T>
00193                   DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00194                       auto ptr=GetProcAddress(m_lib, name.c_str());
00195                       return DynFuncHolder<T>(ptr);
00196                   }
00197                   static std::filesystem::path GetThisLibraryPath(void) {
00198                       static wchar_t path[MAX_PATH]={};
00199                       HMODULE hm = NULL;
00200                       if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00201                               GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00202                               (LPCWSTR) &GetThisLibraryPath, &hm))
00203                       {
00204                           if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00205                               return std::filesystem::path(path);
00206                           }
00207                       }
00208                       return {};
00209                   }
00210               #endif
00211               virtual ~CDynLoader() {
00212                   Close();
00213               }
00214           }; // class CDynLoader
00215       }; // namespace helper
00216
00223       class CDynLib {
00224       private:
00225           helper::CDynLoader m_lib;
```

```
00226         bool m_bOK = false;
00227     public:
00228         CDynLib() = default;
00229 
00230         // All those wrapped functions have the same signature as there C cloudx1_XXX
00231         helper::DynFuncHolder< decltype(cloudx1_GetVersion             ) > GetVersion             ;
00232         helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames   ) > GetChunkSizeInFrames   ;
00233         helper::DynFuncHolder< decltype(cloudx1_GetChannelCount        ) > GetChannelCount        ;
00234         helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount     ) > GetAudioInputCount     ;
00235         helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount    ) > GetAudioOutputCount    ;
00236         helper::DynFuncHolder< decltype(cloudx1_GetSampleRate          ) > GetSampleRate          ;
00237         helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback      ) > SetLoggerCallback      ;
00238         helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity         ) > SetLogSeverity         ;
00239         helper::DynFuncHolder< decltype(cloudx1_NewParameters          ) > NewParameters          ;
00240         helper::DynFuncHolder< decltype(cloudx1_FreeParameters         ) > FreeParameters         ;
00241         helper::DynFuncHolder< decltype(cloudx1_SetParameter           ) > SetParameter           ;
00242         helper::DynFuncHolder< decltype(cloudx1_GetParameter           ) > GetParameter           ;
00243         helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue     ) > FreeParameterValue     ;
00244         helper::DynFuncHolder< decltype(cloudx1_InitProcess            ) > InitProcess            ;
00245         helper::DynFuncHolder< decltype(cloudx1_InitProcess2           ) > InitProcess2           ;
00246         helper::DynFuncHolder< decltype(cloudx1_TerminateProcess       ) > TerminateProcess       ;
00247         helper::DynFuncHolder< decltype(cloudx1_ExitProcess            ) > ExitProcess            ;
00248         helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread      ) > StartUpdateThread      ;
00249         helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread       ) > StopUpdateThread       ;
00250         helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady  ) > WaitUpdateThreadReady  ;
00251         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio           ) > ProcessAudio           ;
00252         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar    ) > ProcessAudio_Planar    ;
00253         helper::DynFuncHolder< decltype(cloudx1_GetBufferIn            ) > GetBufferIn            ;
00254         helper::DynFuncHolder< decltype(cloudx1_GetBufferOut           ) > GetBufferOut           ;
00255         helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay      ) > GetEstimatedDelay      ;
00256         helper::DynFuncHolder< decltype(cloudx1_GetFormatName          ) > GetFormatName          ;
00257         helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName      ) > GetFormatFromName      ;
00258         helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat     ) > GetBytesFromFormat     ;
00259         helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame      ) > GetMaxPacketFrame      ;
00260         helper::DynFuncHolder< decltype(cloudx1_AddAudio               ) > AddAudio               ;
00261         helper::DynFuncHolder< decltype(cloudx1_AddPadAudio            ) > AddPadAudio            ;
00262         helper::DynFuncHolder< decltype(cloudx1_GetOutputCount         ) > GetOutputCount         ;
00263         helper::DynFuncHolder< decltype(cloudx1_GetAudio               ) > GetAudio               ;
00264         helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom       ) > AudioConvertFrom       ;
00265         helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo         ) > AudioConvertTo         ;
00266         helper::DynFuncHolder< decltype(cloudx1_StereoToMono           ) > StereoToMono           ;
00267         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo           ) > MonoToStereo           ;
00268         helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo;
00269         helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo  ) > AudioMonoToLiveStereo  ;
00270         helper::DynFuncHolder< decltype(cloudx1_NewClient              ) > NewClient              ;
00271         helper::DynFuncHolder< decltype(cloudx1_DeleteClient           ) > DeleteClient           ;
00272         helper::DynFuncHolder< decltype(cloudx1_ProcessJson            ) > ProcessJson            ;
00273         helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer         ) > FreeJsonAnswer         ;
00274         helper::DynFuncHolder< decltype(cloudx1_SaveState              ) > SaveState              ;
00275 #if CLOUDX1_HAS_WEBSERVER
00276         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp          ) > Webserver_tcp          ;
00277         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2         ) > Webserver_tcp2         ;
00278         helper::DynFuncHolder< decltype(cloudx1_Webserver              ) > Webserver              ;
00279         helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop         ) > Webserver_Stop         ;
00280         helper::DynFuncHolder< decltype(cloudx1_Webserver_Status       ) > Webserver_Status       ;
00281 #endif // CLOUDX1_HAS_WEBSERVER
00282 
00283         // added 2023-03-22
00284         helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar    ) > StereoToMono_Planar    ;
00285         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar    ) > MonoToStereo_Planar    ;
00286 
00287         // added 2023-05-22
00288         helper::DynFuncHolder< decltype(cloudx1_SetMetadata            ) > SetMetadata            ;
00289         helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos       ) > GetMetadataInfos       ;
00290         helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos      ) > FreeMetadataInfos      ;
00291 
00292         // added 2023-06-20
00293         helper::DynFuncHolder< decltype(cloudx1_SetPresetManager               ) > SetPresetManager
      ;
00294         helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange     ) >
      PresetManager_InformChange     ;
00295 
00296         // added 2023-09-21
00297         helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames   ) >
      GetPossibleChunkSizeInFrames   ;
00298         helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames          ) >
      GetProcessChunkFrames   ;
00299         helper::DynFuncHolder< decltype(cloudx1_InitProcess3                   ) > InitProcess3
      ;
00300 
00301 #if CLOUDX1_HAS_CLOUDBUS
00302         // added 2023-06-26
00303         helper::DynFuncHolder< decltype(cloudx1_NewBus                 ) > NewBus      ;
00304         helper::DynFuncHolder< decltype(cloudx1_FreeBus                ) > FreeBus     ;
00305         helper::DynFuncHolder< decltype(cloudx1_SetInstanceBus         ) > SetInstanceBus      ;
00306 #endif // CLOUDX1_HAS_CLOUDBUS
00307 
```

```
00308         // added 2024-02-19
00309         helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti       ) > SetMetadataMulti         ;
00310
00311 #if CLOUDX1_HAS_WEBSERVER
00312         // added 2024-05-23
00313         helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth    ) > Webserver_SetAppHealth
    ;
00314         helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth    ) > Webserver_GetAppHealth
    ;
00315         helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString      ) > Webserver_FreeString
    ;
00316 #endif // CLOUDX1_HAS_WEBSERVER
00317
00318 #ifdef _WIN32
00319         // added 2025-05-28
00320         helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracing                ) >
    SetInstanceTracing                ;
00321         helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracingProcessActivity ) >
    SetInstanceTracingProcessActivity ;
00322 #endif // _WIN32
00323
00331         bool Load(const std::filesystem::path& path = {}) {
00332             std::string filename;
00333             #if defined(__APPLE__)
00334                 filename="libsound4.x1.cloud.dylib";
00335             #elif defined(__unix__)
00336                 filename="libsound4.x1.cloud.so";
00337             #else
00338                 filename="sound4.x1.cloud.dll";
00339             #endif
00340             m_lib.Close();
00341             // If a path is given, use it directly and do not try other path
00342             if (!path.empty() && !m_lib.Load(path / filename)) {
00343                 return false;
00344             } else if (!m_lib.IsOk()) {
00345                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00346                 #ifndef _WIN32
00347                 // Linux: if in a bin directory, try ../lib first
00348                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00349                     auto libdir = thisdir.parent_path() / "lib";
00350                     m_lib.Load(libdir / filename);
00351                 }
00352                 #endif // !_WIN32
00353                 if (!m_lib.IsOk()) {
00354                     // Search in the same directory this code is in
00355                     m_lib.Load(thisdir / filename);
00356                 }
00357                 if (!m_lib.IsOk()) {
00358                     // Try current path
00359                     std::error_code ec;
00360                     auto p = std::filesystem::current_path(ec);
00361                     if (!ec) {
00362                         m_lib.Load(p / filename);
00363                     }
00364                 }
00365                 if (!m_lib.IsOk()) {
00366                     return false;
00367                 }
00368             }
00369             // Load all C functions
00370             try {
00371                 GetVersion              = m_lib.GetSymbol< decltype(cloudx1_GetVersion             )
    >("cloudx1_GetVersion"             );
00372                 GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(cloudx1_GetChunkSizeInFrames   )
    >("cloudx1_GetChunkSizeInFrames"   );
00373                 GetChannelCount         = m_lib.GetSymbol< decltype(cloudx1_GetChannelCount        )
    >("cloudx1_GetChannelCount"        );
00374                 GetAudioInputCount      = m_lib.GetSymbol< decltype(cloudx1_GetAudioInputCount     )
    >("cloudx1_GetAudioInputCount"     );
00375                 GetAudioOutputCount     = m_lib.GetSymbol< decltype(cloudx1_GetAudioOutputCount    )
    >("cloudx1_GetAudioOutputCount"    );
00376                 GetSampleRate           = m_lib.GetSymbol< decltype(cloudx1_GetSampleRate          )
    >("cloudx1_GetSampleRate"          );
00377                 SetLoggerCallback       = m_lib.GetSymbol< decltype(cloudx1_SetLoggerCallback      )
    >("cloudx1_SetLoggerCallback"      );
00378                 SetLogSeverity          = m_lib.GetSymbol< decltype(cloudx1_SetLogSeverity         )
    >("cloudx1_SetLogSeverity"         );
00379                 NewParameters           = m_lib.GetSymbol< decltype(cloudx1_NewParameters          )
    >("cloudx1_NewParameters"          );
00380                 FreeParameters          = m_lib.GetSymbol< decltype(cloudx1_FreeParameters         )
    >("cloudx1_FreeParameters"         );
00381                 SetParameter            = m_lib.GetSymbol< decltype(cloudx1_SetParameter           )
    >("cloudx1_SetParameter"           );
00382                 GetParameter            = m_lib.GetSymbol< decltype(cloudx1_GetParameter           )
    >("cloudx1_GetParameter"           );
00383                 FreeParameterValue      = m_lib.GetSymbol< decltype(cloudx1_FreeParameterValue     )
    >("cloudx1_FreeParameterValue"     );
```

```
00384                 InitProcess             = m_lib.GetSymbol< decltype(cloudx1_InitProcess            )
      >("cloudx1_InitProcess"            );
00385                 InitProcess2            = m_lib.GetSymbol< decltype(cloudx1_InitProcess2           )
      >("cloudx1_InitProcess2"           );
00386                 TerminateProcess        = m_lib.GetSymbol< decltype(cloudx1_TerminateProcess       )
      >("cloudx1_TerminateProcess"       );
00387                 ExitProcess             = m_lib.GetSymbol< decltype(cloudx1_ExitProcess            )
      >("cloudx1_ExitProcess"            );
00388                 StartUpdateThread       = m_lib.GetSymbol< decltype(cloudx1_StartUpdateThread      )
      >("cloudx1_StartUpdateThread"      );
00389                 StopUpdateThread        = m_lib.GetSymbol< decltype(cloudx1_StopUpdateThread       )
      >("cloudx1_StopUpdateThread"       );
00390                 WaitUpdateThreadReady   = m_lib.GetSymbol< decltype(cloudx1_WaitUpdateThreadReady  )
      >("cloudx1_WaitUpdateThreadReady"  );
00391                 ProcessAudio            = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio           )
      >("cloudx1_ProcessAudio"           );
00392                 ProcessAudio_Planar     = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio_Planar    )
      >("cloudx1_ProcessAudio_Planar"    );
00393                 GetBufferIn             = m_lib.GetSymbol< decltype(cloudx1_GetBufferIn            )
      >("cloudx1_GetBufferIn"            );
00394                 GetBufferOut            = m_lib.GetSymbol< decltype(cloudx1_GetBufferOut           )
      >("cloudx1_GetBufferOut"           );
00395                 GetEstimatedDelay       = m_lib.GetSymbol< decltype(cloudx1_GetEstimatedDelay      )
      >("cloudx1_GetEstimatedDelay"      );
00396                 GetFormatName           = m_lib.GetSymbol< decltype(cloudx1_GetFormatName          )
      >("cloudx1_GetFormatName"          );
00397                 GetFormatFromName       = m_lib.GetSymbol< decltype(cloudx1_GetFormatFromName      )
      >("cloudx1_GetFormatFromName"      );
00398                 GetBytesFromFormat      = m_lib.GetSymbol< decltype(cloudx1_GetBytesFromFormat     )
      >("cloudx1_GetBytesFromFormat"     );
00399                 GetMaxPacketFrame       = m_lib.GetSymbol< decltype(cloudx1_GetMaxPacketFrame      )
      >("cloudx1_GetMaxPacketFrame"      );
00400                 AddAudio                = m_lib.GetSymbol< decltype(cloudx1_AddAudio               )
      >("cloudx1_AddAudio"               );
00401                 AddPadAudio             = m_lib.GetSymbol< decltype(cloudx1_AddPadAudio            )
      >("cloudx1_AddPadAudio"            );
00402                 GetOutputCount          = m_lib.GetSymbol< decltype(cloudx1_GetOutputCount         )
      >("cloudx1_GetOutputCount"         );
00403                 GetAudio                = m_lib.GetSymbol< decltype(cloudx1_GetAudio               )
      >("cloudx1_GetAudio"               );
00404                 AudioConvertFrom        = m_lib.GetSymbol< decltype(cloudx1_AudioConvertFrom       )
      >("cloudx1_AudioConvertFrom"       );
00405                 AudioConvertTo          = m_lib.GetSymbol< decltype(cloudx1_AudioConvertTo         )
      >("cloudx1_AudioConvertTo"         );
00406                 StereoToMono            = m_lib.GetSymbol< decltype(cloudx1_StereoToMono           )
      >("cloudx1_StereoToMono"           );
00407                 MonoToStereo            = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo           )
      >("cloudx1_MonoToStereo"           );
00408                 AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(cloudx1_AudioMonoFromLiveStereo)
      >("cloudx1_AudioMonoFromLiveStereo");
00409                 AudioMonoToLiveStereo   = m_lib.GetSymbol< decltype(cloudx1_AudioMonoToLiveStereo  )
      >("cloudx1_AudioMonoToLiveStereo"  );
00410                 NewClient               = m_lib.GetSymbol< decltype(cloudx1_NewClient              )
      >("cloudx1_NewClient"              );
00411                 DeleteClient            = m_lib.GetSymbol< decltype(cloudx1_DeleteClient           )
      >("cloudx1_DeleteClient"           );
00412                 ProcessJson             = m_lib.GetSymbol< decltype(cloudx1_ProcessJson            )
      >("cloudx1_ProcessJson"            );
00413                 FreeJsonAnswer          = m_lib.GetSymbol< decltype(cloudx1_FreeJsonAnswer         )
      >("cloudx1_FreeJsonAnswer"         );
00414                 SaveState               = m_lib.GetSymbol< decltype(cloudx1_SaveState              )
      >("cloudx1_SaveState"              );
00415 #if CLOUDX1_HAS_WEBSERVER
00416                 Webserver_tcp           = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp          )
      >("cloudx1_Webserver_tcp"          );
00417                 Webserver_tcp2          = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp2         )
      >("cloudx1_Webserver_tcp2"         );
00418                 Webserver               = m_lib.GetSymbol< decltype(cloudx1_Webserver              )
      >("cloudx1_Webserver"              );
00419                 Webserver_Stop          = m_lib.GetSymbol< decltype(cloudx1_Webserver_Stop         )
      >("cloudx1_Webserver_Stop"         );
00420                 Webserver_Status        = m_lib.GetSymbol< decltype(cloudx1_Webserver_Status       )
      >("cloudx1_Webserver_Status"       );
00421 #endif // CLOUDX1_HAS_WEBSERVER
00422
00423                 // added 2023-03-22
00424                 StereoToMono_Planar     = m_lib.GetSymbol< decltype(cloudx1_StereoToMono_Planar    )
      >("cloudx1_StereoToMono_Planar"    );
00425                 MonoToStereo_Planar     = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo_Planar    )
      >("cloudx1_MonoToStereo_Planar"    );
00426
00427             } catch (std::runtime_error& ) {
00428                 return false;
00429             }
00430             // C functions allowed to be missing
00431             try {
00432                 // added 2023-05-22
```

```
00433                 SetMetadata             = m_lib.GetSymbol< decltype(cloudx1_SetMetadata            )
      >("cloudx1_SetMetadata"           );
00434                 GetMetadataInfos        = m_lib.GetSymbol< decltype(cloudx1_GetMetadataInfos       )
      >("cloudx1_GetMetadataInfos"      );
00435                 FreeMetadataInfos       = m_lib.GetSymbol< decltype(cloudx1_FreeMetadataInfos      )
      >("cloudx1_FreeMetadataInfos"     );
00436                 // added 2023-06-20
00437                 SetPresetManager            = m_lib.GetSymbol< decltype(cloudx1_SetPresetManager
      ) >("cloudx1_SetPresetManager"            );
00438                 PresetManager_InformChange  = m_lib.GetSymbol<
      decltype(cloudx1_PresetManager_InformChange     ) >("cloudx1_PresetManager_InformChange"    );
00439
00440                 // added 2023-09-21
00441                 GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
      decltype(cloudx1_GetPossibleChunkSizeInFrames  ) >("cloudx1_GetPossibleChunkSizeInFrames"  );
00442                 GetProcessChunkFrames        = m_lib.GetSymbol< decltype(cloudx1_GetProcessChunkFrames
      ) >("cloudx1_GetProcessChunkFrames"       );
00443                 InitProcess3                 = m_lib.GetSymbol< decltype(cloudx1_InitProcess3
      ) >("cloudx1_InitProcess3"                );
00444
00445 #if CLOUDX1_HAS_CLOUDBUS
00446                 // added 2023-06-26
00447                 NewBus                  = m_lib.GetSymbol< decltype(cloudx1_NewBus                 )
      >("cloudx1_NewBus"                );
00448                 FreeBus                 = m_lib.GetSymbol< decltype(cloudx1_FreeBus                )
      >("cloudx1_FreeBus"               );
00449                 SetInstanceBus          = m_lib.GetSymbol< decltype(cloudx1_SetInstanceBus         )
      >("cloudx1_SetInstanceBus"        );
00450 #endif // CLOUDX1_HAS_CLOUDBUS
00451
00452                 // added 2024-02-19
00453                 SetMetadataMulti        = m_lib.GetSymbol< decltype(cloudx1_SetMetadataMulti       )
      >("cloudx1_SetMetadataMulti"      );
00454
00455 #if CLOUDX1_HAS_WEBSERVER
00456                 // added 2024-05-23
00457                 Webserver_SetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_SetAppHealth
      ) >("Webserver_SetAppHealth"     );
00458                 Webserver_GetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_GetAppHealth
      ) >("Webserver_GetAppHealth"     );
00459                 Webserver_FreeString        = m_lib.GetSymbol< decltype(cloudx1_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00460                 #endif // CLOUDX1_HAS_WEBSERVER
00461
00462 #ifdef _WIN32
00463                 // added 2025-05-28
00464                 SetInstanceTracing                = m_lib.GetSymbol<
      decltype(cloudx1_SetInstanceTracing               ) >("SetInstanceTracing"               );
00465                 SetInstanceTracingProcessActivity = m_lib.GetSymbol<
      decltype(cloudx1_SetInstanceTracingProcessActivity) >("SetInstanceTracingProcessActivity");
00466 #endif // _WIN32
00467             }  catch (std::runtime_error& ) {
00468                 // Ignored, handler will take care
00469             }
00470
00471             m_bOK=true;
00472             return true;
00473         }
00480         bool IsOk() const {
00481             return m_bOK;
00482         }
00483
00484     };
00485
00486
00487
00488     static constexpr const char *process_name = "SOUND4 x1.CLOUD";
00489     static constexpr const char *process_shortname = "x1";
00490     using SampleFormat = cloudx1_SampleFormat;
00495     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00500     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00505     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00510     const size_t InputSampleSize=ChunkFrames*InputChannels;
00515     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00520     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
00521
00525     namespace helper {
00526         #ifdef _WIN32
00532         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00533             if (wstr.empty()) return {};
00534             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00535             std::string str(size, 0);
00536             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00537             return str;
00538         }
00544         static inline std::wstring UTF8ToWString(const std::string& str) {
```

```
00545             if (str.empty()) return std::wstring();
00546             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00547             std::wstring wstr(size, 0);
00548             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00549             return wstr;
00550         }
00551         #endif // _WIN32
00552
00558             template<typename T>
00559         struct SampleFormat {
00560         };
00561             template<>
00562         struct SampleFormat<int16_t> {
00563             const cloudx1_SampleFormat format = S16_NATIVE;
00564         };
00565             template<>
00566         struct SampleFormat<int32_t> {
00567             const cloudx1_SampleFormat format = S32_NATIVE;
00568         };
00569             template<>
00570         struct SampleFormat<float> {
00571             const cloudx1_SampleFormat format = F32_NATIVE;
00572         };
00573
00574         // ------------------------------------------------------------
00586         static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00587             dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00588         }
00589
00601         static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00602             dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00603         }
00604
00615         static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00616             dynlib.StereoToMono(input, output, nFrame);
00617         }
00618
00629         static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00630             dynlib.MonoToStereo(input, output, nFrame);
00631         }
00632
00644         static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
      *inputR, float *output, size_t nFrame) {
00645             dynlib.StereoToMono_Planar(inputL, inputR, output, nFrame);
00646         }
00647
00659         static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
      float *outputR, size_t nFrame) {
00660             dynlib.MonoToStereo_Planar(input, outputL, outputR, nFrame);
00661         }
00662
00670         static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
      *output) {
00671             dynlib.AudioMonoFromLiveStereo(payload, output);
00672         }
00673
00681         static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
      *payload) {
00682             dynlib.AudioMonoToLiveStereo(input, payload);
00683         }
00684
00685     }; // namespace helper
00686
00693     static inline std::string GetVersion(CDynLib& dynlib)
00694         {
00695             return std::string(dynlib.GetVersion());
00696         }
00697
00709     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00710         {
00711             return dynlib.GetChunkSizeInFrames();
00712         }
00713
00723     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00724         {
00725             std::vector<unsigned int> list;
00726             if (dynlib.GetPossibleChunkSizeInFrames) {
00727                 for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00728                     list.push_back(*src);
00729                 }
00730             } else {
00731                 list.push_back(64);
00732             }
```

```
00733             return list;
00734         }
00735
00742     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00743         {
00744             return dynlib.GetChannelCount();
00745         }
00746
00755     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00756         {
00757             return dynlib.GetAudioInputCount();
00758         }
00759
00768     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00769         {
00770             return dynlib.GetAudioOutputCount();
00771         }
00772
00781     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00782         {
00783             return dynlib.GetSampleRate();
00784         }
00785
00793     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00794     {
00795         // Do some sanity checks
00796         if (dynlib.GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00797             throw std::runtime_error("Bad library sampling rate");
00798         }
00799         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
00800             throw std::runtime_error("Bad library frame size");
00801         }
00802         if (dynlib.GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00803             throw std::runtime_error("Bad library channel count");
00804         }
00805         if (dynlib.GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00806             throw std::runtime_error("Bad library input count");
00807         }
00808         if (dynlib.GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00809             throw std::runtime_error("Bad library output count");
00810         }
00811     }
00812
00820     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00821         {
00822             return std::string(dynlib.GetFormatName(fmt));
00823         }
00824
00832     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00833         {
00834             return dynlib.GetFormatFromName(name.c_str());
00835         }
00836
00844     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00845         {
00846             return dynlib.GetBytesFromFormat(fmt);
00847         }
00848
00852     using LogSeverity = cloudx1_LogSeverity;
00856     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00857
00861     static log_cb_t _log_cb;
00862     extern "C" {
00863         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00864             _log_cb(severity, std::string(c_msg));
00865         }
00866     }
00870     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00871         {
00872             dynlib.SetLogSeverity(severity);
00873         }
00877     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00878         {
00879             _log_cb=cb;
00880             dynlib.SetLoggerCallback(&_log_cb_c);
00881         }
00882
00883 #if CLOUDX1_HAS_CLOUDBUS
00889     class CBus {
00890         public:
00891             CBus(CDynLib& dynlib)
00892                 : m_dynlib(dynlib)
00893                 {
00894                     if (m_dynlib.NewBus) {
00895                         m_bus=m_dynlib.NewBus();
00896                     }
00897                 }
```

```
00898             ~CBus()
00899                 {
00900                     if (m_bus && m_dynlib.FreeBus) {
00901                         m_dynlib.FreeBus(m_bus);
00902                     }
00903                     m_bus = nullptr;
00904                 }
00905             cloudx1_CBus *Get() const { return m_bus; }
00906         private:
00907             CDynLib& m_dynlib;
00908             cloudx1_CBus *m_bus;
00909     };
00910 #endif // CLOUDX1_HAS_CLOUDBUS
00911
00917     class CPresetLoader {
00918     public:
00919         CPresetLoader() = default;
00920         virtual ~CPresetLoader() = default;
00921
00922         virtual bool IsReadOnly() = 0;
00923         virtual bool Exists(const std::filesystem::path &name) = 0;
00924         virtual bool Remove(const std::filesystem::path &name) = 0;
00925         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00926         virtual std::vector<std::filesystem::path> GetAll() = 0;
00927         virtual std::string Read(const std::filesystem::path &filename) = 0;
00928         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00929     };
00930     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00931
00937     class CInstance {
00938     public:
00942         CInstance(CDynLib& dynlib)
00943         : m_dynlib(dynlib)
00944             {
00945                 params = m_dynlib.NewParameters();
00946                 if (!params) throw std::bad_alloc();
00947             }
00948         // non-copyable
00949         CInstance( const CInstance& ) = delete;
00950         CInstance& operator=( const CInstance& ) = delete;
00954         virtual ~CInstance()
00955             {
00956                 Stop();
00957                 if (params) m_dynlib.FreeParameters(params);
00958             }
00962        bool IsOk() const { return instance!=nullptr; }
00972         void SetParam(const std::string& name, const std::string& value)
00973             {
00974                 assert(params);
00975                 m_dynlib.SetParameter(params, name.c_str(), value.c_str());
00976             }
00983         std::string GetParam(const std::string& name)
00984             {
00985                 assert(params);
00986                 auto c_value=m_dynlib.GetParameter(params, name.c_str());
00987                 std::string ret(c_value);
00988                 m_dynlib.FreeParameterValue(c_value);
00989                 return ret;
00990             }
00991 #if CLOUDX1_HAS_CLOUDBUS
01000             void SetBus(const CBus& bus) {
01001                 assert(params);
01002                 if (m_dynlib.SetInstanceBus) {
01003                     m_dynlib.SetInstanceBus(params,bus.Get());
01004                 }
01005             }
01006 #endif // CLOUDX1_HAS_CLOUDBUS
01007
01018         void SetPresetManager(CPresetLoader *preset_manager)
01019             {
01020                 if (!preset_manager) return;
01021                 if (!m_dynlib.SetPresetManager) return;
01022                 assert(params);
01023                 m_dynlib.SetPresetManager(params,
01024                     cloudx1_custom_reader,
01025                     cloudx1_custom_reader_free,
01026                     cloudx1_custom_writer,
01027                     cloudx1_custom_exists,
01028                     cloudx1_custom_getall,
01029                     cloudx1_custom_getall_free,
01030                     cloudx1_custom_remove,
01031                     cloudx1_custom_rename,
01032                     preset_manager->IsReadOnly(),
01033                     preset_manager
01034                 );
01035             }
01036
```

```
01059         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=64)
01060             {
01061                 assert(!instance);
01062                 std::string l_save_path;
01063                 if (!save_path.empty()) {
01064                     l_save_path = save_path.u8string();
01065                 }
01066                 if (m_dynlib.InitProcess3) {
01067                     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);
01068                 } else if (frames_per_chunk!=64) {
01069                     return false;
01070                 } else {
01071                     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);
01072                 }
01073                 if (!instance) return false;
01074                 if (!init_metadata.empty()) {
01075                     for (auto&& [key,value]: init_metadata) {
01076                         m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
01077                     }
01078                     init_metadata.clear();
01079                 }
01080                 update_thread = std::thread([this, json_port](){
01081                     OnUpdateThreadStart();
01082                     m_dynlib.StartUpdateThread(instance, json_port);
01083                     OnUpdateThreadStop();
01084                 });
01085                 int timeout=5000;
01086                 #ifdef DEBUG
01087                     timeout=60*1000; // For debugger pauses
01088                 #endif
01089                 if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
01090                     return false;
01091                 }
01092                 return true;
01093             }
01094
01102         unsigned int GetChunkFrames() {
01103             if (instance && m_dynlib.GetProcessChunkFrames) {
01104                 return m_dynlib.GetProcessChunkFrames(instance);
01105             }
01106             return 64;
01107         }
01108
01116         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01117             {
01118                 if (m_dynlib.PresetManager_InformChange) {
01119                     m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
      (cloudx1_PresetChange_Kind) change_kind);
01120                 }
01121             }
01122
01134         void SetMetadata(const std::string& key, const char* value)
01135             {
01136                 if (m_dynlib.SetMetadata) {
01137                     if (instance) {
01138                         m_dynlib.SetMetadata(instance, key.c_str(), value);
01139                     } else {
01140                         init_metadata.push_back( {key, value} );
01141                     }
01142                 }
01143             }
01144
01145
01155         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01156             {
01157                 if (instance) {
01158                     if (m_dynlib.SetMetadataMulti) {
01159                         typedef const char* pchar;
01160                         pchar* keyvalue=new pchar[2*list.size()+1];
01161                         size_t n=0;
01162                         for (auto&& [key,value]: list) {
01163                             keyvalue[2*n+0]=key.c_str();
01164                             keyvalue[2*n+1]=value;
01165                             n++;
01166                         }
01167                         keyvalue[2*n+0]=nullptr;
01168
01169                         cloudx1_SetMetadataMulti(instance, keyvalue);
01170
01171                         delete[] keyvalue;
01172                     } else {
01173                         for (auto&& [key,value]: list) {
```

```
01174                             SetMetadata(key, value);
01175                         }
01176                     }
01177                 } else {
01178                     for (auto&& [key,value]: list) {
01179                         if (value) {
01180                             init_metadata.push_back( {key, value} );
01181                         }
01182                     }
01183                 }
01184             }
01185
01191         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01192             {
01193                 std::vector< std::tuple<std::string,std::string> > values;
01194                 if (m_dynlib.GetMetadataInfos) {
01195                     const char** c_values = m_dynlib.GetMetadataInfos(instance);
01196                     if (c_values) {
01197                         for (const char** c_value=c_values; *c_value; ) {
01198                             std::string key(*c_value);
01199                             c_value++;
01200                             if (*c_value) {
01201                                 std::string descr(*c_value);
01202                                 c_value++;
01203                                 values.push_back( {key,descr} );
01204                             }
01205                         }
01206                         m_dynlib.FreeMetadataInfos(instance, c_values);
01207                     }
01208                 }
01209                 return values;
01210             }
01211
01215         virtual void OnUpdateThreadStart() {}
01219         virtual void OnUpdateThreadStop() {}
01220
01221 #if CLOUDX1_HAS_WEBSERVER
01235         bool StartWebServer(int http_port, int https_port=0)
01236             {
01237                 assert(instance);
01238                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01239                 webserver = m_dynlib.Webserver(http_port,https_port,instance);
01240                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01241                     return false;
01242                 }
01243                 int web_status=m_dynlib.Webserver_Status(webserver);
01244                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01245                     return false;
01246                 }
01247                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01248                     return false;
01249                 }
01250                 return true;
01251             }
01252
01260         void StopWebServer(int timeout_ms = 1000)
01261             {
01262                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01263                     m_dynlib.Webserver_Stop(webserver, timeout_ms);
01264                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01265                 }
01266             }
01267
01271         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
01272             {
01273                 assert(instance);
01274                 if (!m_dynlib.Webserver_SetAppHealth) return;
01275                 m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
01276             }
01280         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01281             {
01282                 assert(instance);
01283                 if (!m_dynlib.Webserver_GetAppHealth) return;
01284                 char* c_contenttype=nullptr;
01285                 char* c_content=nullptr;
01286                 m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01287                 contenttype=c_contenttype;
01288                 content=c_content;
01289                 m_dynlib.Webserver_FreeString(c_contenttype);
01290                 m_dynlib.Webserver_FreeString(c_content);
01291             }
01292 #endif // CLOUDX1_HAS_WEBSERVER
01293
01302         int Terminate()
01303             {
```

```
01304                 if (!instance) return 0;
01305                 return m_dynlib.TerminateProcess(instance);
01306             }
01307 
01311         void Stop()
01312             {
01313                 if (!instance) return;
01314 #if CLOUDX1_HAS_WEBSERVER
01315                 StopWebServer();
01316 #endif // CLOUDX1_HAS_WEBSERVER
01317                 if (update_thread.joinable()) {
01318                     m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01319                     update_thread.join();
01320                 }
01321                 m_dynlib.ExitProcess(instance);
01322                 instance=nullptr;
01323             }
01324 
01334         unsigned int GetEstimatedDelay()
01335             {
01336                 assert(instance);
01337                 return m_dynlib.GetEstimatedDelay(instance);
01338             }
01339 
01352         std::array<float,InputSampleSize>& GetBufferIn()
01353             {
01354                 assert(instance);
01355                 float *buf=m_dynlib.GetBufferIn(instance);
01356                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
01357 
01358             }
01370         std::array<float,OutputSampleSize>& GetBufferOut()
01371             {
01372                 assert(instance);
01373                 float *buf= m_dynlib.GetBufferOut(instance);
01374                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
01375             }
01379         void ProcessAudio()
01380             {
01381                 assert(instance);
01382                 m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
      m_dynlib.GetBufferOut(instance));
01383             }
01395         void ProcessAudio_Planar(float const * const *input, float * const *output)
01396             {
01397                 assert(instance);
01398                 m_dynlib.ProcessAudio_Planar(instance, input, output);
01399             }
01400 
01412             template<typename T>
01413         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01414             {
01415                 assert(instance);
01416                 std::vector<T> output;
01417                 unsigned int out_offset=0;
01418                 unsigned int in_offset=0;
01419                 unsigned int todo = input.size();
01420                 while (todo>0) {
01421                     unsigned int left = AddAudio(&input[in_offset], todo);
01422                     unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01423                     output.resize(out_offset + out_avail);
01424                     GetAudio(&output[out_offset], out_avail);
01425                     out_offset+=out_avail;
01426                     in_offset += todo-left;
01427                     todo=left;
01428                 }
01429                 return output;
01430             }
01431 
01437         class CClient {
01438             public:
01439                 CClient(cloudx1_CInstance* instance, CDynLib& dynlib)
01440                 : m_dynlib(dynlib)
01441                     {
01442                         assert(instance);
01443                         client=m_dynlib.NewClient(instance);
01444                         if (!client) throw std::bad_alloc();
01445                     }
01446                 // non-copyable
01447                 CClient( const CInstance& ) = delete;
01448                 CClient& operator=( const CInstance& ) = delete;
01449                 ~CClient()
01450                     {
01451                         if (client) {
01452                             m_dynlib.DeleteClient(client);
01453                             client=nullptr;
01454                         }
```

```
01455                     }
01463                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01464                     {
01465                         assert(client);
01466                         int need_save=0;
01467                         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
      &need_save);
01468                         if (!canswer) return {};
01469                         std::string answer(canswer);
01470                         m_dynlib.FreeJsonAnswer (canswer);
01471                         if (NeedSave) {
01472                             *NeedSave=(need_save!=0);
01473                         }
01474                         return answer;
01475                     }
01476             private:
01477                 CDynLib& m_dynlib;
01478                 cloudx1_CClientInstance *client = nullptr;
01479         };
01487         std::shared_ptr<CClient> NewClient()
01488             {
01489                 assert(instance);
01490                 return std::make_shared<CClient>(instance, m_dynlib);
01491             }
01492
01503         bool SaveState() {
01504             assert(instance);
01505             if (m_dynlib.SaveState(instance)!=0) {
01506                 return false;
01507             }
01508             return true;
01509         }
01510 #ifdef _WIN32
01517         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01518             if (m_dynlib.SetInstanceTracing) {
01519                 m_dynlib.SetInstanceTracing(params, tracing_provider, activity_guid);
01520             }
01521         }
01528         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01529             assert(instance);
01530             if (m_dynlib.SetInstanceTracingProcessActivity) {
01531                 m_dynlib.SetInstanceTracingProcessActivity(instance, activity_guid);
01532             }
01533         }
01534 #endif // _WIN32
01535     protected:
01536             template<typename T>
01537         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01538             {
01539                 assert(instance);
01540                 return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01541             }
01542             template<typename T>
01543         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01544             {
01545                 assert(instance);
01546                 return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01547             }
01548
01549     private:
01550         CDynLib& m_dynlib;
01551         cloudx1_CParameters* params = nullptr;
01552         cloudx1_CInstance* instance = nullptr;
01553         std::thread update_thread;
01554         std::vector< std::pair<std::string,std::string> > init_metadata;
01555 #if CLOUDX1_HAS_WEBSERVER
01556         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01557 #endif // CLOUDX1_HAS_WEBSERVER
01558     };
01559
01560 }; // namespace dyn
01561 }; // namespace cloudx1
01562 }; // namespace sound4
01563
01564 // bridge C callbacks to CPresetLoader
01565 extern "C" {
01566     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01567         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01568         std::string content=preset_loader->Read(filename);
01569         return strdup(content.c_str());
01570     }
01571     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01572         if (!content) return;
01573         free(content);
```

```
01574     }
01575     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01576         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01577         auto res=preset_loader->Write(filename,content);
01578         return res?0:-1;
01579     }
01580     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01581         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01582         auto res=preset_loader->Exists(filename);
01583         return res?0:-1;
01584     }
01585     static fs_char** cloudx1_custom_getall(void* userdata) {
01586         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01587         auto res=preset_loader->GetAll();
01588         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01589         for (size_t n=0;n<res.size();n++) {
01590             all[n]=fs_strdup(res[n].c_str());
01591         }
01592         all[res.size()]=nullptr;
01593         return all;
01594     }
01595     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01596         if (!all) return;
01597         for (fs_char** one=all;*one;one++) {
01598             free(*one);
01599         }
01600         free(all);
01601     }
01602     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01603         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01604         auto res=preset_loader->Remove(filename);
01605         return res?0:-1;
01606     }
01607     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01608         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01609         auto res=preset_loader->Rename(from,to);
01610         return res?0:-1;
01611     }
01612
01613 };
```

15.25 sound4cl.hpp File Reference

C++ wrapper for dynamic loading library.

```
#include <string>
#include <exception>
#include <thread>
#include <functional>
#include <memory>
#include <array>
#include <assert.h>
#include <stdexcept>
#include <filesystem>
#include <string.h>
#include <libloaderapi.h>
#include "sound4cl_cdef.h"
```

Classes

- struct sound4::helper::SampleFormat< T >

  Helper for Sample format types.
- class sound4::helper::DynFuncHolder< T >

Helper to wrap a function pointer.

- class sound4::helper::CDynLoader

  Helper to load dynamic library.
- class sound4::CProcessor

  Dynamic library interface.
- class sound4::CBus

  Shared Bus.
- class sound4::CPresetLoader

  Custom preset handler helper.
- class sound4::CInstance

  Instance handling class.
- class sound4::CInstance::CClient

Namespaces

- namespace sound4
- namespace sound4::helper

Macros

- #define UNIXLIKE

Typedefs

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

Enumerations

- enum sound4::SampleFormat {
  sound4::INVALID_FORMAT , sound4::S16_LE , sound4::S16_BE , sound4::S24_LE ,
  sound4::S24_BE , sound4::S32_LE , sound4::S32_BE , sound4::F32_LE ,
  sound4::F32_BE , sound4::S16_NATIVE , sound4::S24_NATIVE , sound4::S32_NATIVE ,
  sound4::F32_NATIVE }
- enum sound4::LogSeverity {
  sound4::none , sound4::fatal , sound4::error , sound4::warning ,
  sound4::info , sound4::verbose , sound4::verbose2 , sound4::verbose3 ,
  sound4::verbose4 , sound4::verbose5 }
- enum sound4::PresetChange_Kind { sound4::change_kind_created , sound4::change_kind_modified ,
  sound4::change_kind_deleted }

Functions

- static std::string sound4::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::helper::UTF8ToWString (const std::string &str)
- static void sound4::_log_cb_c (sound4cl_LogSeverity severity, const char *c_msg)

Variables

- static log_cb_t sound4::_log_cb

15.25.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file sound4cl.hpp.

15.25.2 Macro Definition Documentation

15.25.2.1 UNIXLIKE

```
#define UNIXLIKE
```

Definition at line 24 of file sound4cl.hpp.

15.26 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                       std::filesystem::path p(rp);
00209                       free(rp);
00210                       return p;
00211                   }
00212               #else
00213                   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_
00214                   {
00215                       Close();
00216                       m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00217                       return IsOk();
00218                   }
00219                   void Close() {
00220                       if (IsOk()) {
00221                           // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00222                           // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00223                           #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00224                               FreeLibraryAndExitThread(m_lib, 0);
00225                           #elif defined(SOUND4_CALL_FREELIBRARY)
00226                               FreeLibrary(m_lib);
00227                           #endif // SOUND4_CALL_FREELIBRARY
00228                           m_lib=NULL;
00229                       }
00230                   }
00231                       template <typename T>
00232                   DynFuncHolder<T> GetSymbol(const std::string& name) {
00233                       auto ptr=GetProcAddress(m_lib, name.c_str());
00234                       if (!ptr) {
00235                           throw std::runtime_error("Missing function in library");
00236                       }
00237                       return DynFuncHolder<T>(ptr);
00238                   }
00239                       template <typename T>
00240                   DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00241                       auto ptr=GetProcAddress(m_lib, name.c_str());
00242                       return DynFuncHolder<T>(ptr);
00243                   }
00244                   static std::filesystem::path GetThisLibraryPath(void) {
00245                       static wchar_t path[MAX_PATH]={};
00246                       HMODULE hm = NULL;
00247                       if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00248                               GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00249                               (LPCWSTR) &GetThisLibraryPath, &hm))
00250                       {
00251                           if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00252                               return std::filesystem::path(path);
00253                           }
00254                       }
00255                       return {};
00256                   }
00257               #endif
00258               virtual ~CDynLoader() {
00259                   Close();
00260               }
00261       }; // class CDynLoader
00262   }; // namespace helper
```

```
00263
00264
00268     enum LogSeverity {
00269        none     = sound4cl_none,
00270        fatal    = sound4cl_fatal,
00271        error    = sound4cl_error,
00272        warning  = sound4cl_warning,
00273        info     = sound4cl_info,
00274        verbose  = sound4cl_verbose,
00275        verbose2 = sound4cl_verbose2,
00276        verbose3 = sound4cl_verbose3,
00277        verbose4 = sound4cl_verbose4,
00278        verbose5 = sound4cl_verbose5,
00279     };
00280
00284     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00285
00289     static log_cb_t _log_cb;
00290     extern "C" {
00291         static inline void _log_cb_c(sound4cl_LogSeverity severity, const char *c_msg) {
00292             _log_cb((LogSeverity)severity, std::string(c_msg));
00293         }
00294     }
00295
00302     class CProcessor {
00303     private:
00304         helper::CDynLoader m_lib;
00305         bool m_bOK = false;
00306         std::string m_prefix; // The prefix of functions, depends on the library (cloud_, impact_...)
00307     protected:
00308         friend class CBus;
00309         friend class CInstance;
00310
00311         // All those wrapped functions have the same signature as there C sound4cl_XXX
00312         helper::DynFuncHolder< decltype(sound4cl_GetVersion              ) > fnGetVersion              ;
00313         helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames    ) > fnGetChunkSizeInFrames    ;
00314         helper::DynFuncHolder< decltype(sound4cl_GetChannelCount         ) > fnGetChannelCount         ;
00315         helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount      ) > fnGetAudioInputCount      ;
00316         helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount     ) > fnGetAudioOutputCount     ;
00317         helper::DynFuncHolder< decltype(sound4cl_GetSampleRate           ) > fnGetSampleRate           ;
00318         helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback       ) > fnSetLoggerCallback       ;
00319         helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity          ) > fnSetLogSeverity          ;
00320         helper::DynFuncHolder< decltype(sound4cl_NewParameters           ) > fnNewParameters           ;
00321         helper::DynFuncHolder< decltype(sound4cl_FreeParameters          ) > fnFreeParameters          ;
00322         helper::DynFuncHolder< decltype(sound4cl_SetParameter            ) > fnSetParameter            ;
00323         helper::DynFuncHolder< decltype(sound4cl_GetParameter            ) > fnGetParameter            ;
00324         helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue      ) > fnFreeParameterValue      ;
00325         helper::DynFuncHolder< decltype(sound4cl_InitProcess             ) > fnInitProcess             ;
00326         helper::DynFuncHolder< decltype(sound4cl_InitProcess2            ) > fnInitProcess2            ;
00327         helper::DynFuncHolder< decltype(sound4cl_TerminateProcess        ) > fnTerminateProcess        ;
00328         helper::DynFuncHolder< decltype(sound4cl_ExitProcess             ) > fnExitProcess             ;
00329         helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread       ) > fnStartUpdateThread       ;
00330         helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread        ) > fnStopUpdateThread        ;
00331         helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady   ) > fnWaitUpdateThreadReady   ;
00332         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio            ) > fnProcessAudio            ;
00333         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar     ) > fnProcessAudio_Planar     ;
00334         helper::DynFuncHolder< decltype(sound4cl_GetBufferIn             ) > fnGetBufferIn             ;
00335         helper::DynFuncHolder< decltype(sound4cl_GetBufferOut            ) > fnGetBufferOut            ;
00336         helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay       ) > fnGetEstimatedDelay       ;
00337         helper::DynFuncHolder< decltype(sound4cl_GetFormatName           ) > fnGetFormatName           ;
00338         helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName       ) > fnGetFormatFromName       ;
00339         helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat      ) > fnGetBytesFromFormat      ;
00340         helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame       ) > fnGetMaxPacketFrame       ;
00341         helper::DynFuncHolder< decltype(sound4cl_AddAudio                ) > fnAddAudio                ;
00342         helper::DynFuncHolder< decltype(sound4cl_AddPadAudio             ) > fnAddPadAudio             ;
00343         helper::DynFuncHolder< decltype(sound4cl_GetOutputCount          ) > fnGetOutputCount          ;
00344         helper::DynFuncHolder< decltype(sound4cl_GetAudio                ) > fnGetAudio                ;
00345         helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom        ) > fnAudioConvertFrom        ;
00346         helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo          ) > fnAudioConvertTo          ;
00347         helper::DynFuncHolder< decltype(sound4cl_StereoToMono            ) > fnStereoToMono            ;
00348         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo            ) > fnMonoToStereo            ;
00349         helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
00350         helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo   ) > fnAudioMonoToLiveStereo   ;
00351         helper::DynFuncHolder< decltype(sound4cl_NewClient               ) > fnNewClient               ;
00352         helper::DynFuncHolder< decltype(sound4cl_DeleteClient            ) > fnDeleteClient            ;
00353         helper::DynFuncHolder< decltype(sound4cl_ProcessJson             ) > fnProcessJson             ;
00354         helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer          ) > fnFreeJsonAnswer          ;
00355         helper::DynFuncHolder< decltype(sound4cl_SaveState               ) > fnSaveState               ;
00356
00357         // only for processes supporting webserver
00358         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp           ) > fnWebserver_tcp           ;
00359         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2          ) > fnWebserver_tcp2          ;
00360         helper::DynFuncHolder< decltype(sound4cl_Webserver               ) > fnWebserver               ;
00361         helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop          ) > fnWebserver_Stop          ;
00362         helper::DynFuncHolder< decltype(sound4cl_Webserver_Status        ) > fnWebserver_Status        ;
00363
00364         // added 2023-03-22
```

```
00365         helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar    ) > fnStereoToMono_Planar    ;
00366         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar    ) > fnMonoToStereo_Planar    ;
00367 
00368         // added 2023-05-22
00369         helper::DynFuncHolder< decltype(sound4cl_SetMetadata            ) > fnSetMetadata            ;
00370         helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos       ) > fnGetMetadataInfos       ;
00371         helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos      ) > fnFreeMetadataInfos      ;
00372 
00373         // added 2023-06-20
00374         helper::DynFuncHolder< decltype(sound4cl_SetPresetManager               ) >
      fnSetPresetManager      ;
00375         helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange     ) >
      fnPresetManager_InformChange      ;
00376 
00377         // added 2023-09-21
00378         helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) >
      fnGetPossibleChunkSizeInFrames   ;
00379         helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) >
      fnGetProcessChunkFrames   ;
00380         helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3
      ;
00381 
00382         // added 2023-06-26, only for processes supporting bus
00383         helper::DynFuncHolder< decltype(sound4cl_NewBus                  ) > fnNewBus      ;
00384         helper::DynFuncHolder< decltype(sound4cl_FreeBus                 ) > fnFreeBus      ;
00385         helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus          ) > fnSetInstanceBus      ;
00386 
00387         // added 2024-02-19
00388         helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti        ) > fnSetMetadataMulti
      ;
00389 
00390         // added 2024-05-23
00391         helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) >
      fnWebserver_SetAppHealth      ;
00392         helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) >
      fnWebserver_GetAppHealth      ;
00393         helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString
      ;
00394 
00395 #ifdef _WIN32
00396         // added 2025-05-28
00397         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracing                ) >
      fnSetInstanceTracing                ;
00398         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracingProcessActivity ) >
      fnSetInstanceTracingProcessActivity ;
00399 #endif // _WIN32
00400 
00401             template <typename T>
00402         helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
00403             std::string prefix_name = m_prefix+name;
00404             return m_lib.GetSymbol<T>(prefix_name);
00405         }
00406 
00407     public:
00408         CProcessor() = default;
00409 
00417         bool Load(const std::filesystem::path& filepath) {
00418             m_lib.Close();
00419             // If a path is given, use it directly and do not try other path
00420             if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
00421                 return false;
00422             } else if (!m_lib.IsOk()) {
00423                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00424                 auto filename = filepath.filename();
00425                 #ifndef _WIN32
00426                 // Linux: if in a bin directory, try ../lib first
00427                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00428                     auto libdir = thisdir.parent_path() / "lib";
00429                     m_lib.Load(libdir / filename);
00430                 }
00431                 #endif // !_WIN32
00432                 if (!m_lib.IsOk()) {
00433                     // Search in the same directory this code is in
00434                     m_lib.Load(thisdir / filename);
00435                 }
00436                 if (!m_lib.IsOk()) {
00437                     // Try current path
00438                     std::error_code ec;
00439                     auto p = std::filesystem::current_path(ec);
00440                     if (!ec) {
00441                         m_lib.Load(p / filename);
00442                     }
00443                 }
00444                 if (!m_lib.IsOk()) {
00445                     return false;
00446                 }
00447             }
```

```
00448             // Get information from the library itself
00449             try {
00450                 auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             )
      >("SOUND4_GetProcessInfo"   );
00451                 auto info = GetInfo();
00452                 m_prefix = info->prefix;
00453             } catch (std::runtime_error& ) {
00454                 // Compatibility: search in known libraries without this function
00455                 #ifdef _WIN32
00456                     std::string filename = helper::WStringToUTF8(filepath.stem());
00457                 #else // !_WIN32
00458                     std::string filename = filepath.stem();
00459                     // remove lib prefix
00460                     filename=filename.substr(3);
00461                 #endif // ! _WIN32
00462                 if (filename=="sound4.x1.cloud") {
00463                     m_prefix="cloudx1_";
00464                 } else if (filename=="sound4.impact.cl") {
00465                     m_prefix="impact_";
00466                 } else if (filename=="sound4.bigvoice.cl") {
00467                     m_prefix="bigvoice_";
00468                 } else {
00469                     return false;
00470                 }
00471             }
00472
00473             // Load all C functions
00474             try {
00475                 fnGetVersion             = GetPrefixSymbol< decltype(sound4cl_GetVersion
      ) >("GetVersion"             );
00476                 fnGetChunkSizeInFrames   = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
      ) >("GetChunkSizeInFrames"   );
00477                 fnGetChannelCount        = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
      ) >("GetChannelCount"        );
00478                 fnGetAudioInputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
      ) >("GetAudioInputCount"     );
00479                 fnGetAudioOutputCount    = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
      ) >("GetAudioOutputCount"    );
00480                 fnGetSampleRate          = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
      ) >("GetSampleRate"          );
00481                 fnSetLoggerCallback      = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
      ) >("SetLoggerCallback"      );
00482                 fnSetLogSeverity         = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
      ) >("SetLogSeverity"         );
00483                 fnNewParameters          = GetPrefixSymbol< decltype(sound4cl_NewParameters
      ) >("NewParameters"          );
00484                 fnFreeParameters         = GetPrefixSymbol< decltype(sound4cl_FreeParameters
      ) >("FreeParameters"         );
00485                 fnSetParameter           = GetPrefixSymbol< decltype(sound4cl_SetParameter
      ) >("SetParameter"           );
00486                 fnGetParameter           = GetPrefixSymbol< decltype(sound4cl_GetParameter
      ) >("GetParameter"           );
00487                 fnFreeParameterValue     = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
      ) >("FreeParameterValue"     );
00488                 fnInitProcess            = GetPrefixSymbol< decltype(sound4cl_InitProcess
      ) >("InitProcess"            );
00489                 fnInitProcess2           = GetPrefixSymbol< decltype(sound4cl_InitProcess2
      ) >("InitProcess2"           );
00490                 fnTerminateProcess       = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
      ) >("TerminateProcess"       );
00491                 fnExitProcess            = GetPrefixSymbol< decltype(sound4cl_ExitProcess
      ) >("ExitProcess"            );
00492                 fnStartUpdateThread      = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
      ) >("StartUpdateThread"      );
00493                 fnStopUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
      ) >("StopUpdateThread"       );
00494                 fnWaitUpdateThreadReady  = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
      ) >("WaitUpdateThreadReady"  );
00495                 fnProcessAudio           = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
      ) >("ProcessAudio"           );
00496                 fnProcessAudio_Planar    = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
      ) >("ProcessAudio_Planar"    );
00497                 fnGetBufferIn            = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
      ) >("GetBufferIn"            );
00498                 fnGetBufferOut           = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
      ) >("GetBufferOut"           );
00499                 fnGetEstimatedDelay      = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
      ) >("GetEstimatedDelay"      );
00500                 fnGetFormatName          = GetPrefixSymbol< decltype(sound4cl_GetFormatName
      ) >("GetFormatName"          );
00501                 fnGetFormatFromName      = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
      ) >("GetFormatFromName"      );
00502                 fnGetBytesFromFormat     = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
      ) >("GetBytesFromFormat"     );
00503                 fnGetMaxPacketFrame      = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
      ) >("GetMaxPacketFrame"      );
00504                 fnAddAudio               = GetPrefixSymbol< decltype(sound4cl_AddAudio
```

```
    ) >("AddAudio"              );
00505                   fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
    ) >("AddPadAudio"           );
00506                   fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
    ) >("GetOutputCount"        );
00507                   fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
    ) >("GetAudio"              );
00508                   fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
    ) >("AudioConvertFrom"      );
00509                   fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
    ) >("AudioConvertTo"        );
00510                   fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
    ) >("StereoToMono"          );
00511                   fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo
    ) >("MonoToStereo"          );
00512                   fnAudioMonoFromLiveStereo = GetPrefixSymbol<
    decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
00513                   fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo
    ) >("AudioMonoToLiveStereo" );
00514                   fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient
    ) >("NewClient"             );
00515                   fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient
    ) >("DeleteClient"          );
00516                   fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson
    ) >("ProcessJson"           );
00517                   fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer
    ) >("FreeJsonAnswer"        );
00518                   fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState
    ) >("SaveState"             );
00519               } catch (std::runtime_error& ) {
00520                   return false;
00521               }
00522               // Webserver : not for all projects
00523               try {
00524                   fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp
    ) >("Webserver_tcp"         );
00525                   fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2
    ) >("Webserver_tcp2"        );
00526                   fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver
    ) >("Webserver"             );
00527                   fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop
    ) >("Webserver_Stop"        );
00528                   fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status
    ) >("Webserver_Status"      );
00529               } catch (std::runtime_error& ) {
00530                   // Ignored, handler will take care
00531               }
00532               // Stereo/Mono conversion : must be present
00533               try {
00534                   // added 2023-03-22
00535                   fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar
    ) >("StereoToMono_Planar"   );
00536                   fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar
    ) >("MonoToStereo_Planar"   );
00537
00538               } catch (std::runtime_error& ) {
00539                   return false;
00540               }
00541               // C functions allowed to be missing
00542               try {
00543                   // added 2023-05-22
00544                   fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata
    ) >("SetMetadata"           );
00545                   fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos
    ) >("GetMetadataInfos"      );
00546                   fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos
    ) >("FreeMetadataInfos"     );
00547                   // added 2023-06-20
00548                   fnSetPresetManager          = GetPrefixSymbol< decltype(sound4cl_SetPresetManager
    ) >("SetPresetManager"          );
00549                   fnPresetManager_InformChange  = GetPrefixSymbol<
    decltype(sound4cl_PresetManager_InformChange    ) >("PresetManager_InformChange"   );
00550
00551                   // added 2023-09-21
00552                   fnGetPossibleChunkSizeInFrames = GetPrefixSymbol<
    decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
00553                   fnGetProcessChunkFrames       = GetPrefixSymbol<
    decltype(sound4cl_GetProcessChunkFrames        ) >("GetProcessChunkFrames"        );
00554                   fnInitProcess3                = GetPrefixSymbol< decltype(sound4cl_InitProcess3
    ) >("InitProcess3"              );
00555               }  catch (std::runtime_error& ) {
00556                   // Ignored, handler will take care
00557               }
00558
00559               // bus: not for all projects
00560               try {
00561                   // added 2023-06-26
```

```
00562                 fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus
      ) >("NewBus"                );
00563                 fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus
      ) >("FreeBus"               );
00564                 fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus
      ) >("SetInstanceBus"        );
00565             }  catch (std::runtime_error& ) {
00566                 // Ignored, handler will take care
00567             }
00568
00569             // Other optional
00570             try {
00571                 // added 2024-02-19
00572                 fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti
      ) >("SetMetadataMulti"      );
00573
00574             }  catch (std::runtime_error& ) {
00575                 // Ignored, handler will take care
00576             }
00577
00578             // Other optional, web server only
00579             try {
00580                 // added 2024-05-23
00581                 fnWebserver_SetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
00582                 fnWebserver_GetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
00583                 fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString
      ) >("Webserver_FreeString"      );
00584             }  catch (std::runtime_error& ) {
00585                 // Ignored, handler will take care
00586             }
00587
00588 #ifdef _WIN32
00589             // Other optional, windows tracing
00590             try {
00591                 // added 2025-05-28
00592                 fnSetInstanceTracing               = GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracing                    ) >("SetInstanceTracing"               );
00593                 fnSetInstanceTracingProcessActivity= GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracingProcessActivity    ) >("SetInstanceTracingProcessActivity");
00594             }  catch (std::runtime_error& ) {
00595                 // Ignored, handler will take care
00596             }
00597 #endif // _WIN32
00598
00599             m_bOK=true;
00600             return true;
00601         }
00608         bool IsOk() const {
00609             return m_bOK;
00610         }
00611
00612         // ------------------------------------------------------------
00619         // ------------------------------------------------------------
00630         void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
00631             { fnAudioConvertFrom(payload, output, nSpl, (sound4cl_SampleFormat)fmt); }
00632
00643         void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
00644             { fnAudioConvertTo(input, payload, nSpl, (sound4cl_SampleFormat)fmt); }
00645
00655         void StereoToMono(const float *input, float *output, size_t nFrame)
00656             { fnStereoToMono(input, output, nFrame); }
00657
00667         void MonoToStereo(const float *input, float *output, size_t nFrame)
00668             { fnMonoToStereo(input, output, nFrame); }
00669
00680         void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame)
00681             { fnStereoToMono_Planar(inputL, inputR, output, nFrame); }
00682
00693         void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame)
00694             { fnMonoToStereo_Planar(input, outputL, outputR, nFrame); }
00695
00702         void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
00703             { fnAudioMonoFromLiveStereo(payload, output); }
00704
00711         void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
00712             { fnAudioMonoToLiveStereo(input, payload); }
00713
00721         std::string GetVersion()
00722             { return std::string(fnGetVersion()); }
00723
00734         unsigned int GetChunkSizeInFrames()
00735             { return fnGetChunkSizeInFrames(); }
00736
00745         std::vector<unsigned int> GetPossibleChunkSizeInFrames()
```

```
00746             {
00747                 std::vector<unsigned int> list;
00748                 if (fnGetPossibleChunkSizeInFrames) {
00749                     for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
00750                         list.push_back(*src);
00751                     }
00752                 } else {
00753                     list.push_back(fnGetChunkSizeInFrames());
00754                 }
00755                 return list;
00756             }
00757
00763         unsigned int GetChannelCount()
00764             { return fnGetChannelCount(); }
00765
00773         unsigned int GetAudioInputCount()
00774             { return fnGetAudioInputCount(); }
00775
00783         unsigned int GetAudioOutputCount()
00784             { return fnGetAudioOutputCount(); }
00785
00793         unsigned int GetSampleRate()
00794             { return fnGetSampleRate(); }
00795
00802         std::string GetFormatName( const SampleFormat fmt)
00803             { return std::string(fnGetFormatName((sound4cl_SampleFormat)fmt)); }
00804
00811         SampleFormat GetFormatFromName( const std::string& name)
00812             { return SampleFormat(fnGetFormatFromName(name.c_str())); }
00813
00820         unsigned int GetBytesFromFormat( const SampleFormat fmt)
00821             { return fnGetBytesFromFormat((sound4cl_SampleFormat)fmt); }
00822
00826         void SetLogSeverity(LogSeverity severity)
00827             { fnSetLogSeverity((sound4cl_LogSeverity)severity); }
00828
00832         void SetLoggerCallback(log_cb_t cb)
00833             { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
00834     };
00835
00841     class CBus {
00842         public:
00843             CBus(CProcessor& dynlib)
00844                 : m_dynlib(dynlib)
00845                 {
00846                     if (m_dynlib.fnNewBus) {
00847                         m_bus=m_dynlib.fnNewBus();
00848                     }
00849                 }
00850             ~CBus()
00851                 {
00852                     if (m_bus && m_dynlib.fnFreeBus) {
00853                         m_dynlib.fnFreeBus(m_bus);
00854                     }
00855                     m_bus = nullptr;
00856                 }
00857         protected:
00858             friend class CInstance;
00859             sound4cl_CBus *Get() const { return m_bus; }
00860         private:
00861             CProcessor& m_dynlib;
00862             sound4cl_CBus *m_bus;
00863     };
00864
00870     class CPresetLoader {
00871     public:
00872         CPresetLoader() = default;
00873         virtual ~CPresetLoader() = default;
00874
00875         virtual bool IsReadOnly() = 0;
00876         virtual bool Exists(const std::filesystem::path &name) = 0;
00877         virtual bool Remove(const std::filesystem::path &name) = 0;
00878         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00879         virtual std::vector<std::filesystem::path> GetAll() = 0;
00880         virtual std::string Read(const std::filesystem::path &filename) = 0;
00881         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00882     };
00883     enum PresetChange_Kind {
00884         change_kind_created  = sound4cl_PresetChange_Kind::change_kind_created,
00885         change_kind_modified = sound4cl_PresetChange_Kind::change_kind_modified,
00886         change_kind_deleted  = sound4cl_PresetChange_Kind::change_kind_deleted,
00887     };
00893     class CInstance {
00894     public:
00900         CInstance(CProcessor& dynlib)
00901         : m_dynlib(dynlib)
00902             {
```

```
00903                 params = m_dynlib.fnNewParameters();
00904                 if (!params) throw std::bad_alloc();
00905             }
00906
00915         CInstance(CProcessor& dynlib, void* _instance)
00916         : m_dynlib(dynlib)
00917         , instance(reinterpret_cast<sound4cl_CInstance*>(_instance))
00918         , m_bOwned(false)
00919             {
00920             }
00921         // non-copyable
00922         CInstance( const CInstance& ) = delete;
00923         CInstance& operator=( const CInstance& ) = delete;
00927         virtual ~CInstance()
00928             {
00929                 Stop();
00930                 if (params) m_dynlib.fnFreeParameters(params);
00931             }
00935        bool IsOk() const { return instance!=nullptr; }
00945         void SetParam(const std::string& name, const std::string& value)
00946             {
00947                 assert(params);
00948                 m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
00949             }
00956         std::string GetParam(const std::string& name)
00957             {
00958                 assert(params);
00959                 auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
00960                 std::string ret(c_value);
00961                 m_dynlib.fnFreeParameterValue(c_value);
00962                 return ret;
00963             }
00964
00973         void SetBus(const CBus& bus) {
00974             assert(params);
00975             if (m_dynlib.fnSetInstanceBus) {
00976                 m_dynlib.fnSetInstanceBus(params,bus.Get());
00977             }
00978         }
00979
00990         void SetPresetManager(CPresetLoader *preset_manager)
00991             {
00992                 if (!preset_manager) return;
00993                 if (!m_dynlib.fnSetPresetManager) return;
00994                 assert(params);
00995                 m_dynlib.fnSetPresetManager(params,
00996                     sound4cl_custom_reader,
00997                     sound4cl_custom_reader_free,
00998                     sound4cl_custom_writer,
00999                     sound4cl_custom_exists,
01000                     sound4cl_custom_getall,
01001                     sound4cl_custom_getall_free,
01002                     sound4cl_custom_remove,
01003                     sound4cl_custom_rename,
01004                     preset_manager->IsReadOnly(),
01005                     preset_manager
01006                 );
01007             }
01008
01031         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)
01032             {
01033                 assert(!instance);
01034                 std::string l_save_path;
01035                 if (!save_path.empty()) {
01036                     l_save_path = save_path.u8string();
01037                 }
01038                 if (frames_per_chunk==0) {
01039                     frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
01040                 }
01041                 if (m_dynlib.fnInitProcess3) {
01042                     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);
01043                 } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
01044                     return false;
01045                 } else {
01046                     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);
01047                 }
01048                 if (!instance) return false;
01049                 if (!init_metadata.empty()) {
01050                     for (auto&& [key,value]: init_metadata) {
01051                         m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
01052                     }
01053                     init_metadata.clear();
01054                 }
```

```
01055                 update_thread = std::thread([this, json_port](){
01056                     OnUpdateThreadStart();
01057                     m_dynlib.fnStartUpdateThread(instance, json_port);
01058                     OnUpdateThreadStop();
01059                 });
01060                 int timeout=5000;
01061                 #ifdef DEBUG
01062                     timeout=60*1000; // For debugger pauses
01063                 #endif
01064                 if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
01065                     return false;
01066                 }
01067                 return true;
01068             }
01069
01077         unsigned int GetChunkFrames() {
01078             if (instance && m_dynlib.fnGetProcessChunkFrames) {
01079                 return m_dynlib.fnGetProcessChunkFrames(instance);
01080             }
01081             return m_dynlib.fnGetChunkSizeInFrames();
01082         }
01083
01091         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
    change_kind)
01092             {
01093                 if (m_dynlib.fnPresetManager_InformChange) {
01094                     m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(),
    (sound4cl_PresetChange_Kind) change_kind);
01095                 }
01096             }
01097
01109         void SetMetadata(const std::string& key, const char* value)
01110             {
01111                 if (m_dynlib.fnSetMetadata) {
01112                     if (instance) {
01113                         m_dynlib.fnSetMetadata(instance, key.c_str(), value);
01114                     } else {
01115                         init_metadata.push_back( {key, value} );
01116                     }
01117                 }
01118             }
01119
01129         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01130             {
01131                 if (instance) {
01132                     if (m_dynlib.fnSetMetadataMulti) {
01133                         typedef const char* pchar;
01134                         pchar* keyvalue=new pchar[2*list.size()+1];
01135                         size_t n=0;
01136                         for (auto&& [key,value]: list) {
01137                             keyvalue[2*n+0]=key.c_str();
01138                             keyvalue[2*n+1]=value;
01139                             n++;
01140                         }
01141                         keyvalue[2*n+0]=nullptr;
01142
01143                         sound4cl_SetMetadataMulti(instance, keyvalue);
01144
01145                         delete[] keyvalue;
01146                     } else {
01147                         for (auto&& [key,value]: list) {
01148                             SetMetadata(key, value);
01149                         }
01150                     }
01151                 } else {
01152                     for (auto&& [key,value]: list) {
01153                         if (value) {
01154                             init_metadata.push_back( {key, value} );
01155                         }
01156                     }
01157                 }
01158             }
01159
01165         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01166             {
01167                 std::vector< std::tuple<std::string,std::string> > values;
01168                 if (m_dynlib.fnGetMetadataInfos) {
01169                     const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
01170                     if (c_values) {
01171                         for (const char** c_value=c_values; *c_value; ) {
01172                             std::string key(*c_value);
01173                             c_value++;
01174                             if (*c_value) {
01175                                 std::string descr(*c_value);
01176                                 c_value++;
01177                                 values.push_back( {key,descr} );
01178                             }
```

```
01179                       }
01180                       m_dynlib.fnFreeMetadataInfos(instance, c_values);
01181                   }
01182               }
01183               return values;
01184           }
01185
01189       virtual void OnUpdateThreadStart() {}
01193       virtual void OnUpdateThreadStop() {}
01194
01195
01209       bool StartWebServer(int http_port, int https_port=0)
01210           {
01211               assert(instance);
01212               assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01213               if (!m_dynlib.fnWebserver) {
01214                   return false;
01215               }
01216               webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
01217               if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01218                   return false;
01219               }
01220               int web_status=m_dynlib.fnWebserver_Status(webserver);
01221               if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01222                   return false;
01223               }
01224               if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01225                   return false;
01226               }
01227               return true;
01228           }
01229
01237           void StopWebServer(int timeout_ms = 1000)
01238           {
01239               if (!m_dynlib.fnWebserver_Stop) return;
01240               if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01241                   m_dynlib.fnWebserver_Stop(webserver, timeout_ms);
01242                   webserver = SOUND4_INVALID_WEBSERVER_ID;
01243               }
01244           }
01245
01249       void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
01250           {
01251               assert(instance);
01252               if (!m_dynlib.fnWebserver_SetAppHealth) return;
01253               m_dynlib.fnWebserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
    content.c_str());
01254           }
01258       void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01259           {
01260               assert(instance);
01261               if (!m_dynlib.fnWebserver_GetAppHealth) return;
01262               char* c_contenttype=nullptr;
01263               char* c_content=nullptr;
01264               m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01265               contenttype=c_contenttype;
01266               content=c_content;
01267               m_dynlib.fnWebserver_FreeString(c_contenttype);
01268               m_dynlib.fnWebserver_FreeString(c_content);
01269           }
01278       int Terminate()
01279           {
01280               if (!instance) return 0;
01281               return m_dynlib.fnTerminateProcess(instance);
01282           }
01283
01289       void Stop()
01290           {
01291               if (!instance) return;
01292               if (m_bOwned) {
01293                   StopWebServer();
01294                   if (update_thread.joinable()) {
01295                       m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
01296                       update_thread.join();
01297                   }
01298                   m_dynlib.fnExitProcess(instance);
01299               }
01300               instance=nullptr;
01301           }
01302
01312       unsigned int GetEstimatedDelay()
01313           {
01314               assert(instance);
01315               return m_dynlib.fnGetEstimatedDelay(instance);
01316           }
01317
```

```
01330         float* GetBufferIn()
01331             {
01332                 assert(instance);
01333                 return m_dynlib.fnGetBufferIn(instance);
01334 
01335             }
01347         float* GetBufferOut()
01348             {
01349                 assert(instance);
01350                 return m_dynlib.fnGetBufferOut(instance);
01351             }
01355         void ProcessAudio()
01356             {
01357                 assert(instance);
01358                 m_dynlib.fnProcessAudio(instance, GetBufferIn(), GetBufferOut());
01359             }
01371         void ProcessAudio_Planar(float const * const *input, float * const *output)
01372             {
01373                 assert(instance);
01374                 m_dynlib.fnProcessAudio_Planar(instance, input, output);
01375             }
01376 
01388             template<typename T>
01389         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01390             {
01391                 assert(instance);
01392                 std::vector<T> output;
01393                 unsigned int out_offset=0;
01394                 unsigned int in_offset=0;
01395                 unsigned int todo = input.size();
01396                 while (todo>0) {
01397                     unsigned int left = AddAudio(&input[in_offset], todo);
01398                     unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
01399                     output.resize(out_offset + out_avail);
01400                     GetAudio(&output[out_offset], out_avail);
01401                     out_offset+=out_avail;
01402                     in_offset += todo-left;
01403                     todo=left;
01404                 }
01405                 return output;
01406             }
01407 
01413         class CClient {
01414             public:
01415                 CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
01416                 : m_dynlib(dynlib)
01417                     {
01418                         assert(instance);
01419                         client=m_dynlib.fnNewClient(instance);
01420                         if (!client) throw std::bad_alloc();
01421                     }
01422                 // non-copyable
01423                 CClient( const CInstance& ) = delete;
01424                 CClient& operator=( const CInstance& ) = delete;
01425                 ~CClient()
01426                     {
01427                         if (client) {
01428                             m_dynlib.fnDeleteClient(client);
01429                             client=nullptr;
01430                         }
01431                     }
01439                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01440                     {
01441                         assert(client);
01442                         int need_save=0;
01443                         const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(),
      &need_save);
01444                         if (!canswer) return {};
01445                         std::string answer(canswer);
01446                         m_dynlib.fnFreeJsonAnswer (canswer);
01447                         if (NeedSave) {
01448                             *NeedSave=(need_save!=0);
01449                         }
01450                         return answer;
01451                     }
01452             private:
01453                 CProcessor& m_dynlib;
01454                 sound4cl_CClientInstance *client = nullptr;
01455         };
01463         std::shared_ptr<CClient> NewClient()
01464             {
01465                 assert(instance);
01466                 return std::make_shared<CClient>(instance, m_dynlib);
01467             }
01468 
01479         bool SaveState() {
01480             assert(instance);
```

```
01481             if (!m_dynlib.fnSaveState || m_dynlib.fnSaveState(instance)!=0) {
01482                 return false;
01483             }
01484             return true;
01485         }
01486 #ifdef _WIN32
01493         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01494             if (m_dynlib.fnSetInstanceTracing) {
01495                 m_dynlib.fnSetInstanceTracing(params, tracing_provider, activity_guid);
01496             }
01497         }
01504         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01505             assert(instance);
01506             if (m_dynlib.fnSetInstanceTracingProcessActivity) {
01507                 m_dynlib.fnSetInstanceTracingProcessActivity(instance, activity_guid);
01508             }
01509         }
01510 #endif // _WIN32
01511     protected:
01512             template<typename T>
01513         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01514             {
01515                 assert(instance);
01516                 return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01517             }
01518             template<typename T>
01519         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01520             {
01521                 assert(instance);
01522                 return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01523             }
01524
01525     private:
01526         CProcessor& m_dynlib;
01527         sound4cl_CParameters* params = nullptr;
01528         sound4cl_CInstance* instance = nullptr;
01529         std::thread update_thread;
01530         std::vector< std::pair<std::string,std::string> > init_metadata;
01531         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01532         bool m_bOwned = true;
01533     };
01534
01535 }; // namespace sound4
01536
01537
01538 // bridge C callbacks to CPresetLoader
01539 extern "C" {
01540     static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
01541         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01542         std::string content=preset_loader->Read(filename);
01543         return strdup(content.c_str());
01544     }
01545     static void sound4cl_custom_reader_free(char *content, void* userdata) {
01546         if (!content) return;
01547         free(content);
01548     }
01549     static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01550         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01551         auto res=preset_loader->Write(filename,content);
01552         return res?0:-1;
01553     }
01554     static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
01555         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01556         auto res=preset_loader->Exists(filename);
01557         return res?0:-1;
01558     }
01559     static fs_char** sound4cl_custom_getall(void* userdata) {
01560         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01561         auto res=preset_loader->GetAll();
01562         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01563         for (size_t n=0;n<res.size();n++) {
01564             all[n]=fs_strdup(res[n].c_str());
01565         }
01566         all[res.size()]=nullptr;
01567         return all;
01568     }
01569     static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
01570         if (!all) return;
01571         for (fs_char** one=all;*one;one++) {
01572             free(*one);
01573         }
01574         free(all);
01575     }
01576     static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
01577         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
```

```
01578         auto res=preset_loader->Remove(filename);
01579         return res?0:-1;
01580     }
01581     static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01582         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01583         auto res=preset_loader->Rename(from,to);
01584         return res?0:-1;
01585     }
01586
01587 };
```

15.27 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

15.27.1 Macro Definition Documentation

15.27.1.1 SOUND4_INVALID_WEBSERVER_ID

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

Definition at line 65 of file sound4cl_cdef.h.

15.27.1.2 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 66 of file sound4cl_cdef.h.

15.27.1.3 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 67 of file sound4cl_cdef.h.

15.27.2 Typedef Documentation

15.27.2.1 fs_char

```
typedef wchar_t fs_char
```

Definition at line 18 of file sound4cl_cdef.h.

15.27.2.2 TraceLoggingHProvider

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

Definition at line 210 of file sound4cl_cdef.h.

15.28 sound4cl_cdef.h

Go to the documentation of this file.

```
00001
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #ifdef __unix__
00010     #include <sys/types.h>
00011 #endif
00012
00013 #ifdef _WIN32
00014     #include <stringapiset.h>
00015     #include <wchar.h>
00017     #define fs_strdup wcsdup
00018     typedef wchar_t fs_char;
00019     #ifdef _MSC_VER
00020         #define __BYTE_ORDER__  REG_DWORD
00021         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00022         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00023     #endif
00024     #ifdef _MSC_VER
00025         #pragma warning(disable : 4996)
00026     #endif
00027 #else // !_WIN32
00029     #define fs_strdup strdup
00030     typedef char fs_char;
00031 #endif // !_WIN32
00032
00033 #ifdef __cplusplus
00034 extern "C" {
00035 #endif // __cplusplus
00036
00037 struct sound4cl_SOUND4_ProcessInfo {
00038     size_t struct_size;
00039     const char *name;
00040     const char *prefix;
00041     const char *shortname;
00042     const char *version;
00043 };
00044 enum sound4cl_LogSeverity {
00045     sound4cl_none     = 0,
00046     sound4cl_fatal    = 1,
00047     sound4cl_error    = 2,
00048     sound4cl_warning  = 3,
00049     sound4cl_info     = 4,
00050     sound4cl_verbose  = 5,
00051     sound4cl_verbose2 = 6,
00052     sound4cl_verbose3 = 7,
00053     sound4cl_verbose4 = 8,
00054     sound4cl_verbose5 = 9,
00055 };
00056 struct sound4cl_CInstance;
00057 struct sound4cl_CParameters;
00058 struct sound4cl_CBus;
00059 struct sound4cl_CClientInstance;
00060 enum sound4cl_PresetChange_Kind {
```

```
00061     change_kind_created = 1,
00062     change_kind_modified   ,
00063     change_kind_deleted    ,
00064 };
00065 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
00066 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
00067 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
00068
00072 enum sound4cl_SampleFormat {
00073     INVALID_FORMAT=0,
00074     S16_LE,
00075     S16_BE,
00076     S24_LE,
00077     S24_BE,
00078     S32_LE,
00079     S32_BE,
00080     F32_LE,
00081     F32_BE,
00082
00083     // Native definitions, only links to the physical ones
00084     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00085         S16_NATIVE=S16_LE,
00086         S24_NATIVE=S24_LE,
00087         S32_NATIVE=S32_LE,
00088         F32_NATIVE=F32_LE,
00089     #else
00090         S16_NATIVE=S16_BE,
00091         S24_NATIVE=S24_BE,
00092         S32_NATIVE=S32_BE,
00093         F32_NATIVE=F32_BE,
00094     #endif
00095 };
00096
00097 const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
00098 const char *sound4cl_GetVersion();
00099 unsigned int sound4cl_GetChunkSizeInFrames();
00100 unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
00101 unsigned int sound4cl_GetChannelCount();
00102 unsigned int sound4cl_GetAudioInputCount();
00103 unsigned int sound4cl_GetAudioOutputCount();
00104 unsigned int sound4cl_GetSampleRate();
00105 typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
00106 void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
00107 void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
00108 struct sound4cl_CParameters *sound4cl_NewParameters();
00109 void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
00110 void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
00111 const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
00112 void sound4cl_FreeParameterValue(const char *value);
00113 struct sound4cl_CInstance *sound4cl_InitProcess(
00114         const char *LoginKey,
00115         const char *RadioName,
00116         const char *Access_Key_ID,
00117         const char *Access_Key_Secret,
00118         const char *save_path
00119     );
00120 struct sound4cl_CInstance *sound4cl_InitProcess2(
00121         const char *LoginKey,
00122         const char *RadioName,
00123         const char *Access_Key_ID,
00124         const char *Access_Key_Secret,
00125         const char *save_path,
00126         const struct sound4cl_CParameters *parameters
00127     );
00128 struct sound4cl_CInstance *sound4cl_InitProcess3(
00129         const char *LoginKey,
00130         const char *RadioName,
00131         const char *Access_Key_ID,
00132         const char *Access_Key_Secret,
00133         const char *save_path,
00134         const struct sound4cl_CParameters *parameters,
00135         unsigned int frames_per_chunk
00136     );
00137 unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
00138 int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
00139 void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
00140 struct sound4cl_CBus* sound4cl_NewBus();
00141 void sound4cl_FreeBus(struct sound4cl_CBus *bus);
00142 void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
00143 typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
00144 typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
00145 typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00146 typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
00147 typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
00148 typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
00149 typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
00150 typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
```

```
00151 void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
00152     , sound4cl_storage_reader reader, sound4cl_storage_reader_free
00153     , sound4cl_storage_writer writer
00154     , sound4cl_storage_exists exists
00155     , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
00156     , sound4cl_storage_remove remove
00157     , sound4cl_storage_rename rename
00158     , int IsReadOnly
00159     , void* userdata
00160     );
00161 void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
      *relative_path, enum sound4cl_PresetChange_Kind change_kind);
00162 void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
00163 void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
00164 const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
00165 void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
00166 void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
00167 void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
00168 int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
00169 void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
00170 void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
00171     , float const * const *input
00172     , float * const *output);
00173 float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
00174 float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
00175 unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
00176 const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
00177 enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
00178 unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
00179 unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
00180 unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
      int nFrame, enum sound4cl_SampleFormat fmt );
00181 unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
00182 unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
00183 unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
      max_nFrame, enum sound4cl_SampleFormat fmt );
00184 void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00185 void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00186 void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
00187 void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
00188 void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame);
00189 void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
00190 void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00191 void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00192 struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
00193 void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
00194 const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
      *need_save);
00195 void sound4cl_FreeJsonAnswer(const char *json_str);
00196 int sound4cl_SaveState(struct sound4cl_CInstance *instance);
00197 uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port);
00198 uint64_t sound4cl_Webserver_tcp2(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port, const struct sound4cl_CParameters *parameters);
00199 uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
      sound4cl_CInstance *instance);
00200 int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
00201 int sound4cl_Webserver_Status(uint64_t id);
00202 void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
      contenttype, const char* content);
00203 void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
      contenttype, char** content);
00204 void sound4cl_Webserver_FreeString(char* str);
00205
00206 //   Tracing (Windows only)
00207 #ifdef _WIN32
00208     #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00209         struct _tlgProvider_t;
00210         typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00211     #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00212 void sound4cl_SetInstanceTracing(struct sound4cl_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00213 void sound4cl_SetInstanceTracingProcessActivity(struct sound4cl_CInstance *instance, GUID
      activity_guid);
00214 #endif // _WIN32
00215
00216
00217 #ifdef __cplusplus
00218 } // extern "C"
00219 #endif // __cplusplus
```

Index

_log_cb
  sound4, 84
  sound4::cloudx1, 90
  sound4::cloudx1::dyn, 99
_log_cb_c
  sound4, 83
  sound4::cloudx1, 86
  sound4::cloudx1::dyn, 93
~CBus
  sound4::CBus, 113
~CClient
  sound4::CInstance::CClient, 115
  sound4::cloudx1::CInstance::CClient, 116
  sound4::cloudx1::dyn::CInstance::CClient, 118
~CDynLoader
  sound4::cloudx1::dyn::helper::CDynLoader, 131
  sound4::helper::CDynLoader, 133
~CInstance
  sound4::CInstance, 137
  sound4::cloudx1::CInstance, 149
  sound4::cloudx1::dyn::CInstance, 161
~CPresetLoader
  sound4::cloudx1::CPresetLoader, 173
  sound4::cloudx1::dyn::CPresetLoader, 175
  sound4::CPresetLoader, 177
~MyPresetManager
  MyPresetManager, 188

AddAudio
  sound4::CInstance, 138
  sound4::cloudx1::CInstance, 149
  sound4::cloudx1::dyn::CDynLib, 121
  sound4::cloudx1::dyn::CInstance, 161
AddPadAudio
  sound4::cloudx1::dyn::CDynLib, 121
Audio convert helpers, 29
  cloudx1_AudioConvertFrom, 29
  cloudx1_AudioConvertTo, 30
  cloudx1_AudioMonoFromLiveStereo, 30
  cloudx1_AudioMonoToLiveStereo, 30
  cloudx1_MonoToStereo, 32
  cloudx1_MonoToStereo_Planar, 32
  cloudx1_StereoToMono, 32
  cloudx1_StereoToMono_Planar, 33
Audio convert helpers (C++), 33
  AudioConvertFrom, 34
  AudioConvertTo, 34
  AudioMonoFromLiveStereo, 35
  AudioMonoToLiveStereo, 35
  MonoToStereo, 35
  MonoToStereo_Planar, 36
  StereoToMono, 36
  StereoToMono_Planar, 37
Audio managing (any audio), 37
  cloudx1_AddAudio, 39
  cloudx1_AddPadAudio, 39
  cloudx1_GetAudio, 39
  cloudx1_GetBytesFromFormat, 40
  cloudx1_GetFormatFromName, 40
  cloudx1_GetFormatName, 40
  cloudx1_GetMaxPacketFrame, 41
  cloudx1_GetOutputCount, 41
  cloudx1_SampleFormat, 38
  F32_BE, 38
  F32_LE, 38
  F32_NATIVE, 38
  INVALID_FORMAT, 38
  S16_BE, 38
  S16_LE, 38
  S16_NATIVE, 38
  S24_BE, 38
  S24_LE, 38
  S24_NATIVE, 38
  S32_BE, 38
  S32_LE, 38
  S32_NATIVE, 38
AudioConvertFrom
  Audio convert helpers (C++), 34
  sound4::cloudx1::dyn::CDynLib, 121
  sound4::cloudx1::dyn::helper, 102
  sound4::cloudx1::helper, 107
AudioConvertTo
  Audio convert helpers (C++), 34
  sound4::cloudx1::dyn::CDynLib, 121
  sound4::cloudx1::dyn::helper, 102
  sound4::cloudx1::helper, 107
AudioMonoFromLiveStereo
  Audio convert helpers (C++), 35
  sound4::cloudx1::dyn::CDynLib, 121
  sound4::cloudx1::dyn::helper, 103
  sound4::cloudx1::helper, 107
AudioMonoToLiveStereo
  Audio convert helpers (C++), 35
  sound4::cloudx1::dyn::CDynLib, 121
  sound4::cloudx1::dyn::helper, 103
  sound4::cloudx1::helper, 108

CBus
  sound4::CBus, 113
  sound4::CProcessor, 184

CClient
sound4::CInstance::CClient, 115
sound4::cloudx1::CInstance::CClient, 116
sound4::cloudx1::dyn::CInstance::CClient, 117
CDynLib
sound4::cloudx1::dyn::CDynLib, 120
CDynLoader
sound4::cloudx1::dyn::helper::CDynLoader, 131
sound4::helper::CDynLoader, 133
change_kind_created
Custom preset managagement, 48
sound4, 83
change_kind_deleted
Custom preset managagement, 48
sound4, 83
change_kind_modified
Custom preset managagement, 48
sound4, 83
CHANNEL_COUNT
sound4.x1.cloud-pipe.cpp, 205
CHUNK_SIZE
sound4.x1.cloud-pipe.cpp, 205
ChunkFrames
sound4::cloudx1, 90
sound4::cloudx1::dyn, 99
CInstance
sound4::CBus, 114
sound4::CInstance, 137
sound4::cloudx1::CInstance, 149
sound4::cloudx1::dyn::CInstance, 161
sound4::CProcessor, 184
Client's JSon config interfacing, 41
cloudx1_DeleteClient, 42
cloudx1_FreeJsonAnswer, 42
cloudx1_NewClient, 42
cloudx1_ProcessJson, 43
cloudx1_SaveState, 43
Close
sound4::cloudx1::dyn::helper::CDynLoader, 131
sound4::helper::CDynLoader, 134
cloudx1_AddAudio
Audio managing (any audio), 39
cloudx1_AddPadAudio
Audio managing (any audio), 39
CLOUDX1_AUDIO_INPUT_COUNT
Global setup, 54
CLOUDX1_AUDIO_OUTPUT_COUNT
Global setup, 54
cloudx1_AudioConvertFrom
Audio convert helpers, 29
cloudx1_AudioConvertTo
Audio convert helpers, 30
CLOUDX1_AUDIOFRAME_COUNT
Global setup, 54
cloudx1_AudioMonoFromLiveStereo
Audio convert helpers, 30
cloudx1_AudioMonoToLiveStereo
Audio convert helpers, 30
cloudx1_CClientInstance, 172
CLOUDX1_CHANNEL_COUNT
Global setup, 54
cloudx1_CInstance, 172
cloudx1_CParameters, 172
cloudx1_custom_exists
sound4.x1.cloud.hpp, 224
sound4.x1.cloud_dyn.hpp, 236
cloudx1_custom_getall
sound4.x1.cloud.hpp, 224
sound4.x1.cloud_dyn.hpp, 236
cloudx1_custom_getall_free
sound4.x1.cloud.hpp, 224
sound4.x1.cloud_dyn.hpp, 236
cloudx1_custom_reader
sound4.x1.cloud.hpp, 225
sound4.x1.cloud_dyn.hpp, 236
cloudx1_custom_reader_free
sound4.x1.cloud.hpp, 225
sound4.x1.cloud_dyn.hpp, 236
cloudx1_custom_remove
sound4.x1.cloud.hpp, 225
sound4.x1.cloud_dyn.hpp, 237
cloudx1_custom_rename
sound4.x1.cloud.hpp, 225
sound4.x1.cloud_dyn.hpp, 237
cloudx1_custom_writer
sound4.x1.cloud.hpp, 225
sound4.x1.cloud_dyn.hpp, 237
CLOUDX1_DECL
sound4.x1.cloud.h, 217
cloudx1_DeleteClient
Client's JSon config interfacing, 42
cloudx1_ExitProcess
Process creation/destruction, 62
cloudx1_FreeJsonAnswer
Client's JSon config interfacing, 42
cloudx1_FreeMetadataInfos
Metadata Management, 60
cloudx1_FreeParameters
Process creation/destruction, 62
cloudx1_FreeParameterValue
Process creation/destruction, 62
cloudx1_GetAudio
Audio managing (any audio), 39
cloudx1_GetAudioInputCount
Global setup, 56
cloudx1_GetAudioOutputCount
Global setup, 56
cloudx1_GetBufferIn
Direct processing, 50
cloudx1_GetBufferOut
Direct processing, 50
cloudx1_GetBytesFromFormat
Audio managing (any audio), 40
cloudx1_GetChannelCount
Global setup, 56
cloudx1_GetChunkSizeInFrames

Global setup, 56
cloudx1_GetEstimatedDelay
Direct processing, 51
cloudx1_GetFormatFromName
Audio managing (any audio), 40
cloudx1_GetFormatName
Audio managing (any audio), 40
cloudx1_GetMaxPacketFrame
Audio managing (any audio), 41
cloudx1_GetMetadataInfos
Metadata Management, 60
cloudx1_GetOutputCount
Audio managing (any audio), 41
cloudx1_GetParameter
Process creation/destruction, 63
cloudx1_GetPossibleChunkSizeInFrames
Global setup, 57
cloudx1_GetProcessChunkFrames
Process creation/destruction, 63
cloudx1_GetSampleRate
Global setup, 57
cloudx1_GetVersion
Global setup, 57
CLOUDX1_HAS_CLOUDBUS
sound4.x1.cloud.h, 217
CLOUDX1_HAS_WEBSERVER
sound4.x1.cloud.h, 218
cloudx1_InitProcess
Process creation/destruction, 63
cloudx1_InitProcess2
Process creation/destruction, 64
cloudx1_InitProcess3
Process creation/destruction, 64
cloudx1_loggerfn
Global setup, 55
cloudx1_LogSeverity
Global setup, 55
cloudx1_MonoToStereo
Audio convert helpers, 32
cloudx1_MonoToStereo_Planar
Audio convert helpers, 32
cloudx1_NewClient
Client's JSon config interfacing, 42
cloudx1_NewParameters
Process creation/destruction, 65
cloudx1_PresetChange_Kind
Custom preset managagement, 48
cloudx1_PresetManager_InformChange
Custom preset managagement, 48
cloudx1_ProcessAudio
Direct processing, 51
cloudx1_ProcessAudio_Planar
Direct processing, 52
cloudx1_ProcessJson
Client's JSon config interfacing, 43
CLOUDX1_SAMPLE_RATE
Global setup, 54
cloudx1_SampleFormat
Audio managing (any audio), 38
cloudx1_SaveState
Client's JSon config interfacing, 43
cloudx1_SetInstanceTracing
Instance tracing (Windows only), 59
cloudx1_SetInstanceTracingProcessActivity
Instance tracing (Windows only), 59
cloudx1_SetLoggerCallback
Global setup, 58
cloudx1_SetLogSeverity
Global setup, 58
cloudx1_SetMetadata
Metadata Management, 60
cloudx1_SetMetadataMulti
Metadata Management, 61
cloudx1_SetParameter
Process creation/destruction, 65
cloudx1_SetPresetManager
Custom preset managagement, 49
cloudx1_StartUpdateThread
Update Management, 73
cloudx1_StereoToMono
Audio convert helpers, 32
cloudx1_StereoToMono_Planar
Audio convert helpers, 33
cloudx1_StopUpdateThread
Update Management, 73
cloudx1_storage_exists
Custom preset managagement, 45
cloudx1_storage_getall
Custom preset managagement, 45
cloudx1_storage_getall_free
Custom preset managagement, 46
cloudx1_storage_reader
Custom preset managagement, 46
cloudx1_storage_reader_free
Custom preset managagement, 46
cloudx1_storage_remove
Custom preset managagement, 47
cloudx1_storage_rename
Custom preset managagement, 47
cloudx1_storage_writer
Custom preset managagement, 47
cloudx1_TerminateProcess
Process creation/destruction, 65
cloudx1_WaitUpdateThreadReady
Update Management, 74
cloudx1_Webserver
Web server, 75
cloudx1_Webserver_FreeString
Web server, 76
cloudx1_Webserver_GetAppHealth
Web server, 76
cloudx1_Webserver_SetAppHealth
Web server, 76
cloudx1_Webserver_Status
Web server, 78
cloudx1_Webserver_Stop

Web server, 78
cloudx1_Webserver_tcp
Web server, 79
cloudx1_Webserver_tcp2
Web server, 79
CPresetLoader
sound4::cloudx1::CPresetLoader, 173
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177
CProcessor
sound4::CProcessor, 179
Create
sound4::CInstance, 138
sound4::cloudx1::CInstance, 149
sound4::cloudx1::dyn::CInstance, 161
Custom preset managagement, 44
change_kind_created, 48
change_kind_deleted, 48
change_kind_modified, 48
cloudx1_PresetChange_Kind, 48
cloudx1_PresetManager_InformChange, 48
cloudx1_SetPresetManager, 49
cloudx1_storage_exists, 45
cloudx1_storage_getall, 45
cloudx1_storage_getall_free, 46
cloudx1_storage_reader, 46
cloudx1_storage_reader_free, 46
cloudx1_storage_remove, 47
cloudx1_storage_rename, 47
cloudx1_storage_writer, 47
fs_char, 48

DeleteClient
sound4::cloudx1::dyn::CDynLib, 122
Deprecated List, 17
Direct processing, 49
cloudx1_GetBufferIn, 50
cloudx1_GetBufferOut, 50
cloudx1_GetEstimatedDelay, 51
cloudx1_ProcessAudio, 51
cloudx1_ProcessAudio_Planar, 52
doc_examples.h, 195
doc_gencpp.h, 195
doc_json.h, 195
doc_main.h, 196
doc_params.h, 196
DynFuncHolder
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 185
sound4::helper::DynFuncHolder< T >, 186

Environment variables and parameters, 13
error
Global setup, 55
sound4, 82
Examples, 52
Exists
MyPresetManager, 188
sound4::cloudx1::CPresetLoader, 173
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177
ExitProcess
sound4::cloudx1::dyn::CDynLib, 122

F32_BE
Audio managing (any audio), 38
sound4, 83
F32_LE
Audio managing (any audio), 38
sound4, 83
F32_NATIVE
Audio managing (any audio), 38
sound4, 83
fatal
Global setup, 55
sound4, 82
FreeJsonAnswer
sound4::cloudx1::dyn::CDynLib, 122
FreeMetadataInfos
sound4::cloudx1::dyn::CDynLib, 122
FreeParameters
sound4::cloudx1::dyn::CDynLib, 122
FreeParameterValue
sound4::cloudx1::dyn::CDynLib, 122
fs_char
Custom preset managagement, 48
sound4cl_cdef.h, 268

Generic C++ access to SOUND4 .CL librairies, 3
Get
sound4::CBus, 114
GetAll
MyPresetManager, 189
sound4::cloudx1::CPresetLoader, 173
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177
GetAudio
sound4::CInstance, 139
sound4::cloudx1::CInstance, 150
sound4::cloudx1::dyn::CDynLib, 122
sound4::cloudx1::dyn::CInstance, 162
GetAudioInputCount
sound4::cloudx1, 86
sound4::cloudx1::dyn, 93
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 179
GetAudioOutputCount
sound4::cloudx1, 86
sound4::cloudx1::dyn, 94
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 179
GetBufferIn
sound4::CInstance, 139
sound4::cloudx1::CInstance, 150
sound4::cloudx1::dyn::CDynLib, 123
sound4::cloudx1::dyn::CInstance, 162
GetBufferOut
sound4::CInstance, 139

sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CDynLib, 123
sound4::cloudx1::dyn::CInstance, 163
GetBytesFromFormat
sound4::cloudx1, 86
sound4::cloudx1::dyn, 94
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 180
GetChannelCount
sound4::cloudx1, 87
sound4::cloudx1::dyn, 94
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 180
GetChunkFrames
sound4::CInstance, 139
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CInstance, 163
GetChunkSizeInFrames
sound4::cloudx1, 87
sound4::cloudx1::dyn, 95
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 180
GetEnv
sound4.x1.cloud-pipe-cxx.cpp, 196
sound4.x1.cloud-pipe-dyn.cpp, 200
sound4cl-pipe.cpp, 210
GetEstimatedDelay
sound4::CInstance, 140
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 163
GetFormatFromName
sound4::cloudx1, 87
sound4::cloudx1::dyn, 95
sound4::cloudx1::dyn::CDynLib, 124
sound4::CProcessor, 181
GetFormatName
sound4::cloudx1, 88
sound4::cloudx1::dyn, 96
sound4::cloudx1::dyn::CDynLib, 124
sound4::CProcessor, 181
GetMaxPacketFrame
sound4::cloudx1::dyn::CDynLib, 124
GetMetadataInfos
sound4::CInstance, 140
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 164
GetOutputCount
sound4::cloudx1::dyn::CDynLib, 124
GetParam
sound4::CInstance, 140
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CInstance, 164
GetParameter
sound4::cloudx1::dyn::CDynLib, 124
GetPossibleChunkSizeInFrames
sound4::cloudx1, 88
sound4::cloudx1::dyn, 96
sound4::cloudx1::dyn::CDynLib, 125
sound4::CProcessor, 182
GetPrefixSymbol
sound4::CProcessor, 182
GetProcessChunkFrames
sound4::cloudx1::dyn::CDynLib, 125
GetSampleRate
sound4::cloudx1, 88
sound4::cloudx1::dyn, 97
sound4::cloudx1::dyn::CDynLib, 125
sound4::CProcessor, 182
GetSymbol
sound4::cloudx1::dyn::helper::CDynLoader, 131
sound4::helper::CDynLoader, 134
GetSymbol_safe
sound4::cloudx1::dyn::helper::CDynLoader, 131
sound4::helper::CDynLoader, 134
GetThisLibraryPath
sound4::cloudx1::dyn::helper::CDynLoader, 132
sound4::helper::CDynLoader, 134
GetVersion
sound4::cloudx1, 89
sound4::cloudx1::dyn, 97
sound4::cloudx1::dyn::CDynLib, 125
sound4::CProcessor, 182
GetWebServerAppHealth
sound4::CInstance, 141
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CInstance, 164
Global setup, 53
CLOUDX1_AUDIO_INPUT_COUNT, 54
CLOUDX1_AUDIO_OUTPUT_COUNT, 54
CLOUDX1_AUDIOFRAME_COUNT, 54
CLOUDX1_CHANNEL_COUNT, 54
cloudx1_GetAudioInputCount, 56
cloudx1_GetAudioOutputCount, 56
cloudx1_GetChannelCount, 56
cloudx1_GetChunkSizeInFrames, 56
cloudx1_GetPossibleChunkSizeInFrames, 57
cloudx1_GetSampleRate, 57
cloudx1_GetVersion, 57
cloudx1_loggerfn, 55
cloudx1_LogSeverity, 55
CLOUDX1_SAMPLE_RATE, 54
cloudx1_SetLoggerCallback, 58
cloudx1_SetLogSeverity, 58
error, 55
fatal, 55
info, 55
none, 55
SOUND4_GetProcessInfo, 58
verbose, 55
verbose2, 55
verbose3, 55
verbose4, 55
verbose5, 55
warning, 55

info
Global setup, 55
sound4, 82
InitProcess
sound4::cloudx1::dyn::CDynLib, 125
InitProcess2
sound4::cloudx1::dyn::CDynLib, 125
InitProcess3
sound4::cloudx1::dyn::CDynLib, 125
InputChannels
sound4::cloudx1, 90
sound4::cloudx1::dyn, 99
InputSampleSize
sound4::cloudx1, 90
sound4::cloudx1::dyn, 100
Instance tracing (Windows only), 59
cloudx1_SetInstanceTracing, 59
cloudx1_SetInstanceTracingProcessActivity, 59
INVALID_FORMAT
Audio managing (any audio), 38
sound4, 83
IsOk
sound4::CInstance, 141
sound4::cloudx1::CInstance, 153
sound4::cloudx1::dyn::CDynLib, 120
sound4::cloudx1::dyn::CInstance, 165
sound4::cloudx1::dyn::helper::CDynLoader, 132
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 185
sound4::CProcessor, 183
sound4::helper::CDynLoader, 134
sound4::helper::DynFuncHolder< T >, 186
IsReadOnly
MyPresetManager, 189
sound4::cloudx1::CPresetLoader, 173
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177

JSON syntax, 5
JSON syntax for internal setup, 9

Load
sound4::cloudx1::dyn::CDynLib, 120
sound4::cloudx1::dyn::helper::CDynLoader, 132
sound4::CProcessor, 183
sound4::helper::CDynLoader, 134
log_cb_t
sound4, 82
sound4::cloudx1, 85
sound4::cloudx1::dyn, 93
LogSeverity
sound4, 82
sound4::cloudx1, 85
sound4::cloudx1::dyn, 93

main
sound4.x1.cloud-pipe-cxx.cpp, 196
sound4.x1.cloud-pipe-dyn.cpp, 200
sound4.x1.cloud-pipe.cpp, 206
sound4cl-pipe.cpp, 210
Metadata Management, 60
cloudx1_FreeMetadataInfos, 60
cloudx1_GetMetadataInfos, 60
cloudx1_SetMetadata, 60
cloudx1_SetMetadataMulti, 61
MonoToStereo
Audio convert helpers (C++), 35
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::helper, 103
sound4::cloudx1::helper, 108
MonoToStereo_Planar
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::helper, 104
sound4::cloudx1::helper, 109
MyLogger
sound4.x1.cloud-pipe-cxx.cpp, 197
sound4.x1.cloud-pipe-dyn.cpp, 201
sound4.x1.cloud-pipe.cpp, 206
MyPresetManager, 187
~MyPresetManager, 188
Exists, 188
GetAll, 189
IsReadOnly, 189
MyPresetManager, 188
Read, 189
Remove, 190
Rename, 190
SetupPath, 190, 191
Write, 191

name
SOUND4_ProcessInfo, 193
NewClient
sound4::CInstance, 141
sound4::cloudx1::CInstance, 153
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::CInstance, 165
NewParameters
sound4::cloudx1::dyn::CDynLib, 126
none
Global setup, 55
sound4, 82

OnUpdateThreadStart
sound4::CInstance, 141
sound4::cloudx1::CInstance, 153
sound4::cloudx1::dyn::CInstance, 165
OnUpdateThreadStop
sound4::CInstance, 141
sound4::cloudx1::CInstance, 153
sound4::cloudx1::dyn::CInstance, 165
operator bool
sound4::cloudx1::dyn::helper::CDynLoader, 132
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 185
sound4::helper::CDynLoader, 135
sound4::helper::DynFuncHolder< T >, 186

operator T∗
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 185
sound4::helper::DynFuncHolder< T >, 187
operator=
sound4::CInstance, 142
sound4::CInstance::CClient, 115
sound4::cloudx1::CInstance, 153
sound4::cloudx1::CInstance::CClient, 116
sound4::cloudx1::dyn::CInstance, 165
sound4::cloudx1::dyn::CInstance::CClient, 118
sound4::cloudx1::dyn::helper::CDynLoader, 132
sound4::helper::CDynLoader, 135
OutputChannels
sound4::cloudx1, 90
sound4::cloudx1::dyn, 100
OutputSampleSize
sound4::cloudx1, 91
sound4::cloudx1::dyn, 100

prefix
SOUND4_ProcessInfo, 193
PresetChange_Kind
sound4, 82
sound4::cloudx1, 85
sound4::cloudx1::dyn, 93
PresetManager_InformChange
sound4::CInstance, 142
sound4::cloudx1::CInstance, 153
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::CInstance, 165
Process creation/destruction, 61
cloudx1_ExitProcess, 62
cloudx1_FreeParameters, 62
cloudx1_FreeParameterValue, 62
cloudx1_GetParameter, 63
cloudx1_GetProcessChunkFrames, 63
cloudx1_InitProcess, 63
cloudx1_InitProcess2, 64
cloudx1_InitProcess3, 64
cloudx1_NewParameters, 65
cloudx1_SetParameter, 65
cloudx1_TerminateProcess, 65
process_name
sound4::cloudx1, 91
sound4::cloudx1::dyn, 100
process_shortname
sound4::cloudx1, 91
sound4::cloudx1::dyn, 100
ProcessAnyAudio
sound4::CInstance, 142
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CInstance, 166
ProcessAudio
sound4::CInstance, 143
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::CInstance, 166
ProcessAudio_Planar
sound4::CInstance, 143
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::CInstance, 166
ProcessJson
sound4::CInstance::CClient, 115
sound4::cloudx1::CInstance::CClient, 116
sound4::cloudx1::dyn::CDynLib, 127
sound4::cloudx1::dyn::CInstance::CClient, 118
ProcessLoop
sound4.x1.cloud-pipe-cxx.cpp, 197
sound4.x1.cloud-pipe-dyn.cpp, 201
sound4.x1.cloud-pipe.cpp, 206
sound4cl-pipe.cpp, 211

Read
MyPresetManager, 189
sound4::cloudx1::CPresetLoader, 173
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177
Remove
MyPresetManager, 190
sound4::cloudx1::CPresetLoader, 174
sound4::cloudx1::dyn::CPresetLoader, 175
sound4::CPresetLoader, 177
Rename
MyPresetManager, 190
sound4::cloudx1::CPresetLoader, 174
sound4::cloudx1::dyn::CPresetLoader, 176
sound4::CPresetLoader, 178

S16_BE
Audio managing (any audio), 38
sound4, 83
S16_LE
Audio managing (any audio), 38
sound4, 83
S16_NATIVE
Audio managing (any audio), 38
sound4, 83
S24_BE
Audio managing (any audio), 38
sound4, 83
S24_LE
Audio managing (any audio), 38
sound4, 83
S24_NATIVE
Audio managing (any audio), 38
sound4, 83
S32_BE
Audio managing (any audio), 38
sound4, 83
S32_LE
Audio managing (any audio), 38
sound4, 83
S32_NATIVE
Audio managing (any audio), 38
sound4, 83
Sample using C interface, 66

Sample using C++ interface wrapper, 67
Sample using C++ interface wrapper with dynamic library loading, 68
Sample using Generic C++ interface (multi-process compatible), 70
SAMPLE_RATE
    sound4.x1.cloud-pipe.cpp, 205
SampleFormat
    sound4, 83
    sound4::cloudx1, 85
    sound4::cloudx1::dyn, 93
SampleRate
    sound4::cloudx1, 91
    sound4::cloudx1::dyn, 101
SanityCheck
    sound4::cloudx1, 89
    sound4::cloudx1::dyn, 97
SaveState
    sound4::CInstance, 143
    sound4::cloudx1::CInstance, 156
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::cloudx1::dyn::CInstance, 168
SetBus
    sound4::CInstance, 144
SetInstanceTracing
    sound4::CInstance, 144
    sound4::cloudx1::CInstance, 156
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::cloudx1::dyn::CInstance, 168
SetInstanceTracingProcessActivity
    sound4::CInstance, 144
    sound4::cloudx1::CInstance, 156
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::cloudx1::dyn::CInstance, 168
SetLoggerCallback
    sound4::cloudx1, 89
    sound4::cloudx1::dyn, 99
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::CProcessor, 183
SetLogSeverity
    sound4::cloudx1, 89
    sound4::cloudx1::dyn, 99
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::CProcessor, 183
SetMetadata
    sound4::CInstance, 145
    sound4::cloudx1::CInstance, 157
    sound4::cloudx1::dyn::CDynLib, 127
    sound4::cloudx1::dyn::CInstance, 169
SetMetadataMulti
    sound4::CInstance, 145
    sound4::cloudx1::CInstance, 157
    sound4::cloudx1::dyn::CDynLib, 128
    sound4::cloudx1::dyn::CInstance, 169
SetParam
    sound4::CInstance, 145
    sound4::cloudx1::CInstance, 157
    sound4::cloudx1::dyn::CInstance, 169
SetParameter
    sound4::cloudx1::dyn::CDynLib, 128
SetPresetManager
    sound4::CInstance, 146
    sound4::cloudx1::CInstance, 158
    sound4::cloudx1::dyn::CDynLib, 128
    sound4::cloudx1::dyn::CInstance, 170
SetupPath
    MyPresetManager, 190, 191
SetWebServerAppHealth
    sound4::CInstance, 146
    sound4::cloudx1::CInstance, 158
    sound4::cloudx1::dyn::CInstance, 170
shortname
    SOUND4_ProcessInfo, 194
sound4, 81
    _log_cb, 84
    _log_cb_c, 83
    change_kind_created, 83
    change_kind_deleted, 83
    change_kind_modified, 83
    error, 82
    F32_BE, 83
    F32_LE, 83
    F32_NATIVE, 83
    fatal, 82
    info, 82
    INVALID_FORMAT, 83
    log_cb_t, 82
    LogSeverity, 82
    none, 82
    PresetChange_Kind, 82
    S16_BE, 83
    S16_LE, 83
    S16_NATIVE, 83
    S24_BE, 83
    S24_LE, 83
    S24_NATIVE, 83
    S32_BE, 83
    S32_LE, 83
    S32_NATIVE, 83
    SampleFormat, 83
    verbose, 82
    verbose2, 82
    verbose3, 82
    verbose4, 82
    verbose5, 82
    warning, 82
SOUND4 x1.CLOUD Library, 1
sound4.x1.cloud-pipe-cxx.cpp, 196, 198
    GetEnv, 196
    main, 196
    MyLogger, 197
    ProcessLoop, 197
    Usage, 197
sound4.x1.cloud-pipe-dyn.cpp, 200, 202
    GetEnv, 200
    main, 200

MyLogger, 201
ProcessLoop, 201
Usage, 201
sound4.x1.cloud-pipe.cpp, 205, 207
CHANNEL_COUNT, 205
CHUNK_SIZE, 205
main, 206
MyLogger, 206
ProcessLoop, 206
SAMPLE_RATE, 205
Usage, 206
sound4.x1.cloud.h, 215
CLOUDX1_DECL, 217
CLOUDX1_HAS_CLOUDBUS, 217
CLOUDX1_HAS_WEBSERVER, 218
TraceLoggingHProvider, 218
sound4.x1.cloud.hpp, 222
cloudx1_custom_exists, 224
cloudx1_custom_getall, 224
cloudx1_custom_getall_free, 224
cloudx1_custom_reader, 225
cloudx1_custom_reader_free, 225
cloudx1_custom_remove, 225
cloudx1_custom_rename, 225
cloudx1_custom_writer, 225
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_exists, 236
cloudx1_custom_getall, 236
cloudx1_custom_getall_free, 236
cloudx1_custom_reader, 236
cloudx1_custom_reader_free, 236
cloudx1_custom_remove, 237
cloudx1_custom_rename, 237
cloudx1_custom_writer, 237
UNIXLIKE, 236
sound4::CBus, 113
~CBus, 113
CBus, 113
CInstance, 114
Get, 114
sound4::CInstance, 135
~CInstance, 137
AddAudio, 138
CInstance, 137
Create, 138
GetAudio, 139
GetBufferIn, 139
GetBufferOut, 139
GetChunkFrames, 139
GetEstimatedDelay, 140
GetMetadataInfos, 140
GetParam, 140
GetWebServerAppHealth, 141
IsOk, 141
NewClient, 141
OnUpdateThreadStart, 141
OnUpdateThreadStop, 141
operator=, 142
PresetManager_InformChange, 142
ProcessAnyAudio, 142
ProcessAudio, 143
ProcessAudio_Planar, 143
SaveState, 143
SetBus, 144
SetInstanceTracing, 144
SetInstanceTracingProcessActivity, 144
SetMetadata, 145
SetMetadataMulti, 145
SetParam, 145
SetPresetManager, 146
SetWebServerAppHealth, 146
StartWebServer, 146
Stop, 147
StopWebServer, 147
Terminate, 147
sound4::CInstance::CClient, 114
~CClient, 115
CClient, 115
operator=, 115
ProcessJson, 115
sound4::cloudx1, 84
_log_cb, 90
_log_cb_c, 86
ChunkFrames, 90
GetAudioInputCount, 86
GetAudioOutputCount, 86
GetBytesFromFormat, 86
GetChannelCount, 87
GetChunkSizeInFrames, 87
GetFormatFromName, 87
GetFormatName, 88
GetPossibleChunkSizeInFrames, 88
GetSampleRate, 88
GetVersion, 89
InputChannels, 90
InputSampleSize, 90
log_cb_t, 85
LogSeverity, 85
OutputChannels, 90
OutputSampleSize, 91
PresetChange_Kind, 85
process_name, 91
process_shortname, 91
SampleFormat, 85
SampleRate, 91
SanityCheck, 89
SetLoggerCallback, 89
SetLogSeverity, 89
sound4::cloudx1::CInstance, 148
~CInstance, 149
AddAudio, 149
CInstance, 149
Create, 149
GetAudio, 150
GetBufferIn, 150
GetBufferOut, 151

GetChunkFrames, 151
GetEstimatedDelay, 151
GetMetadataInfos, 152
GetParam, 152
GetWebServerAppHealth, 152
IsOk, 153
NewClient, 153
OnUpdateThreadStart, 153
OnUpdateThreadStop, 153
operator=, 153
PresetManager_InformChange, 153
ProcessAnyAudio, 154
ProcessAudio, 154
ProcessAudio_Planar, 154
SaveState, 156
SetInstanceTracing, 156
SetInstanceTracingProcessActivity, 156
SetMetadata, 157
SetMetadataMulti, 157
SetParam, 157
SetPresetManager, 158
SetWebServerAppHealth, 158
StartWebServer, 158
Stop, 159
StopWebServer, 159
Terminate, 159
sound4::cloudx1::CInstance::CClient, 116
~CClient, 116
CClient, 116
operator=, 116
ProcessJson, 116
sound4::cloudx1::CPresetLoader, 172
~CPresetLoader, 173
CPresetLoader, 173
Exists, 173
GetAll, 173
IsReadOnly, 173
Read, 173
Remove, 174
Rename, 174
Write, 174
sound4::cloudx1::dyn, 92
_log_cb, 99
_log_cb_c, 93
ChunkFrames, 99
GetAudioInputCount, 93
GetAudioOutputCount, 94
GetBytesFromFormat, 94
GetChannelCount, 94
GetChunkSizeInFrames, 95
GetFormatFromName, 95
GetFormatName, 96
GetPossibleChunkSizeInFrames, 96
GetSampleRate, 97
GetVersion, 97
InputChannels, 99
InputSampleSize, 100
log_cb_t, 93
LogSeverity, 93
OutputChannels, 100
OutputSampleSize, 100
PresetChange_Kind, 93
process_name, 100
process_shortname, 100
SampleFormat, 93
SampleRate, 101
SanityCheck, 97
SetLoggerCallback, 99
SetLogSeverity, 99
sound4::cloudx1::dyn::CDynLib, 118
AddAudio, 121
AddPadAudio, 121
AudioConvertFrom, 121
AudioConvertTo, 121
AudioMonoFromLiveStereo, 121
AudioMonoToLiveStereo, 121
CDynLib, 120
DeleteClient, 122
ExitProcess, 122
FreeJsonAnswer, 122
FreeMetadataInfos, 122
FreeParameters, 122
FreeParameterValue, 122
GetAudio, 122
GetAudioInputCount, 123
GetAudioOutputCount, 123
GetBufferIn, 123
GetBufferOut, 123
GetBytesFromFormat, 123
GetChannelCount, 123
GetChunkSizeInFrames, 123
GetEstimatedDelay, 124
GetFormatFromName, 124
GetFormatName, 124
GetMaxPacketFrame, 124
GetMetadataInfos, 124
GetOutputCount, 124
GetParameter, 124
GetPossibleChunkSizeInFrames, 125
GetProcessChunkFrames, 125
GetSampleRate, 125
GetVersion, 125
InitProcess, 125
InitProcess2, 125
InitProcess3, 125
IsOk, 120
Load, 120
MonoToStereo, 126
MonoToStereo_Planar, 126
NewClient, 126
NewParameters, 126
PresetManager_InformChange, 126
ProcessAudio, 126
ProcessAudio_Planar, 126
ProcessJson, 127
SaveState, 127

SetInstanceTracing, 127
SetInstanceTracingProcessActivity, 127
SetLoggerCallback, 127
SetLogSeverity, 127
SetMetadata, 127
SetMetadataMulti, 128
SetParameter, 128
SetPresetManager, 128
StartUpdateThread, 128
StereoToMono, 128
StereoToMono_Planar, 128
StopUpdateThread, 128
TerminateProcess, 129
WaitUpdateThreadReady, 129
Webserver, 129
Webserver_FreeString, 129
Webserver_GetAppHealth, 129
Webserver_SetAppHealth, 129
Webserver_Status, 129
Webserver_Stop, 130
Webserver_tcp, 130
Webserver_tcp2, 130
sound4::cloudx1::dyn::CInstance, 160
~CInstance, 161
AddAudio, 161
CInstance, 161
Create, 161
GetAudio, 162
GetBufferIn, 162
GetBufferOut, 163
GetChunkFrames, 163
GetEstimatedDelay, 163
GetMetadataInfos, 164
GetParam, 164
GetWebServerAppHealth, 164
IsOk, 165
NewClient, 165
OnUpdateThreadStart, 165
OnUpdateThreadStop, 165
operator=, 165
PresetManager_InformChange, 165
ProcessAnyAudio, 166
ProcessAudio, 166
ProcessAudio_Planar, 166
SaveState, 168
SetInstanceTracing, 168
SetInstanceTracingProcessActivity, 168
SetMetadata, 169
SetMetadataMulti, 169
SetParam, 169
SetPresetManager, 170
SetWebServerAppHealth, 170
StartWebServer, 170
Stop, 171
StopWebServer, 171
Terminate, 171
sound4::cloudx1::dyn::CInstance::CClient, 117
~CClient, 118
CClient, 117
operator=, 118
ProcessJson, 118
sound4::cloudx1::dyn::CPresetLoader, 174
~CPresetLoader, 175
CPresetLoader, 175
Exists, 175
GetAll, 175
IsReadOnly, 175
Read, 175
Remove, 175
Rename, 176
Write, 176
sound4::cloudx1::dyn::helper, 101
AudioConvertFrom, 102
AudioConvertTo, 102
AudioMonoFromLiveStereo, 103
AudioMonoToLiveStereo, 103
MonoToStereo, 103
MonoToStereo_Planar, 104
StereoToMono, 104
StereoToMono_Planar, 105
UTF8ToWString, 105
WStringToUTF8, 106
sound4::cloudx1::dyn::helper::CDynLoader, 130
~CDynLoader, 131
CDynLoader, 131
Close, 131
GetSymbol, 131
GetSymbol_safe, 131
GetThisLibraryPath, 132
IsOk, 132
Load, 132
operator bool, 132
operator=, 132
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 184
DynFuncHolder, 185
IsOk, 185
operator bool, 185
operator T*, 185
sound4::cloudx1::dyn::helper::SampleFormat< T >, 191
sound4::cloudx1::helper, 106
AudioConvertFrom, 107
AudioConvertTo, 107
AudioMonoFromLiveStereo, 107
AudioMonoToLiveStereo, 108
MonoToStereo, 108
MonoToStereo_Planar, 109
StereoToMono, 109
StereoToMono_Planar, 109
UTF8ToWString, 110
WStringToUTF8, 110
sound4::cloudx1::helper::SampleFormat< T >, 192
sound4::CPresetLoader, 176
~CPresetLoader, 177
CPresetLoader, 177

Exists, 177
GetAll, 177
IsReadOnly, 177
Read, 177
Remove, 177
Rename, 178
Write, 178
sound4::CProcessor, 178
CBus, 184
CInstance, 184
CProcessor, 179
GetAudioInputCount, 179
GetAudioOutputCount, 179
GetBytesFromFormat, 180
GetChannelCount, 180
GetChunkSizeInFrames, 180
GetFormatFromName, 181
GetFormatName, 181
GetPossibleChunkSizeInFrames, 182
GetPrefixSymbol, 182
GetSampleRate, 182
GetVersion, 182
IsOk, 183
Load, 183
SetLoggerCallback, 183
SetLogSeverity, 183
sound4::helper, 111
UTF8ToWString, 111
WStringToUTF8, 111
sound4::helper::CDynLoader, 133
∼CDynLoader, 133
CDynLoader, 133
Close, 134
GetSymbol, 134
GetSymbol_safe, 134
GetThisLibraryPath, 134
IsOk, 134
Load, 134
operator bool, 135
operator=, 135
sound4::helper::DynFuncHolder< T >, 186
DynFuncHolder, 186
IsOk, 186
operator bool, 186
operator T∗, 187
sound4::helper::SampleFormat< T >, 192
SOUND4_GetProcessInfo
Global setup, 58
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 267
Web server, 75
SOUND4_ProcessInfo, 193
name, 193
prefix, 193
shortname, 194
struct_size, 194
version, 194
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 267
Web server, 75
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 267
Web server, 75
sound4cl-pipe.cpp, 210, 212
GetEnv, 210
main, 210
ProcessLoop, 211
sound4cl.hpp, 251
UNIXLIKE, 253
sound4cl_cdef.h, 267
fs_char, 268
SOUND4_INVALID_WEBSERVER_ID, 267
SOUND4_WEBSERVER_HTTP_OK, 267
SOUND4_WEBSERVER_HTTPS_OK, 267
TraceLoggingHProvider, 268
StartUpdateThread
sound4::cloudx1::dyn::CDynLib, 128
StartWebServer
sound4::CInstance, 146
sound4::cloudx1::CInstance, 158
sound4::cloudx1::dyn::CInstance, 170
StereoToMono
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 128
sound4::cloudx1::dyn::helper, 104
sound4::cloudx1::helper, 109
StereoToMono_Planar
Audio convert helpers (C++), 37
sound4::cloudx1::dyn::CDynLib, 128
sound4::cloudx1::dyn::helper, 105
sound4::cloudx1::helper, 109
Stop
sound4::CInstance, 147
sound4::cloudx1::CInstance, 159
sound4::cloudx1::dyn::CInstance, 171
StopUpdateThread
sound4::cloudx1::dyn::CDynLib, 128
StopWebServer
sound4::CInstance, 147
sound4::cloudx1::CInstance, 159
sound4::cloudx1::dyn::CInstance, 171
struct_size
SOUND4_ProcessInfo, 194

Terminate
sound4::CInstance, 147
sound4::cloudx1::CInstance, 159
sound4::cloudx1::dyn::CInstance, 171
TerminateProcess
sound4::cloudx1::dyn::CDynLib, 129
TraceLoggingHProvider
sound4.x1.cloud.h, 218
sound4cl_cdef.h, 268

UNIXLIKE
sound4.x1.cloud_dyn.hpp, 236
sound4cl.hpp, 253

Update Management, 73
cloudx1_StartUpdateThread, 73
cloudx1_StopUpdateThread, 73
cloudx1_WaitUpdateThreadReady, 74
Usage
sound4.x1.cloud-pipe-cxx.cpp, 197
sound4.x1.cloud-pipe-dyn.cpp, 201
sound4.x1.cloud-pipe.cpp, 206
UTF8ToWString
sound4::cloudx1::dyn::helper, 105
sound4::cloudx1::helper, 110
sound4::helper, 111

verbose
Global setup, 55
sound4, 82
verbose2
Global setup, 55
sound4, 82
verbose3
Global setup, 55
sound4, 82
verbose4
Global setup, 55
sound4, 82
verbose5
Global setup, 55
sound4, 82
version
SOUND4_ProcessInfo, 194

WaitUpdateThreadReady
sound4::cloudx1::dyn::CDynLib, 129
warning
Global setup, 55
sound4, 82
Web server, 74
cloudx1_Webserver, 75
cloudx1_Webserver_FreeString, 76
cloudx1_Webserver_GetAppHealth, 76
cloudx1_Webserver_SetAppHealth, 76
cloudx1_Webserver_Status, 78
cloudx1_Webserver_Stop, 78
cloudx1_Webserver_tcp, 79
cloudx1_Webserver_tcp2, 79
SOUND4_INVALID_WEBSERVER_ID, 75
SOUND4_WEBSERVER_HTTP_OK, 75
SOUND4_WEBSERVER_HTTPS_OK, 75
Webserver
sound4::cloudx1::dyn::CDynLib, 129
Webserver_FreeString
sound4::cloudx1::dyn::CDynLib, 129
Webserver_GetAppHealth
sound4::cloudx1::dyn::CDynLib, 129
Webserver_SetAppHealth
sound4::cloudx1::dyn::CDynLib, 129
Webserver_Status
sound4::cloudx1::dyn::CDynLib, 129
Webserver_Stop
sound4::cloudx1::dyn::CDynLib, 130
Webserver_tcp
sound4::cloudx1::dyn::CDynLib, 130
Webserver_tcp2
sound4::cloudx1::dyn::CDynLib, 130
Write
MyPresetManager, 191
sound4::cloudx1::CPresetLoader, 174
sound4::cloudx1::dyn::CPresetLoader, 176
sound4::CPresetLoader, 178
WStringToUTF8
sound4::cloudx1::dyn::helper, 106
sound4::cloudx1::helper, 110
sound4::helper, 111