SOUND4 x1.CLOUD Library [1.5.6]

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 Metadata Management . . . 59
12.9.1 Detailed Description . . . 59
12.9.2 Function Documentation . . . 59
12.9.2.1 cloudx1_FreeMetadataInfos() . . . 59
12.9.2.2 cloudx1_GetMetadataInfos() . . . 59
12.9.2.3 cloudx1_SetMetadata() . . . 60
12.9.2.4 cloudx1_SetMetadataMulti() . . . 60
12.10 Process creation/destruction . . . 61
12.10.1 Detailed Description . . . 61
12.10.2 Function Documentation . . . 61
12.10.2.1 cloudx1_ExitProcess() . . . 61
12.10.2.2 cloudx1_FreeParameters() . . . 62
12.10.2.3 cloudx1_FreeParameterValue() . . . 62
12.10.2.4 cloudx1_GetParameter() . . . 62
12.10.2.5 cloudx1_GetProcessChunkFrames() . . . 62
12.10.2.6 cloudx1_InitProcess() . . . 63
12.10.2.7 cloudx1_InitProcess2() . . . 63
12.10.2.8 cloudx1_InitProcess3() . . . 63
12.10.2.9 cloudx1_NewParameters() . . . 64
12.10.2.10 cloudx1_SetParameter() . . . 64
12.10.2.11 cloudx1_TerminateProcess() . . . 64
12.11 Sample using C interface . . . 65
12.11.1 Sample to create the processing instance . . . 65
12.11.2 Sample processing loop . . . 66
12.11.3 Sample to stop the processing instance . . . 66
12.12 Sample using C++ interface wrapper . . . 66
12.12.1 Setup and sanity checks . . . 67
12.12.2 Sample to create the processing instance . . . 67
12.12.3 Communicating with the instance . . . 67
12.12.4 Starting a web-server . . . 67
12.12.5 Sample processing loop . . . 67
12.13 Sample using C++ interface wrapper with dynamic library loading . . . 67
12.13.1 Dynamic loading . . . 67
12.13.2 Setup and sanity checks . . . 68
12.13.3 Sample to create the processing instance . . . 68
12.13.4 Communicating with the instance . . . 68
12.13.5 Starting a web-server . . . 68
12.13.6 Sample processing loop . . . 68
12.13.7 Custom preset manager . . . 68
12.13.8 Custom preset manager loading . . . 69

12.14 Sample using Generic C++ interface (multi-process compatible) . . . 69
12.14.1 Loading the library . . . 70
12.14.2 Setup and configure . . . 70
12.14.3 Create the processing instance . . . 70
12.14.4 Processing loop . . . 70
12.14.5 Starting a web-server . . . 70
12.14.6 Communicating with the instance . . . 71
12.14.7 Custom preset manager example . . . 71
12.14.8 Custom preset manager loading . . . 72
12.15 Update Management . . . 72
12.15.1 Detailed Description . . . 72
12.15.2 Function Documentation . . . 72
12.15.2.1 cloudx1_StartUpdateThread() . . . 72
12.15.2.2 cloudx1_StopUpdateThread() . . . 73
12.15.2.3 cloudx1_WaitUpdateThreadReady() . . . 73
12.16 Web server . . . 73
12.16.1 Detailed Description . . . 74
12.16.2 Macro Definition Documentation . . . 74
12.16.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 74
12.16.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 74
12.16.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 74
12.16.3 Function Documentation . . . 75
12.16.3.1 cloudx1_Webserver() . . . 75
12.16.3.2 cloudx1_Webserver_FreeString() . . . 75
12.16.3.3 cloudx1_Webserver_GetAppHealth() . . . 75
12.16.3.4 cloudx1_Webserver_SetAppHealth() . . . 76
12.16.3.5 cloudx1_Webserver_Status() . . . 76
12.16.3.6 cloudx1_Webserver_Stop() . . . 77
12.16.3.7 cloudx1_Webserver_tcp() . . . 77
12.16.3.8 cloudx1_Webserver_tcp2() . . . 77

13 Namespace Documentation **79**

13.1 sound4 Namespace Reference . . . 79
13.1.1 Detailed Description . . . 80
13.1.2 Typedef Documentation . . . 80
13.1.2.1 log_cb_t . . . 80
13.1.3 Enumeration Type Documentation . . . 80
13.1.3.1 LogSeverity . . . 80
13.1.3.2 PresetChange_Kind . . . 81
13.1.3.3 SampleFormat . . . 81
13.1.4 Function Documentation . . . 81
13.1.4.1 _log_cb_c() . . . 81

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

13.3.3.1 _log_cb_c() . . . 91
13.3.3.2 GetAudioInputCount() . . . 91
13.3.3.3 GetAudioOutputCount() . . . 92
13.3.3.4 GetBytesFromFormat() . . . 92
13.3.3.5 GetChannelCount() . . . 93
13.3.3.6 GetChunkSizeInFrames() . . . 93
13.3.3.7 GetFormatFromName() . . . 93
13.3.3.8 GetFormatName() . . . 94
13.3.3.9 GetPossibleChunkSizeInFrames() . . . 94
13.3.3.10 GetSampleRate() . . . 95
13.3.3.11 GetVersion() . . . 95
13.3.3.12 SanityCheck() . . . 95
13.3.3.13 SetLoggerCallback() . . . 97
13.3.3.14 SetLogSeverity() . . . 97
13.3.4 Variable Documentation . . . 97
13.3.4.1 _log_cb . . . 97
13.3.4.2 ChunkFrames . . . 97
13.3.4.3 InputChannels . . . 98
13.3.4.4 InputSampleSize . . . 98
13.3.4.5 OutputChannels . . . 98
13.3.4.6 OutputSampleSize . . . 98
13.3.4.7 process_name . . . 98
13.3.4.8 process_shortname . . . 99
13.3.4.9 SampleRate . . . 99
13.4 sound4::cloudx1::dyn::helper Namespace Reference . . . 99
13.4.1 Detailed Description . . . 100
13.4.2 Function Documentation . . . 100
13.4.2.1 AudioConvertFrom() . . . 100
13.4.2.2 AudioConvertTo() . . . 100
13.4.2.3 AudioMonoFromLiveStereo() . . . 101
13.4.2.4 AudioMonoToLiveStereo() . . . 101
13.4.2.5 MonoToStereo() . . . 101
13.4.2.6 MonoToStereo_Planar() . . . 102
13.4.2.7 StereoToMono() . . . 102
13.4.2.8 StereoToMono_Planar() . . . 103
13.4.2.9 UTF8ToWString() . . . 103
13.4.2.10 WStringToUTF8() . . . 104
13.5 sound4::cloudx1::helper Namespace Reference . . . 104
13.5.1 Detailed Description . . . 104
13.5.2 Function Documentation . . . 105
13.5.2.1 AudioConvertFrom() . . . 105
13.5.2.2 AudioConvertTo() . . . 105

13.5.2.3 AudioMonoFromLiveStereo() . . . 106
13.5.2.4 AudioMonoToLiveStereo() . . . 106
13.5.2.5 MonoToStereo() . . . 106
13.5.2.6 MonoToStereo_Planar() . . . 107
13.5.2.7 StereoToMono() . . . 107
13.5.2.8 StereoToMono_Planar() . . . 108
13.5.2.9 UTF8ToWString() . . . 108
13.5.2.10 WStringToUTF8() . . . 108
13.6 sound4::helper Namespace Reference . . . 109
13.6.1 Detailed Description . . . 109
13.6.2 Function Documentation . . . 109
13.6.2.1 UTF8ToWString() . . . 109
13.6.2.2 WStringToUTF8() . . . 109

14 Class Documentation **111**

14.1 sound4::CBus Class Reference . . . 111
14.1.1 Detailed Description . . . 111
14.1.2 Constructor & Destructor Documentation . . . 111
14.1.2.1 CBus() . . . 111
14.1.2.2 ~CBus() . . . 112
14.1.3 Member Function Documentation . . . 112
14.1.3.1 Get() . . . 112
14.1.4 Friends And Related Symbol Documentation . . . 112
14.1.4.1 CInstance . . . 112
14.2 sound4::CInstance::CClient Class Reference . . . 112
14.2.1 Detailed Description . . . 112
14.2.2 Constructor & Destructor Documentation . . . 113
14.2.2.1 CClient() [1/2] . . . 113
14.2.2.2 CClient() [2/2] . . . 113
14.2.2.3 ~CClient() . . . 113
14.2.3 Member Function Documentation . . . 113
14.2.3.1 operator=() . . . 113
14.2.3.2 ProcessJson() . . . 113
14.3 sound4::cloudx1::CInstance::CClient Class Reference . . . 114
14.3.1 Detailed Description . . . 114
14.3.2 Constructor & Destructor Documentation . . . 114
14.3.2.1 CClient() [1/2] . . . 114
14.3.2.2 CClient() [2/2] . . . 114
14.3.2.3 ~CClient() . . . 114
14.3.3 Member Function Documentation . . . 114
14.3.3.1 operator=() . . . 114
14.3.3.2 ProcessJson() . . . 114

14.4 sound4::cloudx1::dyn::CInstance::CClient Class Reference . . . 115
14.4.1 Detailed Description . . . 115
14.4.2 Constructor & Destructor Documentation . . . 115
14.4.2.1 CClient() [1/2] . . . 115
14.4.2.2 CClient() [2/2] . . . 116
14.4.2.3 ~CClient() . . . 116
14.4.3 Member Function Documentation . . . 116
14.4.3.1 operator=() . . . 116
14.4.3.2 ProcessJson() . . . 116
14.5 sound4::cloudx1::dyn::CDynLib Class Reference . . . 116
14.5.1 Detailed Description . . . 118
14.5.2 Constructor & Destructor Documentation . . . 118
14.5.2.1 CDynLib() . . . 118
14.5.3 Member Function Documentation . . . 118
14.5.3.1 IsOk() . . . 118
14.5.3.2 Load() . . . 118
14.5.4 Member Data Documentation . . . 119
14.5.4.1 AddAudio . . . 119
14.5.4.2 AddPadAudio . . . 119
14.5.4.3 AudioConvertFrom . . . 119
14.5.4.4 AudioConvertTo . . . 119
14.5.4.5 AudioMonoFromLiveStereo . . . 119
14.5.4.6 AudioMonoToLiveStereo . . . 120
14.5.4.7 DeleteClient . . . 120
14.5.4.8 ExitProcess . . . 120
14.5.4.9 FreeJsonAnswer . . . 120
14.5.4.10 FreeMetadataInfos . . . 120
14.5.4.11 FreeParameters . . . 120
14.5.4.12 FreeParameterValue . . . 120
14.5.4.13 GetAudio . . . 121
14.5.4.14 GetAudioInputCount . . . 121
14.5.4.15 GetAudioOutputCount . . . 121
14.5.4.16 GetBufferIn . . . 121
14.5.4.17 GetBufferOut . . . 121
14.5.4.18 GetBytesFromFormat . . . 121
14.5.4.19 GetChannelCount . . . 121
14.5.4.20 GetChunkSizeInFrames . . . 122
14.5.4.21 GetEstimatedDelay . . . 122
14.5.4.22 GetFormatFromName . . . 122
14.5.4.23 GetFormatName . . . 122
14.5.4.24 GetMaxPacketFrame . . . 122
14.5.4.25 GetMetadataInfos . . . 122

14.5.4.26 GetOutputCount . . . 122
14.5.4.27 GetParameter . . . 123
14.5.4.28 GetPossibleChunkSizeInFrames . . . 123
14.5.4.29 GetProcessChunkFrames . . . 123
14.5.4.30 GetSampleRate . . . 123
14.5.4.31 GetVersion . . . 123
14.5.4.32 InitProcess . . . 123
14.5.4.33 InitProcess2 . . . 123
14.5.4.34 InitProcess3 . . . 124
14.5.4.35 MonoToStereo . . . 124
14.5.4.36 MonoToStereo_Planar . . . 124
14.5.4.37 NewClient . . . 124
14.5.4.38 NewParameters . . . 124
14.5.4.39 PresetManager_InformChange . . . 124
14.5.4.40 ProcessAudio . . . 124
14.5.4.41 ProcessAudio_Planar . . . 125
14.5.4.42 ProcessJson . . . 125
14.5.4.43 SaveState . . . 125
14.5.4.44 SetLoggerCallback . . . 125
14.5.4.45 SetLogSeverity . . . 125
14.5.4.46 SetMetadata . . . 125
14.5.4.47 SetMetadataMulti . . . 125
14.5.4.48 SetParameter . . . 126
14.5.4.49 SetPresetManager . . . 126
14.5.4.50 StartUpdateThread . . . 126
14.5.4.51 StereoToMono . . . 126
14.5.4.52 StereoToMono_Planar . . . 126
14.5.4.53 StopUpdateThread . . . 126
14.5.4.54 TerminateProcess . . . 126
14.5.4.55 WaitUpdateThreadReady . . . 127
14.5.4.56 Webserver . . . 127
14.5.4.57 Webserver_FreeString . . . 127
14.5.4.58 Webserver_GetAppHealth . . . 127
14.5.4.59 Webserver_SetAppHealth . . . 127
14.5.4.60 Webserver_Status . . . 127
14.5.4.61 Webserver_Stop . . . 127
14.5.4.62 Webserver_tcp . . . 128
14.5.4.63 Webserver_tcp2 . . . 128
14.6 sound4::cloudx1::dyn::helper::CDynLoader Class Reference . . . 128
14.6.1 Detailed Description . . . 128
14.6.2 Constructor & Destructor Documentation . . . 129
14.6.2.1 CDynLoader() [1/3] . . . 129

14.6.2.2 CDynLoader() [2/3] . . . 129
14.6.2.3 CDynLoader() [3/3] . . . 129
14.6.2.4 ~CDynLoader() . . . 129
14.6.3 Member Function Documentation . . . 129
14.6.3.1 Close() . . . 129
14.6.3.2 GetSymbol() . . . 129
14.6.3.3 GetSymbol_safe() . . . 129
14.6.3.4 GetThisLibraryPath() . . . 130
14.6.3.5 IsOk() . . . 130
14.6.3.6 Load() . . . 130
14.6.3.7 operator bool() . . . 130
14.6.3.8 operator=() . . . 130
14.7 sound4::helper::CDynLoader Class Reference . . . 130
14.7.1 Detailed Description . . . 131
14.7.2 Constructor & Destructor Documentation . . . 131
14.7.2.1 CDynLoader() [1/3] . . . 131
14.7.2.2 CDynLoader() [2/3] . . . 131
14.7.2.3 CDynLoader() [3/3] . . . 131
14.7.2.4 ~CDynLoader() . . . 131
14.7.3 Member Function Documentation . . . 132
14.7.3.1 Close() . . . 132
14.7.3.2 GetSymbol() . . . 132
14.7.3.3 GetSymbol_safe() . . . 132
14.7.3.4 GetThisLibraryPath() . . . 132
14.7.3.5 IsOk() . . . 132
14.7.3.6 Load() . . . 132
14.7.3.7 operator bool() . . . 133
14.7.3.8 operator=() . . . 133
14.8 sound4::CInstance Class Reference . . . 133
14.8.1 Detailed Description . . . 134
14.8.2 Constructor & Destructor Documentation . . . 134
14.8.2.1 CInstance() [1/3] . . . 134
14.8.2.2 CInstance() [2/3] . . . 134
14.8.2.3 CInstance() [3/3] . . . 135
14.8.2.4 ~CInstance() . . . 135
14.8.3 Member Function Documentation . . . 135
14.8.3.1 AddAudio() . . . 135
14.8.3.2 Create() . . . 135
14.8.3.3 GetAudio() . . . 137
14.8.3.4 GetBufferIn() . . . 137
14.8.3.5 GetBufferOut() . . . 138
14.8.3.6 GetChunkFrames() . . . 138

14.8.3.7 GetEstimatedDelay() . . . 138
14.8.3.8 GetMetadataInfos() . . . 139
14.8.3.9 GetParam() . . . 139
14.8.3.10 GetWebServerAppHealth() . . . 139
14.8.3.11 IsOk() . . . 139
14.8.3.12 NewClient() . . . 140
14.8.3.13 OnUpdateThreadStart() . . . 140
14.8.3.14 OnUpdateThreadStop() . . . 140
14.8.3.15 operator=() . . . 140
14.8.3.16 PresetManager_InformChange() . . . 140
14.8.3.17 ProcessAnyAudio() . . . 141
14.8.3.18 ProcessAudio() . . . 141
14.8.3.19 ProcessAudio_Planar() . . . 141
14.8.3.20 SaveState() . . . 142
14.8.3.21 SetBus() . . . 142
14.8.3.22 SetMetadata() . . . 142
14.8.3.23 SetMetadataMulti() . . . 143
14.8.3.24 SetParam() . . . 143
14.8.3.25 SetPresetManager() . . . 144
14.8.3.26 SetWebServerAppHealth() . . . 144
14.8.3.27 StartWebServer() . . . 144
14.8.3.28 Stop() . . . 145
14.8.3.29 StopWebServer() . . . 145
14.8.3.30 Terminate() . . . 145
14.9 sound4::cloudx1::CInstance Class Reference . . . 146
14.9.1 Detailed Description . . . 147
14.9.2 Constructor & Destructor Documentation . . . 147
14.9.2.1 CInstance() [1/2] . . . 147
14.9.2.2 CInstance() [2/2] . . . 147
14.9.2.3 ~CInstance() . . . 147
14.9.3 Member Function Documentation . . . 147
14.9.3.1 AddAudio() . . . 147
14.9.3.2 Create() . . . 148
14.9.3.3 GetAudio() . . . 148
14.9.3.4 GetBufferIn() . . . 149
14.9.3.5 GetBufferOut() . . . 149
14.9.3.6 GetChunkFrames() . . . 149
14.9.3.7 GetEstimatedDelay() . . . 150
14.9.3.8 GetMetadataInfos() . . . 150
14.9.3.9 GetParam() . . . 150
14.9.3.10 GetWebServerAppHealth() . . . 151
14.9.3.11 IsOk() . . . 151

14.9.3.12 NewClient() . . . 151
14.9.3.13 OnUpdateThreadStart() . . . 151
14.9.3.14 OnUpdateThreadStop() . . . 151
14.9.3.15 operator=() . . . 151
14.9.3.16 PresetManager_InformChange() . . . 151
14.9.3.17 ProcessAnyAudio() . . . 152
14.9.3.18 ProcessAudio() . . . 152
14.9.3.19 ProcessAudio_Planar() . . . 153
14.9.3.20 SaveState() . . . 154
14.9.3.21 SetMetadata() . . . 154
14.9.3.22 SetMetadataMulti() . . . 154
14.9.3.23 SetParam() . . . 155
14.9.3.24 SetPresetManager() . . . 155
14.9.3.25 SetWebServerAppHealth() . . . 156
14.9.3.26 StartWebServer() . . . 156
14.9.3.27 Stop() . . . 156
14.9.3.28 StopWebServer() . . . 157
14.9.3.29 Terminate() . . . 157
14.10 sound4::cloudx1::dyn::CInstance Class Reference . . . 157
14.10.1 Detailed Description . . . 158
14.10.2 Constructor & Destructor Documentation . . . 159
14.10.2.1 CInstance() [1/2] . . . 159
14.10.2.2 CInstance() [2/2] . . . 159
14.10.2.3 ~CInstance() . . . 159
14.10.3 Member Function Documentation . . . 159
14.10.3.1 AddAudio() . . . 159
14.10.3.2 Create() . . . 159
14.10.3.3 GetAudio() . . . 160
14.10.3.4 GetBufferIn() . . . 160
14.10.3.5 GetBufferOut() . . . 161
14.10.3.6 GetChunkFrames() . . . 161
14.10.3.7 GetEstimatedDelay() . . . 161
14.10.3.8 GetMetadataInfos() . . . 162
14.10.3.9 GetParam() . . . 162
14.10.3.10 GetWebServerAppHealth() . . . 162
14.10.3.11 IsOk() . . . 162
14.10.3.12 NewClient() . . . 163
14.10.3.13 OnUpdateThreadStart() . . . 163
14.10.3.14 OnUpdateThreadStop() . . . 163
14.10.3.15 operator=() . . . 163
14.10.3.16 PresetManager_InformChange() . . . 163
14.10.3.17 ProcessAnyAudio() . . . 164

14.10.3.18 ProcessAudio() . . . 164
14.10.3.19 ProcessAudio_Planar() . . . 164
14.10.3.20 SaveState() . . . 165
14.10.3.21 SetMetadata() . . . 165
14.10.3.22 SetMetadataMulti() . . . 165
14.10.3.23 SetParam() . . . 166
14.10.3.24 SetPresetManager() . . . 166
14.10.3.25 SetWebServerAppHealth() . . . 167
14.10.3.26 StartWebServer() . . . 167
14.10.3.27 Stop() . . . 167
14.10.3.28 StopWebServer() . . . 168
14.10.3.29 Terminate() . . . 168
14.11 cloudx1_CClientInstance Struct Reference . . . 168
14.11.1 Detailed Description . . . 168
14.12 cloudx1_CInstance Struct Reference . . . 169
14.12.1 Detailed Description . . . 169
14.13 cloudx1_CParameters Struct Reference . . . 169
14.13.1 Detailed Description . . . 169
14.14 sound4::cloudx1::CPresetLoader Class Reference . . . 169
14.14.1 Detailed Description . . . 170
14.14.2 Constructor & Destructor Documentation . . . 170
14.14.2.1 CPresetLoader() . . . 170
14.14.2.2 ~CPresetLoader() . . . 170
14.14.3 Member Function Documentation . . . 170
14.14.3.1 Exists() . . . 170
14.14.3.2 GetAll() . . . 170
14.14.3.3 IsReadOnly() . . . 170
14.14.3.4 Read() . . . 170
14.14.3.5 Remove() . . . 170
14.14.3.6 Rename() . . . 171
14.14.3.7 Write() . . . 171
14.15 sound4::cloudx1::dyn::CPresetLoader Class Reference . . . 171
14.15.1 Detailed Description . . . 171
14.15.2 Constructor & Destructor Documentation . . . 172
14.15.2.1 CPresetLoader() . . . 172
14.15.2.2 ~CPresetLoader() . . . 172
14.15.3 Member Function Documentation . . . 172
14.15.3.1 Exists() . . . 172
14.15.3.2 GetAll() . . . 172
14.15.3.3 IsReadOnly() . . . 172
14.15.3.4 Read() . . . 172
14.15.3.5 Remove() . . . 172

14.15.3.6 Rename() . . . 173
14.15.3.7 Write() . . . 173
14.16 sound4::CPresetLoader Class Reference . . . 173
14.16.1 Detailed Description . . . 173
14.16.2 Constructor & Destructor Documentation . . . 174
14.16.2.1 CPresetLoader() . . . 174
14.16.2.2 ∼CPresetLoader() . . . 174
14.16.3 Member Function Documentation . . . 174
14.16.3.1 Exists() . . . 174
14.16.3.2 GetAll() . . . 174
14.16.3.3 IsReadOnly() . . . 174
14.16.3.4 Read() . . . 174
14.16.3.5 Remove() . . . 174
14.16.3.6 Rename() . . . 175
14.16.3.7 Write() . . . 175
14.17 sound4::CProcessor Class Reference . . . 175
14.17.1 Detailed Description . . . 176
14.17.2 Constructor & Destructor Documentation . . . 176
14.17.2.1 CProcessor() . . . 176
14.17.3 Member Function Documentation . . . 176
14.17.3.1 GetAudioInputCount() . . . 176
14.17.3.2 GetAudioOutputCount() . . . 177
14.17.3.3 GetBytesFromFormat() . . . 177
14.17.3.4 GetChannelCount() . . . 178
14.17.3.5 GetChunkSizeInFrames() . . . 178
14.17.3.6 GetFormatFromName() . . . 178
14.17.3.7 GetFormatName() . . . 179
14.17.3.8 GetPossibleChunkSizeInFrames() . . . 179
14.17.3.9 GetPrefixSymbol() . . . 180
14.17.3.10 GetSampleRate() . . . 180
14.17.3.11 GetVersion() . . . 180
14.17.3.12 IsOk() . . . 180
14.17.3.13 Load() . . . 180
14.17.3.14 SetLoggerCallback() . . . 181
14.17.3.15 SetLogSeverity() . . . 181
14.17.4 Friends And Related Symbol Documentation . . . 181
14.17.4.1 CBus . . . 181
14.17.4.2 CInstance . . . 181
14.18 sound4::cloudx1::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 182
14.18.1 Detailed Description . . . 182
14.18.2 Constructor & Destructor Documentation . . . 182
14.18.2.1 DynFuncHolder() [1/2] . . . 182

14.18.2.2 DynFuncHolder() [2/2] . . . 182
14.18.3 Member Function Documentation . . . 182
14.18.3.1 IsOk() . . . 182
14.18.3.2 operator bool() . . . 183
14.18.3.3 operator T∗() . . . 183
14.19 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 183
14.19.1 Detailed Description . . . 183
14.19.2 Constructor & Destructor Documentation . . . 184
14.19.2.1 DynFuncHolder() [1/2] . . . 184
14.19.2.2 DynFuncHolder() [2/2] . . . 184
14.19.3 Member Function Documentation . . . 184
14.19.3.1 IsOk() . . . 184
14.19.3.2 operator bool() . . . 184
14.19.3.3 operator T∗() . . . 184
14.20 MyPresetManager Class Reference . . . 185
14.20.1 Detailed Description . . . 185
14.20.2 Constructor & Destructor Documentation . . . 186
14.20.2.1 MyPresetManager() [1/2] . . . 186
14.20.2.2 ∼MyPresetManager() [1/2] . . . 186
14.20.2.3 MyPresetManager() [2/2] . . . 186
14.20.2.4 ∼MyPresetManager() [2/2] . . . 186
14.20.3 Member Function Documentation . . . 186
14.20.3.1 Exists() [1/2] . . . 186
14.20.3.2 Exists() [2/2] . . . 186
14.20.3.3 GetAll() [1/2] . . . 186
14.20.3.4 GetAll() [2/2] . . . 187
14.20.3.5 IsReadOnly() [1/2] . . . 187
14.20.3.6 IsReadOnly() [2/2] . . . 187
14.20.3.7 Read() [1/2] . . . 187
14.20.3.8 Read() [2/2] . . . 187
14.20.3.9 Remove() [1/2] . . . 187
14.20.3.10 Remove() [2/2] . . . 188
14.20.3.11 Rename() [1/2] . . . 188
14.20.3.12 Rename() [2/2] . . . 188
14.20.3.13 SetupPath() [1/2] . . . 188
14.20.3.14 SetupPath() [2/2] . . . 188
14.20.3.15 Write() [1/2] . . . 188
14.20.3.16 Write() [2/2] . . . 189
14.21 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference . . . 189
14.21.1 Detailed Description . . . 189
14.22 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference . . . 189
14.22.1 Detailed Description . . . 190

14.23 sound4::helper::SampleFormat< T > Struct Template Reference . . . 191
14.23.1 Detailed Description . . . 191
14.24 SOUND4_ProcessInfo Struct Reference . . . 191
14.24.1 Detailed Description . . . 192
14.24.2 Member Data Documentation . . . 192
14.24.2.1 name . . . 192
14.24.2.2 prefix . . . 192
14.24.2.3 shortname . . . 192
14.24.2.4 struct_size . . . 192
14.24.2.5 version . . . 192

15 File Documentation **193**

15.1 doc_examples.h File Reference . . . 193
15.2 doc_examples.h . . . 193
15.3 doc_gencpp.h File Reference . . . 193
15.4 doc_gencpp.h . . . 193
15.5 doc_json.h File Reference . . . 193
15.6 doc_json.h . . . 193
15.7 doc_main.h File Reference . . . 194
15.8 doc_main.h . . . 194
15.9 doc_params.h File Reference . . . 194
15.10 doc_params.h . . . 194
15.11 sound4.x1.cloud-pipe-cxx.cpp File Reference . . . 194
15.11.1 Function Documentation . . . 194
15.11.1.1 GetEnv() . . . 194
15.11.1.2 main() . . . 195
15.11.1.3 MyLogger() . . . 195
15.11.1.4 ProcessLoop() . . . 195
15.11.1.5 Usage() . . . 195
15.12 sound4.x1.cloud-pipe-cxx.cpp . . . 196
15.13 sound4.x1.cloud-pipe-dyn.cpp File Reference . . . 198
15.13.1 Function Documentation . . . 198
15.13.1.1 GetEnv() . . . 198
15.13.1.2 main() . . . 199
15.13.1.3 MyLogger() . . . 199
15.13.1.4 ProcessLoop() . . . 199
15.13.1.5 Usage() . . . 199
15.14 sound4.x1.cloud-pipe-dyn.cpp . . . 200
15.15 sound4.x1.cloud-pipe.cpp File Reference . . . 203
15.15.1 Macro Definition Documentation . . . 203
15.15.1.1 CHANNEL_COUNT . . . 203
15.15.1.2 CHUNK_SIZE . . . 203

15.15.1.3 SAMPLE_RATE . . . 204
15.15.2 Function Documentation . . . 204
15.15.2.1 main() . . . 204
15.15.2.2 MyLogger() . . . 204
15.15.2.3 ProcessLoop() . . . 204
15.15.2.4 Usage() . . . 204
15.16 sound4.x1.cloud-pipe.cpp . . . 205
15.17 sound4cl-pipe.cpp File Reference . . . 208
15.17.1 Function Documentation . . . 208
15.17.1.1 GetEnv() . . . 208
15.17.1.2 main() . . . 209
15.17.1.3 ProcessLoop() . . . 209
15.18 sound4cl-pipe.cpp . . . 210
15.19 sound4.x1.cloud.h File Reference . . . 213
15.19.1 Macro Definition Documentation . . . 215
15.19.1.1 CLOUDX1_DECL . . . 215
15.19.1.2 CLOUDX1_HAS_CLOUDBUS . . . 216
15.19.1.3 CLOUDX1_HAS_WEBSERVER . . . 216
15.20 sound4.x1.cloud.h . . . 216
15.21 sound4.x1.cloud.hpp File Reference . . . 220
15.21.1 Function Documentation . . . 222
15.21.1.1 cloudx1_custom_exists() . . . 222
15.21.1.2 cloudx1_custom_getall() . . . 222
15.21.1.3 cloudx1_custom_getall_free() . . . 222
15.21.1.4 cloudx1_custom_reader() . . . 222
15.21.1.5 cloudx1_custom_reader_free() . . . 222
15.21.1.6 cloudx1_custom_remove() . . . 222
15.21.1.7 cloudx1_custom_rename() . . . 223
15.21.1.8 cloudx1_custom_writer() . . . 223
15.22 sound4.x1.cloud.hpp . . . 223
15.23 sound4.x1.cloud_dyn.hpp File Reference . . . 231
15.23.1 Detailed Description . . . 233
15.23.2 Macro Definition Documentation . . . 233
15.23.2.1 UNIXLIKE . . . 233
15.23.3 Function Documentation . . . 233
15.23.3.1 cloudx1_custom_exists() . . . 233
15.23.3.2 cloudx1_custom_getall() . . . 233
15.23.3.3 cloudx1_custom_getall_free() . . . 233
15.23.3.4 cloudx1_custom_reader() . . . 233
15.23.3.5 cloudx1_custom_reader_free() . . . 234
15.23.3.6 cloudx1_custom_remove() . . . 234
15.23.3.7 cloudx1_custom_rename() . . . 234

15.23.3.8 cloudx1_custom_writer() . . . 234
15.24 sound4.x1.cloud_dyn.hpp . . . 234
15.25 sound4cl.hpp File Reference . . . 248
15.25.1 Detailed Description . . . 249
15.25.2 Macro Definition Documentation . . . 249
15.25.2.1 UNIXLIKE . . . 249
15.26 sound4cl.hpp . . . 250
15.27 sound4cl_cdef.h File Reference . . . 263
15.27.1 Macro Definition Documentation . . . 263
15.27.1.1 SOUND4_INVALID_WEBSERVER_ID . . . 263
15.27.1.2 SOUND4_WEBSERVER_HTTP_OK . . . 264
15.27.1.3 SOUND4_WEBSERVER_HTTPS_OK . . . 264
15.27.2 Typedef Documentation . . . 264
15.27.2.1 fs_char . . . 264
15.28 sound4cl_cdef.h . . . 264

Index **267**

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
Metadata Management 59
Process creation/destruction 61
Sample using C interface 65
Sample using C++ interface wrapper 66
Sample using C++ interface wrapper with dynamic library loading 67
Sample using Generic C++ interface (multi-process compatible) 69
Update Management 72
Web server 73

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

sound4 79
sound4::cloudx1 82
sound4::cloudx1::dyn 90
sound4::cloudx1::dyn::helper 99
sound4::cloudx1::helper 104
sound4::helper 109

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

sound4::CBus . . . 111
sound4::CInstance::CClient . . . 112
sound4::cloudx1::CInstance::CClient . . . 114
sound4::cloudx1::dyn::CInstance::CClient . . . 115
sound4::cloudx1::dyn::CDynLib . . . 116
sound4::cloudx1::dyn::helper::CDynLoader . . . 128
sound4::helper::CDynLoader . . . 130
sound4::CInstance . . . 133
sound4::cloudx1::CInstance . . . 146
sound4::cloudx1::dyn::CInstance . . . 157
cloudx1_CClientInstance . . . 168
cloudx1_CInstance . . . 169
cloudx1_CParameters . . . 169
sound4::cloudx1::CPresetLoader . . . 169
sound4::cloudx1::dyn::CPresetLoader . . . 171
MyPresetManager . . . 185
sound4::CPresetLoader . . . 173
MyPresetManager . . . 185
sound4::CProcessor . . . 175
sound4::cloudx1::dyn::helper::DynFuncHolder< T > . . . 182
sound4::helper::DynFuncHolder< T > . . . 183
sound4::cloudx1::dyn::helper::SampleFormat< T > . . . 189
sound4::cloudx1::helper::SampleFormat< T > . . . 189
sound4::helper::SampleFormat< T > . . . 191
SOUND4_ProcessInfo . . . 191

Chapter 10

Class Index

10.1 Class List

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

sound4::CBus
Shared Bus . . . 111
sound4::CInstance::CClient . . . 112
sound4::cloudx1::CInstance::CClient . . . 114
sound4::cloudx1::dyn::CInstance::CClient . . . 115
sound4::cloudx1::dyn::CDynLib
Dynamic library interface . . . 116
sound4::cloudx1::dyn::helper::CDynLoader
Helper to load dynamic library . . . 128
sound4::helper::CDynLoader
Helper to load dynamic library . . . 130
sound4::CInstance
Instance handling class . . . 133
sound4::cloudx1::CInstance
Instance handling class . . . 146
sound4::cloudx1::dyn::CInstance
Instance handling class . . . 157
cloudx1_CClientInstance . . . 168
cloudx1_CInstance . . . 169
cloudx1_CParameters . . . 169
sound4::cloudx1::CPresetLoader
Custom preset handler helper . . . 169
sound4::cloudx1::dyn::CPresetLoader
Custom preset handler helper . . . 171
sound4::CPresetLoader
Custom preset handler helper . . . 173
sound4::CProcessor
Dynamic library interface . . . 175
sound4::cloudx1::dyn::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 182
sound4::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 183
MyPresetManager
[Custom preset manager] . . . 185
sound4::cloudx1::dyn::helper::SampleFormat< T >
Helper for Sample format types . . . 189

sound4::cloudx1::helper::SampleFormat< T >
Helper for Sample format types 189
sound4::helper::SampleFormat< T >
Helper for Sample format types 191
SOUND4_ProcessInfo 191

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

doc_examples.h . . . 193
doc_gencpp.h . . . 193
doc_json.h . . . 193
doc_main.h . . . 194
doc_params.h . . . 194
sound4.x1.cloud-pipe-cxx.cpp
C++ sample using C++ interface wrapper . . . 194
sound4.x1.cloud-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading . . . 198
sound4.x1.cloud-pipe.cpp
C++ sample using only C interface . . . 203
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided . . . 208
sound4.x1.cloud.h
C interface for library . . . 213
sound4.x1.cloud.hpp
C++ wrapper for library . . . 220
sound4.x1.cloud_dyn.hpp
C++ wrapper for dynamic loading library . . . 231
sound4cl.hpp
C++ wrapper for dynamic loading library . . . 248
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries . . . 263

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 610 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 623 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 682 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 691 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 647 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 673 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 635 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 660 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 774 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 474 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 483 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 491 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 447 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 455 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 500 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 510 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 465 of file sound4.x1.cloud.h.

12.5.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 435 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 543 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 54 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 60 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 42 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 48 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 66 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 181 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 162 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 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)

12.9.1 Detailed Description

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

12.9.2 Function Documentation

12.9.2.1 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

12.9.2.2 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

12.9.2.3 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.

12.9.2.4 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.

12.10 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)

12.10.1 Detailed Description

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

12.10.2 Function Documentation

12.10.2.1 cloudx1_ExitProcess()

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

Quit and destroy the instance

Parameters

instance	the instance to destroy

12.10.2.2 cloudx1_FreeParameters()

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

Free the parameters created with cloudx1_NewParameters.

12.10.2.3 cloudx1_FreeParameterValue()

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

Free a parameter value returned by cloudx1_GetParameter

Parameters

value	the returned value

12.10.2.4 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

12.10.2.5 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

12.10.2.6 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

12.10.2.7 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

12.10.2.8 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

12.10.2.9 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).

12.10.2.10 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

12.10.2.11 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

12.11 Sample using C interface

C example.

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

12.11.1 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);
     }*/
```

12.11.2 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;
        }
    }
```

12.11.3 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);
```

12.12 Sample using C++ interface wrapper

C++ example (static link)

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

12.12.1 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");
```

12.12.2 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;
}
```

12.12.3 Communicating with the instance

See also

jsonsyntax

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

12.12.4 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;
}
```

12.12.5 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;
    }
}
```

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

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

12.13.1 Dynamic loading

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

12.13.2 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");
```

12.13.3 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;
}
```

12.13.4 Communicating with the instance

See also

jsonsyntax

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

12.13.5 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;
}
```

12.13.6 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;
    }
}
```

12.13.7 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;
        }
};
```

12.13.8 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);
    }
```

12.14 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

12.14.1 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;
}
```

12.14.2 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");
```

12.14.3 Create the processing instance

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

12.14.4 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;
    }
}
```

12.14.5 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 Communicating with the instance

See also

jsonsyntax

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

12.14.7 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;
        }
};
```

12.14.8 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 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)

12.15.1 Detailed Description

Manage the process update and communications.

12.15.2 Function Documentation

12.15.2.1 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

12.15.2.2 cloudx1_StopUpdateThread()

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

Stop the update thread

Parameters

instance	the processing instance

12.15.2.3 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

12.16 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)

12.16.1 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.

12.16.2 Macro Definition Documentation

12.16.2.1 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 1158 of file sound4.x1.cloud.h.

12.16.2.2 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 1163 of file sound4.x1.cloud.h.

12.16.2.3 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 1167 of file sound4.x1.cloud.h.

12.16.3 Function Documentation

12.16.3.1 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

12.16.3.2 cloudx1_Webserver_FreeString()

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

Free a a string returned by cloudx1_Webserver_GetAppHealth

Parameters

str	the string to free

12.16.3.3 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)

12.16.3.4 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

12.16.3.5 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

12.16.3.6 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

12.16.3.7 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

12.16.3.8 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 281 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 265 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 863 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 57 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 288 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 286 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 395 of file sound4.x1.cloud.hpp.

13.2.2.2 LogSeverity

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

Log Severity levels

Definition at line 391 of file sound4.x1.cloud.hpp.

13.2.2.3 PresetChange_Kind

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

Definition at line 461 of file sound4.x1.cloud.hpp.

13.2.2.4 SampleFormat

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

Definition at line 53 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 402 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 301 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 313 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 383 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 289 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 262 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 372 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 361 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 275 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 325 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 247 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 335 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 416 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 409 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 400 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 58 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 63 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 73 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 68 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 78 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 51 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 52 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 83 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 842 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 838 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 916 of file sound4.x1.cloud_dyn.hpp.

13.3.2.4 SampleFormat

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

Definition at line 476 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 849 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 741 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 754 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 830 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 728 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 695 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 818 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 806 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 709 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 767 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 679 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 779 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 863 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 856 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 847 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 481 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 486 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 496 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 491 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 501 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 474 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 475 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 506 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 572 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 587 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 656 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 667 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 615 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 645 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 601 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 630 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 530 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 518 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 148 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 162 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 226 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 236 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 188 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 216 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 175 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 202 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 107 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 95 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 95 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 83 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 821 of file sound4cl.hpp.

14.1.2 Constructor & Destructor Documentation

14.1.2.1 CBus()

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

Definition at line 823 of file sound4cl.hpp.

14.1.2.2 ~CBus()

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

Definition at line 830 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 839 of file sound4cl.hpp.

14.1.4 Friends And Related Symbol Documentation

14.1.4.1 CInstance

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

Definition at line 838 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 1393 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 1395 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 1405 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 1419 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 944 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 946 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 955 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 969 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 1423 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 1425 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 1435 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 1449 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

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 220 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 466 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 322 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 257 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 258 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 261 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 262 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 265 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 266 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 268 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 244 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 270 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 287 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 237 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 240 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 260 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 231 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 232 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 250 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 251 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 255 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 230 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 229 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 252 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 254 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 253 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 256 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 286 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 259 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 239 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 294 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 295 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 233 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 228 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 241 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 242 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 296 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 264 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 282 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 267 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 236 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 291 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 248 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 249 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 269 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 271 of file sound4.x1.cloud_dyn.hpp.

14.5.4.44 SetLoggerCallback

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

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

14.5.4.45 SetLogSeverity

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

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

14.5.4.46 SetMetadata

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

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

14.5.4.47 SetMetadataMulti

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

Definition at line 306 of file sound4.x1.cloud_dyn.hpp.

14.5.4.48 SetParameter

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

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

14.5.4.49 SetPresetManager

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

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

14.5.4.50 StartUpdateThread

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

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

14.5.4.51 StereoToMono

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

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

14.5.4.52 StereoToMono_Planar

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

Definition at line 281 of file sound4.x1.cloud_dyn.hpp.

14.5.4.53 StopUpdateThread

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

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

14.5.4.54 TerminateProcess

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

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

14.5.4.55 WaitUpdateThreadReady

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

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

14.5.4.56 Webserver

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

Definition at line 275 of file sound4.x1.cloud_dyn.hpp.

14.5.4.57 Webserver_FreeString

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

Definition at line 312 of file sound4.x1.cloud_dyn.hpp.

14.5.4.58 Webserver_GetAppHealth

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

Definition at line 311 of file sound4.x1.cloud_dyn.hpp.

14.5.4.59 Webserver_SetAppHealth

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

Definition at line 310 of file sound4.x1.cloud_dyn.hpp.

14.5.4.60 Webserver_Status

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

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

14.5.4.61 Webserver_Stop

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

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

14.5.4.62 Webserver_tcp

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

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

14.5.4.63 Webserver_tcp2

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

Definition at line 274 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_LIBRAR
- 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 105 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 208 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 169 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 182 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 190 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 194 of file sound4.x1.cloud_dyn.hpp.

14.6.3.5 IsOk()

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

Definition at line 123 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 163 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 120 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 152 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 255 of file sound4cl.hpp.

14.7.3 Member Function Documentation

14.7.3.1 Close()

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

Definition at line 216 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 229 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 237 of file sound4cl.hpp.

14.7.3.4 GetThisLibraryPath()

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

Definition at line 241 of file sound4cl.hpp.

14.7.3.5 IsOk()

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

Definition at line 170 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 210 of file sound4cl.hpp.

14.7.3.7 operator bool()

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

Definition at line 167 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 ()

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 873 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 880 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 895 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 907 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 1469 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 1011 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 1475 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 1310 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 1327 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 1057 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 1292 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 1145 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 936 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 1238 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 915 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 1443 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 1169 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 1173 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 1071 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 1369 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 1335 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 1351 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 1459 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 953 of file sound4cl.hpp.

14.8.3.22 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 1089 of file sound4cl.hpp.

14.8.3.23 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 1109 of file sound4cl.hpp.

14.8.3.24 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 925 of file sound4cl.hpp.

14.8.3.25 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 970 of file sound4cl.hpp.

14.8.3.26 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 1229 of file sound4cl.hpp.

14.8.3.27 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 1189 of file sound4cl.hpp.

14.8.3.28 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 1269 of file sound4cl.hpp.

14.8.3.29 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 1217 of file sound4cl.hpp.

14.8.3.30 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 1258 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 ()

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 468 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 473 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 484 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 1018 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 587 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 1024 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 859 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 877 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 624 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 841 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 702 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 513 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 788 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 492 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 992 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 724 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 728 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 638 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 920 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 886 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 902 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 1008 of file sound4.x1.cloud.hpp.

14.9.3.21 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 654 of file sound4.x1.cloud.hpp.

14.9.3.22 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 672 of file sound4.x1.cloud.hpp.

14.9.3.23 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 502 of file sound4.x1.cloud.hpp.

14.9.3.24 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 547 of file sound4.x1.cloud.hpp.

14.9.3.25 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 780 of file sound4.x1.cloud.hpp.

14.9.3.26 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 744 of file sound4.x1.cloud.hpp.

14.9.3.27 Stop()

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

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

Definition at line 818 of file sound4.x1.cloud.hpp.

14.9.3.28 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 769 of file sound4.x1.cloud.hpp.

14.9.3.29 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 809 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 ()

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 923 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 928 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 940 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 1499 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 1045 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 1505 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 1338 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 1356 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 1088 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 1320 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 1177 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 969 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 1266 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 948 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 1473 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 1201 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 1205 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 1102 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 1399 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 1365 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 1381 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 1489 of file sound4.x1.cloud_dyn.hpp.

14.10.3.21 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 1120 of file sound4.x1.cloud_dyn.hpp.

14.10.3.22 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 1141 of file sound4.x1.cloud_dyn.hpp.

14.10.3.23 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 958 of file sound4.x1.cloud_dyn.hpp.

14.10.3.24 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 1004 of file sound4.x1.cloud_dyn.hpp.

14.10.3.25 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 1257 of file sound4.x1.cloud_dyn.hpp.

14.10.3.26 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 1221 of file sound4.x1.cloud_dyn.hpp.

14.10.3.27 Stop()

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

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

Definition at line 1297 of file sound4.x1.cloud_dyn.hpp.

14.10.3.28 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 1246 of file sound4.x1.cloud_dyn.hpp.

14.10.3.29 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 1288 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 448 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:

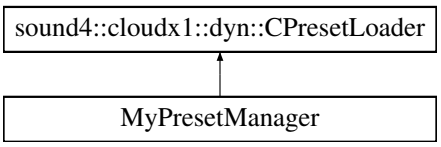
- sound4.x1.cloud.hpp

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 903 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:

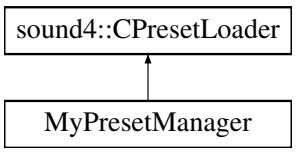
- sound4.x1.cloud_dyn.hpp

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 850 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 299 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 753 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 763 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 800 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 743 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 714 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 791 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 782 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 725 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 393 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 773 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 701 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 588 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 408 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 812 of file sound4cl.hpp.

14.17.3.15 SetLogSeverity()

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

Sets the log severity (global)

Definition at line 806 of file sound4cl.hpp.

14.17.4 Friends And Related Symbol Documentation

14.17.4.1 CBus

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

Definition at line 305 of file sound4cl.hpp.

14.17.4.2 CInstance

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

Definition at line 306 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 84 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 95 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 97 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 96 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 98 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 131 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 142 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 144 of file sound4cl.hpp.

14.19.3.2 operator bool()

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

Definition at line 143 of file sound4cl.hpp.

14.19.3.3 operator T*()

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

Definition at line 145 of file sound4cl.hpp.

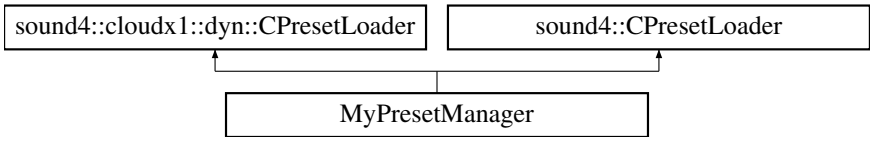
The documentation for this class was generated from the following file:

- 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 545 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 122 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 110 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 73 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 75 of file sound4.x1.cloud.h.

14.24.2.2 prefix

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

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

14.24.2.3 shortname

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

Definition at line 77 of file sound4.x1.cloud.h.

14.24.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

Definition at line 74 of file sound4.x1.cloud.h.

14.24.2.5 version

```
const char* SOUND4_ProcessInfo::version
```

Definition at line 78 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 <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 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_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 22 of file sound4.x1.cloud.h.

15.19.1.3 CLOUDX1_HAS_WEBSERVER

```
#define CLOUDX1_HAS_WEBSERVER 1
```

Definition at line 21 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 #include <stdint.h>
00007 #ifdef __unix__
00008     #include <sys/types.h>
00009     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00010 #elif defined(_WIN32) || defined(WIN32)
00011     #ifdef BUILDING_CLOUDX1_DLL
00012         #define CLOUDX1_DECL    __declspec(dllexport)
00013     #else
00014         #define CLOUDX1_DECL    __declspec(dllimport)
00015     #endif
00016 #elif defined(__APPLE__)
00017     #include <sys/types.h>
00018     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00019 #endif
00020
00021 #define CLOUDX1_HAS_WEBSERVER 1
00022 #define CLOUDX1_HAS_CLOUDBUS 0
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif // __cplusplus
00027
00028 // ----------------------------------------------------------------
00042 #define CLOUDX1_AUDIOFRAME_COUNT      64
00048 #define CLOUDX1_CHANNEL_COUNT          2
00054 #define CLOUDX1_AUDIO_INPUT_COUNT      1
00060 #define CLOUDX1_AUDIO_OUTPUT_COUNT     1
00066 #define CLOUDX1_SAMPLE_RATE            48000
00067
00073 struct SOUND4_ProcessInfo {
00074     size_t struct_size;
00075     const char *name;
00076     const char *prefix;
00077     const char *shortname;
00078     const char *version;
00079 };
00080
00088 CLOUDX1_DECL const struct SOUND4_ProcessInfo* SOUND4_GetProcessInfo();
00089
00095 CLOUDX1_DECL const char *cloudx1_GetVersion();
00096
00109 CLOUDX1_DECL unsigned int cloudx1_GetChunkSizeInFrames();
00110
00119 CLOUDX1_DECL unsigned int* cloudx1_GetPossibleChunkSizeInFrames();
00120
00128 CLOUDX1_DECL unsigned int cloudx1_GetChannelCount();
00129
00138 CLOUDX1_DECL unsigned int cloudx1_GetAudioInputCount();
00139
00148 CLOUDX1_DECL unsigned int cloudx1_GetAudioOutputCount();
00149
00157 CLOUDX1_DECL unsigned int cloudx1_GetSampleRate();
00158
00162 enum cloudx1_LogSeverity {
00163     none     = 0,
```

```
00164     fatal    = 1,
00165     error    = 2,
00166     warning  = 3,
00167     info     = 4,
00168     verbose  = 5,
00169     verbose2 = 6,
00170     verbose3 = 7,
00171     verbose4 = 8,
00172     verbose5 = 9,
00173 };
00174
00181 typedef void (*cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *);
00182
00192 CLOUDX1_DECL void cloudx1_SetLoggerCallback( cloudx1_loggerfn logger );
00193
00201 CLOUDX1_DECL void cloudx1_SetLogSeverity( enum cloudx1_LogSeverity severity );
00202
00205 // ------------------------------------------------------------
00213 // ------------------------------------------------------------
00217 struct cloudx1_CInstance;
00218
00222 struct cloudx1_CParameters;
00223
00231 CLOUDX1_DECL struct cloudx1_CParameters *cloudx1_NewParameters();
00232
00236 CLOUDX1_DECL void cloudx1_FreeParameters(struct cloudx1_CParameters *params);
00237
00249 CLOUDX1_DECL void cloudx1_SetParameter(struct cloudx1_CParameters *params, const char *name, const
      char *value);
00250
00256 CLOUDX1_DECL const char *cloudx1_GetParameter(struct cloudx1_CParameters *params, const char *name);
00257
00263 CLOUDX1_DECL void cloudx1_FreeParameterValue(const char *value);
00264
00272 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess(
00273         const char *LoginKey,
00274         const char *RadioName,
00275         const char *Access_Key_ID,
00276         const char *Access_Key_Secret,
00277         const char *save_path
00278     );
00279
00287 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess2(
00288         const char *LoginKey,
00289         const char *RadioName,
00290         const char *Access_Key_ID,
00291         const char *Access_Key_Secret,
00292         const char *save_path,
00293         const struct cloudx1_CParameters *parameters
00294     );
00295
00323 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess3(
00324         const char *LoginKey,
00325         const char *RadioName,
00326         const char *Access_Key_ID,
00327         const char *Access_Key_Secret,
00328         const char *save_path,
00329         const struct cloudx1_CParameters *parameters,
00330         unsigned int frames_per_chunk
00331     );
00332
00344 CLOUDX1_DECL unsigned int cloudx1_GetProcessChunkFrames(struct cloudx1_CInstance *instance);
00345
00355 CLOUDX1_DECL int cloudx1_TerminateProcess(struct cloudx1_CInstance *instance);
00356
00362 CLOUDX1_DECL void cloudx1_ExitProcess(struct cloudx1_CInstance *instance);
00363
00366 #if CLOUDX1_HAS_CLOUDBUS
00367 // ------------------------------------------------------------
00377 struct cloudx1_CBus;
00378
00384 CLOUDX1_DECL struct cloudx1_CBus* cloudx1_NewBus();
00385
00391 CLOUDX1_DECL void cloudx1_FreeBus(struct cloudx1_CBus *bus);
00392
00402 CLOUDX1_DECL void cloudx1_SetInstanceBus(struct cloudx1_CParameters *params, struct cloudx1_CBus
      *bus);
00403
00404
00406 #endif // CLOUDX1_HAS_CLOUDBUS
00407
00408 // ------------------------------------------------------------
00431 // ------------------------------------------------------------
00432
00433 // OS dependent filesystem path char type
00434 #ifdef _WIN32
00435     typedef wchar_t fs_char;
```

```
00436 #else // !_WIN32
00437     typedef char fs_char;
00438 #endif // !_WIN32
00439
00447 typedef char * (*cloudx1_storage_reader) (const fs_char *filename, void* userdata);
00448
00455 typedef void (*cloudx1_storage_reader_free) (char *content, void* userdata);
00456
00465 typedef int (*cloudx1_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00466
00474 typedef int (*cloudx1_storage_exists) (const fs_char *filename, void* userdata);
00475
00483 typedef fs_char** (*cloudx1_storage_getall) (void* userdata);
00484
00491 typedef void (*cloudx1_storage_getall_free) (fs_char**all, void* userdata);
00492
00500 typedef int (*cloudx1_storage_remove) (const fs_char *filename, void* userdata);
00501
00510 typedef int (*cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00511
00529 CLOUDX1_DECL void cloudx1_SetPresetManager(struct cloudx1_CParameters *params
00530     , cloudx1_storage_reader reader, cloudx1_storage_reader_free
00531     , cloudx1_storage_writer writer
00532     , cloudx1_storage_exists exists
00533     , cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free
00534     , cloudx1_storage_remove remove
00535     , cloudx1_storage_rename rename
00536     , int IsReadOnly
00537     , void* userdata
00538     );
00539
00543 enum cloudx1_PresetChange_Kind {
00544     change_kind_created = 1,
00545     change_kind_modified   ,
00546     change_kind_deleted    ,
00547 };
00548
00557 CLOUDX1_DECL void cloudx1_PresetManager_InformChange(struct cloudx1_CInstance *instance, const fs_char
      *relative_path, enum cloudx1_PresetChange_Kind change_kind);
00558
00561 // ------------------------------------------------------------------
00569 // ------------------------------------------------------------------
00570
00587 CLOUDX1_DECL void cloudx1_SetMetadata(struct cloudx1_CInstance *instance, const char* key, const char*
      value);
00588
00599 CLOUDX1_DECL void cloudx1_SetMetadataMulti(struct cloudx1_CInstance *instance, const char** keyvalue);
00600
00608 CLOUDX1_DECL const char** cloudx1_GetMetadataInfos(struct cloudx1_CInstance *instance);
00609
00616 CLOUDX1_DECL void cloudx1_FreeMetadataInfos(struct cloudx1_CInstance *instance, const char** infos);
00617
00620 // ------------------------------------------------------------------
00627 // ------------------------------------------------------------------
00628
00641 CLOUDX1_DECL void cloudx1_StartUpdateThread(struct cloudx1_CInstance *instance, unsigned int port);
00642
00648 CLOUDX1_DECL void cloudx1_StopUpdateThread(struct cloudx1_CInstance *instance);
00649
00657 CLOUDX1_DECL int cloudx1_WaitUpdateThreadReady(struct cloudx1_CInstance *instance, int milliseconds);
00658
00661 // ------------------------------------------------------------------
00668 // ------------------------------------------------------------------
00690 CLOUDX1_DECL void cloudx1_ProcessAudio(struct cloudx1_CInstance *instance, const float *input, float
      *output );
00691
00710 CLOUDX1_DECL void cloudx1_ProcessAudio_Planar(struct cloudx1_CInstance *instance
00711     , float const * const *input
00712     , float * const *output);
00713
00726 CLOUDX1_DECL float *cloudx1_GetBufferIn(struct cloudx1_CInstance *instance);
00727
00740 CLOUDX1_DECL float *cloudx1_GetBufferOut(struct cloudx1_CInstance *instance);
00741
00751 CLOUDX1_DECL unsigned int cloudx1_GetEstimatedDelay(struct cloudx1_CInstance *instance);
00752
00755 // ------------------------------------------------------------------
00764 // ------------------------------------------------------------------
00765     #ifdef _MSC_VER
00766         #define __BYTE_ORDER__  REG_DWORD
00767         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00768         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00769     #endif
00770
00774 enum cloudx1_SampleFormat {
00775     INVALID_FORMAT=0,
00776     S16_LE,
```

```
00777     S16_BE,
00778     S24_LE,
00779     S24_BE,
00780     S32_LE,
00781     S32_BE,
00782     F32_LE,
00783     F32_BE,
00784
00785     // Native definitions, only links to the physical ones
00786     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00787         S16_NATIVE=S16_LE,
00788         S24_NATIVE=S24_LE,
00789         S32_NATIVE=S32_LE,
00790         F32_NATIVE=F32_LE,
00791     #else
00792         S16_NATIVE=S16_BE,
00793         S24_NATIVE=S24_BE,
00794         S32_NATIVE=S32_BE,
00795         F32_NATIVE=F32_BE,
00796     #endif
00797 };
00798
00805 CLOUDX1_DECL const char *cloudx1_GetFormatName(const enum cloudx1_SampleFormat fmt);
00806
00813 CLOUDX1_DECL enum cloudx1_SampleFormat cloudx1_GetFormatFromName(const char *name);
00814
00821 CLOUDX1_DECL unsigned int cloudx1_GetBytesFromFormat(const enum cloudx1_SampleFormat fmt);
00822
00823
00830 CLOUDX1_DECL unsigned int cloudx1_GetMaxPacketFrame(struct cloudx1_CInstance *instance);
00831
00842 CLOUDX1_DECL unsigned int cloudx1_AddAudio(struct cloudx1_CInstance *instance, const uint8_t *payload,
      unsigned int nFrame, enum cloudx1_SampleFormat fmt );
00843
00850 CLOUDX1_DECL unsigned int cloudx1_AddPadAudio(struct cloudx1_CInstance *instance);
00851
00858 CLOUDX1_DECL unsigned int cloudx1_GetOutputCount(struct cloudx1_CInstance *instance);
00859
00869 CLOUDX1_DECL unsigned int cloudx1_GetAudio(struct cloudx1_CInstance *instance, uint8_t *payload,
      unsigned int max_nFrame, enum cloudx1_SampleFormat fmt );
00870
00874 // ------------------------------------------------------------------
00881 // ------------------------------------------------------------------
00892 CLOUDX1_DECL void cloudx1_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
00893
00904 CLOUDX1_DECL void cloudx1_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
00905
00915 CLOUDX1_DECL void cloudx1_StereoToMono(const float *input, float *output, size_t nFrame);
00916
00926 CLOUDX1_DECL void cloudx1_MonoToStereo(const float *input, float *output, size_t nFrame);
00927
00938 CLOUDX1_DECL void cloudx1_StereoToMono_Planar(const float *inputL, const float *inputR, float *output,
      size_t nFrame);
00939
00950 CLOUDX1_DECL void cloudx1_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame);
00951
00958 CLOUDX1_DECL void cloudx1_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00959
00966 CLOUDX1_DECL void cloudx1_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00967
00970 // ------------------------------------------------------------------
00977 // ------------------------------------------------------------------
00981 struct cloudx1_CClientInstance;
00982
00991 CLOUDX1_DECL struct cloudx1_CClientInstance *cloudx1_NewClient(struct cloudx1_CInstance *instance);
00992
00998 CLOUDX1_DECL void cloudx1_DeleteClient(struct cloudx1_CClientInstance *client);
00999
01013 CLOUDX1_DECL const char *cloudx1_ProcessJson(struct cloudx1_CClientInstance *client, const char
      *json_str, int *need_save);
01014
01020 CLOUDX1_DECL void cloudx1_FreeJsonAnswer(const char *json_str);
01021
01033 CLOUDX1_DECL int cloudx1_SaveState(struct cloudx1_CInstance *instance);
01034
01037 #if CLOUDX1_HAS_WEBSERVER
01038 // ------------------------------------------------------------------
01053 // ------------------------------------------------------------------
01061 CLOUDX1_DECL uint64_t cloudx1_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port);
01062
01063 // ------------------------------------------------------------------
01083 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);
```

```
01084
01102 CLOUDX1_DECL uint64_t cloudx1_Webserver(unsigned int listenport, unsigned int listenport_secure,
      struct cloudx1_CInstance *instance);
01103
01113 CLOUDX1_DECL int cloudx1_Webserver_Stop(uint64_t id, int timeout_ms);
01114
01123 CLOUDX1_DECL int cloudx1_Webserver_Status(uint64_t id);
01124
01135 CLOUDX1_DECL void cloudx1_Webserver_SetAppHealth(struct cloudx1_CInstance *instance, int httpcode,
      const char *contenttype, const char *content);
01136
01145 CLOUDX1_DECL void cloudx1_Webserver_GetAppHealth(struct cloudx1_CInstance *instance, int* httpcode,
      char** contenttype, char** content);
01146
01152 CLOUDX1_DECL void cloudx1_Webserver_FreeString(char *str);
01153
01158 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
01159
01163 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
01167 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
01168
01170 #endif // CLOUDX1_HAS_WEBSERVER
01171
01172 #ifdef __cplusplus
01173 } // extern "C"
01174 #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 1059 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 1064 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 1074 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 1045 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 1050 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 1081 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 1086 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 1054 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 #else // !_WIN32
00028     #define fs_strdup strdup
00029 #endif // !_WIN32
00030
00031 // bridge C callbacks to CPresetLoader
00032 extern "C" {
00033     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00034     static void cloudx1_custom_reader_free(char *content, void* userdata);
00035     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00036     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00037     static fs_char** cloudx1_custom_getall(void* userdata);
00038     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00039     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00040     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00041 };
00042
00046 namespace sound4 {
00050 namespace cloudx1 {
00051     static constexpr const char *process_name = "SOUND4 x1.CLOUD";
00052     static constexpr const char *process_shortname = "x1";
00053     using SampleFormat = cloudx1_SampleFormat;
00058     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00063     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00068     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00073     const size_t InputSampleSize=ChunkFrames*InputChannels;
00078     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00083     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
```

```
00084
00088     namespace helper {
00089         #ifdef _WIN32
00095         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00096             if (wstr.empty()) return {};
00097             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00098             std::string str(size, 0);
00099             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00100             return str;
00101         }
00107         static inline std::wstring UTF8ToWString(const std::string& str) {
00108             if (str.empty()) return std::wstring();
00109             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00110             std::wstring wstr(size, 0);
00111             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00112             return wstr;
00113         }
00114         #endif // _WIN32
00115
00121             template<typename T>
00122         struct SampleFormat {
00123         };
00124             template<>
00125         struct SampleFormat<int16_t> {
00126             const cloudx1_SampleFormat format = S16_NATIVE;
00127         };
00128             template<>
00129         struct SampleFormat<int32_t> {
00130             const cloudx1_SampleFormat format = S32_NATIVE;
00131         };
00132             template<>
00133         struct SampleFormat<float> {
00134             const cloudx1_SampleFormat format = F32_NATIVE;
00135         };
00136
00137         // ------------------------------------------------------------
00148         static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00149             cloudx1_AudioConvertFrom(payload, output, nSpl, fmt);
00150         }
00151
00162         static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00163             cloudx1_AudioConvertTo(input, payload, nSpl, fmt);
00164         }
00165
00175         static inline void StereoToMono(const float *input, float *output, size_t nFrame) {
00176             cloudx1_StereoToMono(input, output, nFrame);
00177         }
00178
00188         static inline void MonoToStereo(const float *input, float *output, size_t nFrame) {
00189             cloudx1_MonoToStereo(input, output, nFrame);
00190         }
00191
00202         static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float
      *output, size_t nFrame) {
00203             cloudx1_StereoToMono_Planar(inputL, inputR, output, nFrame);
00204         }
00205
00216         static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame) {
00217             cloudx1_MonoToStereo_Planar(input, outputL, outputR, nFrame);
00218         }
00219
00226         static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
00227             cloudx1_AudioMonoFromLiveStereo(payload, output);
00228         }
00229
00236         static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
00237             cloudx1_AudioMonoToLiveStereo(input, payload);
00238         }
00239
00240     }; // namespace helper
00241
00247     static inline std::string GetVersion()
00248         {
00249             return std::string(cloudx1_GetVersion());
00250         }
00251
00262     static inline unsigned int GetChunkSizeInFrames()
00263         {
00264             return cloudx1_GetChunkSizeInFrames();
00265         }
00266
00275     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00276         {
```

```
00277             std::vector<unsigned int> list;
00278             for (unsigned int* src=cloudx1_GetPossibleChunkSizeInFrames();*src;src++) {
00279                 list.push_back(*src);
00280             }
00281             return list;
00282         }
00283
00289     static inline unsigned int GetChannelCount()
00290         {
00291             return cloudx1_GetChannelCount();
00292         }
00293
00301     static inline unsigned int GetAudioInputCount()
00302         {
00303             return cloudx1_GetAudioInputCount();
00304         }
00305
00313     static inline unsigned int GetAudioOutputCount()
00314         {
00315             return cloudx1_GetAudioOutputCount();
00316         }
00317
00325     static inline unsigned int GetSampleRate()
00326         {
00327             return cloudx1_GetSampleRate();
00328         }
00329
00335     static inline void SanityCheck(bool a_bCheckFrames = true)
00336     {
00337         // Do some sanity checks
00338         if (cloudx1_GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00339             throw std::runtime_error("Bad library sampling rate");
00340         }
00341         if (a_bCheckFrames && cloudx1_GetChunkSizeInFrames()!=ChunkFrames) {
00342             throw std::runtime_error("Bad library frame size");
00343         }
00344         if (cloudx1_GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00345             throw std::runtime_error("Bad library channel count");
00346         }
00347         if (cloudx1_GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00348             throw std::runtime_error("Bad library input count");
00349         }
00350         if (cloudx1_GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00351             throw std::runtime_error("Bad library output count");
00352         }
00353     }
00354
00361     static inline std::string GetFormatName(const SampleFormat fmt)
00362         {
00363             return std::string(cloudx1_GetFormatName(fmt));
00364         }
00365
00372     static inline SampleFormat GetFormatFromName(const std::string& name)
00373         {
00374             return cloudx1_GetFormatFromName(name.c_str());
00375         }
00376
00383     static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
00384         {
00385             return cloudx1_GetBytesFromFormat(fmt);
00386         }
00387
00391     using LogSeverity = cloudx1_LogSeverity;
00395     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00396
00400     static log_cb_t _log_cb;
00401     extern "C" {
00402         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00403             _log_cb(severity, std::string(c_msg));
00404         }
00405     }
00409     static inline void SetLogSeverity(LogSeverity severity)
00410         {
00411             cloudx1_SetLogSeverity(severity);
00412         }
00416     static inline void SetLoggerCallback(log_cb_t cb)
00417         {
00418             _log_cb=cb;
00419             cloudx1_SetLoggerCallback(&_log_cb_c);
00420         }
00421
00422 #if CLOUDX1_HAS_CLOUDBUS
00428     class CBus {
00429         public:
00430             CBus()
00431                 : m_bus(cloudx1_NewBus()) {}
00432             ~CBus()
```

```
00433                 {
00434                     cloudx1_FreeBus(m_bus);
00435                     m_bus = nullptr;
00436                 }
00437             cloudx1_CBus *Get() const { return m_bus; }
00438         private:
00439             cloudx1_CBus *m_bus;
00440     };
00441 #endif // CLOUDX1_HAS_CLOUDBUS
00442
00448     class CPresetLoader {
00449     public:
00450         CPresetLoader() = default;
00451         virtual ~CPresetLoader() = default;
00452
00453         virtual bool IsReadOnly() = 0;
00454         virtual bool Exists(const std::filesystem::path &name) = 0;
00455         virtual bool Remove(const std::filesystem::path &name) = 0;
00456         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00457         virtual std::vector<std::filesystem::path> GetAll() = 0;
00458         virtual std::string Read(const std::filesystem::path &filename) = 0;
00459         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00460     };
00461     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00462
00468     class CInstance {
00469     public:
00473         CInstance()
00474             {
00475                 params = cloudx1_NewParameters();
00476                 if (!params) throw std::bad_alloc();
00477             }
00478         // non-copyable
00479         CInstance( const CInstance& ) = delete;
00480         CInstance& operator=( const CInstance& ) = delete;
00484         virtual ~CInstance()
00485             {
00486                 Stop();
00487                 if (params) cloudx1_FreeParameters(params);
00488             }
00492        bool IsOk() const { return instance!=nullptr; }
00502         void SetParam(const std::string& name, const std::string& value)
00503             {
00504                 assert(params);
00505                 cloudx1_SetParameter(params, name.c_str(), value.c_str());
00506             }
00513         std::string GetParam(const std::string& name)
00514             {
00515                 assert(params);
00516                 auto c_value=cloudx1_GetParameter(params, name.c_str());
00517                 std::string ret(c_value);
00518                 cloudx1_FreeParameterValue(c_value);
00519                 return ret;
00520             }
00521
00522 #if CLOUDX1_HAS_CLOUDBUS
00531             void SetBus(const CBus& bus) {
00532                 assert(params);
00533                 cloudx1_SetInstanceBus(params,bus.Get());
00534             }
00535 #endif // CLOUDX1_HAS_CLOUDBUS
00536
00547         void SetPresetManager(CPresetLoader *preset_manager)
00548             {
00549                 if (!preset_manager) return;
00550                 assert(params);
00551                 cloudx1_SetPresetManager(params,
00552                     cloudx1_custom_reader,
00553                     cloudx1_custom_reader_free,
00554                     cloudx1_custom_writer,
00555                     cloudx1_custom_exists,
00556                     cloudx1_custom_getall,
00557                     cloudx1_custom_getall_free,
00558                     cloudx1_custom_remove,
00559                     cloudx1_custom_rename,
00560                     preset_manager->IsReadOnly(),
00561                     preset_manager
00562                 );
00563             }
00564
00587         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)
00588             {
00589                 assert(!instance);
00590                 std::string l_save_path;
00591                 if (!save_path.empty()) {
```

```
00592                   l_save_path = save_path.u8string();
00593               }
00594               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);
00595               if (!instance) return false;
00596               if (!init_metadata.empty()) {
00597                   for (auto&& [key,value]: init_metadata) {
00598                       cloudx1_SetMetadata(instance, key.c_str(), value.c_str());
00599                   }
00600                   init_metadata.clear();
00601               }
00602               update_thread = std::thread([this, json_port](){
00603                   OnUpdateThreadStart();
00604                   cloudx1_StartUpdateThread(instance, json_port);
00605                   OnUpdateThreadStop();
00606               });
00607               int timeout=5000;
00608               #ifdef DEBUG
00609                   timeout=60*1000; // For debugger pauses
00610               #endif
00611               if (cloudx1_WaitUpdateThreadReady(instance, timeout)<0) {
00612                   return false;
00613               }
00614               return true;
00615           }
00616
00624       unsigned int GetChunkFrames() {
00625           if (instance) {
00626               return cloudx1_GetProcessChunkFrames(instance);
00627           }
00628           return 0;
00629       }
00630
00638       void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
    change_kind)
00639           {
00640               cloudx1_PresetManager_InformChange(instance, relative_path.c_str(),
    (cloudx1_PresetChange_Kind) change_kind);
00641           }
00642
00654       void SetMetadata(const std::string& key, const char* value)
00655           {
00656               if (instance) {
00657                   cloudx1_SetMetadata(instance, key.c_str(), value);
00658               } else {
00659                   init_metadata.push_back( {key, value} );
00660               }
00661           }
00662
00672       void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
00673           {
00674               if (instance) {
00675                   typedef const char* pchar;
00676                   pchar* keyvalue=new pchar[2*list.size()+1];
00677                   size_t n=0;
00678                   for (auto&& [key,value]: list) {
00679                       keyvalue[2*n+0]=key.c_str();
00680                       keyvalue[2*n+1]=value;
00681                       n++;
00682                   }
00683                   keyvalue[2*n+0]=nullptr;
00684
00685                   cloudx1_SetMetadataMulti(instance, keyvalue);
00686
00687                   delete[] keyvalue;
00688               } else {
00689                   for (auto&& [key,value]: list) {
00690                       if (value) {
00691                           init_metadata.push_back( {key, value} );
00692                       }
00693                   }
00694               }
00695           }
00696
00702       std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
00703           {
00704               std::vector< std::tuple<std::string,std::string> > values;
00705               const char** c_values = cloudx1_GetMetadataInfos(instance);
00706               if (c_values) {
00707                   for (const char** c_value=c_values; *c_value; ) {
00708                       std::string key(*c_value);
00709                       c_value++;
00710                       if (*c_value) {
00711                           std::string descr(*c_value);
00712                           c_value++;
00713                           values.push_back( {key,descr} );
00714                       }
```

```
00715                     }
00716                     cloudx1_FreeMetadataInfos(instance, c_values);
00717                 }
00718                 return values;
00719             }
00720
00724         virtual void OnUpdateThreadStart() {}
00728         virtual void OnUpdateThreadStop() {}
00729
00730 #if CLOUDX1_HAS_WEBSERVER
00744         bool StartWebServer(int http_port, int https_port=0)
00745             {
00746                 assert(instance);
00747                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00748                 webserver = cloudx1_Webserver(http_port,https_port,instance);
00749                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00750                     return false;
00751                 }
00752                 int web_status=cloudx1_Webserver_Status(webserver);
00753                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00754                     return false;
00755                 }
00756                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00757                     return false;
00758                 }
00759                 return true;
00760             }
00761
00769             void StopWebServer(int timeout_ms = 1000)
00770             {
00771                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00772                     cloudx1_Webserver_Stop(webserver, timeout_ms);
00773                     webserver = SOUND4_INVALID_WEBSERVER_ID;
00774                 }
00775             }
00776
00780         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
00781             {
00782                 assert(instance);
00783                 cloudx1_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
00784             }
00788         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
00789             {
00790                 assert(instance);
00791                 char* c_contenttype=nullptr;
00792                 char* c_content=nullptr;
00793                 cloudx1_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
00794                 contenttype=c_contenttype;
00795                 content=c_content;
00796                 cloudx1_Webserver_FreeString(c_contenttype);
00797                 cloudx1_Webserver_FreeString(c_content);
00798         }
00799 #endif // CLOUDX1_HAS_WEBSERVER
00800
00809         int Terminate()
00810             {
00811                 if (!instance) return 0;
00812                 return cloudx1_TerminateProcess(instance);
00813             }
00814
00818         void Stop()
00819             {
00820                 if (!instance) return;
00821 #if CLOUDX1_HAS_WEBSERVER
00822                 StopWebServer();
00823 #endif // CLOUDX1_HAS_WEBSERVER
00824                 if (update_thread.joinable()) {
00825                     cloudx1_StopUpdateThread(instance); // this will exit the update thread
00826                     update_thread.join();
00827                 }
00828                 cloudx1_ExitProcess(instance);
00829                 instance=nullptr;
00830             }
00831
00841         unsigned int GetEstimatedDelay()
00842             {
00843                 assert(instance);
00844                 return cloudx1_GetEstimatedDelay(instance);
00845             }
00846
00859         std::array<float,InputSampleSize>& GetBufferIn()
00860             {
00861                 assert(instance);
00862                 float *buf=cloudx1_GetBufferIn(instance);
00863                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
```

```
00864
00865             }
00877         std::array<float,OutputSampleSize>& GetBufferOut()
00878             {
00879                 assert(instance);
00880                 float *buf= cloudx1_GetBufferOut(instance);
00881                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00882             }
00886         void ProcessAudio()
00887             {
00888                 assert(instance);
00889                 cloudx1_ProcessAudio(instance, cloudx1_GetBufferIn(instance),
      cloudx1_GetBufferOut(instance));
00890             }
00902         void ProcessAudio_Planar(float const * const *input, float * const *output)
00903             {
00904                 assert(instance);
00905                 cloudx1_ProcessAudio_Planar(instance, input, output);
00906             }
00907
00919             template<typename T>
00920         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00921             {
00922                 assert(instance);
00923                 std::vector<T> output;
00924                 unsigned int out_offset=0;
00925                 unsigned int in_offset=0;
00926                 unsigned int todo = input.size();
00927                 while (todo>0) {
00928                     unsigned int left = AddAudio(&input[in_offset], todo);
00929                     unsigned int out_avail = cloudx1_GetOutputCount(instance);
00930                     output.resize(out_offset + out_avail);
00931                     GetAudio(&output[out_offset], out_avail);
00932                     out_offset+=out_avail;
00933                     in_offset += todo-left;
00934                     todo=left;
00935                 }
00936                 return output;
00937             }
00938
00944         class CClient {
00945             public:
00946                 CClient(cloudx1_CInstance* instance)
00947                     {
00948                         assert(instance);
00949                         client=cloudx1_NewClient(instance);
00950                         if (!client) throw std::bad_alloc();
00951                     }
00952                 // non-copyable
00953                 CClient( const CInstance& ) = delete;
00954                 CClient& operator=( const CInstance& ) = delete;
00955                 ~CClient()
00956                     {
00957                         if (client) {
00958                             cloudx1_DeleteClient(client);
00959                             client=nullptr;
00960                         }
00961                     }
00969                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
00970                     {
00971                         assert(client);
00972                         int need_save=0;
00973                         const char *canswer = cloudx1_ProcessJson(client, request.c_str(),
      &need_save);
00974                         if (!canswer) return {};
00975                         std::string answer(canswer);
00976                         cloudx1_FreeJsonAnswer (canswer);
00977                         if (NeedSave) {
00978                             *NeedSave=(need_save!=0);
00979                         }
00980                         return answer;
00981                     }
00982             private:
00983                 cloudx1_CClientInstance *client = nullptr;
00984         };
00992         std::shared_ptr<CClient> NewClient()
00993             {
00994                 assert(instance);
00995                 return std::make_shared<CClient>(instance);
00996             }
00997
01008         bool SaveState() {
01009             assert(instance);
01010             if (cloudx1_SaveState(instance)!=0) {
01011                 return false;
01012             }
01013             return true;
```

```
01014         }
01015
01016     protected:
01017             template<typename T>
01018         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01019             {
01020                 assert(instance);
01021                 return cloudx1_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01022             }
01023             template<typename T>
01024         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01025             {
01026                 assert(instance);
01027                 return cloudx1_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01028             }
01029
01030     private:
01031         cloudx1_CParameters* params = nullptr;
01032         cloudx1_CInstance* instance = nullptr;
01033         std::thread update_thread;
01034         std::vector< std::pair<std::string,std::string> > init_metadata;
01035 #if CLOUDX1_HAS_WEBSERVER
01036         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01037 #endif // CLOUDX1_HAS_WEBSERVER
01038     };
01039
01040 }; // namespace cloudx1
01041 }; // namespace sound4
01042
01043 // bridge C callbacks to CPresetLoader
01044 extern "C" {
01045     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01046         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01047         std::string content=preset_loader->Read(filename);
01048         return strdup(content.c_str());
01049     }
01050     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01051         if (!content) return;
01052         free(content);
01053     }
01054     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01055         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01056         auto res=preset_loader->Write(filename,content);
01057         return res?0:-1;
01058     }
01059     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01060         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01061         auto res=preset_loader->Exists(filename);
01062         return res?0:-1;
01063     }
01064     static fs_char** cloudx1_custom_getall(void* userdata) {
01065         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01066         auto res=preset_loader->GetAll();
01067         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01068         for (size_t n=0;n<res.size();n++) {
01069             all[n]=fs_strdup(res[n].c_str());
01070         }
01071         all[res.size()]=nullptr;
01072         return all;
01073     }
01074     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01075         if (!all) return;
01076         for (fs_char** one=all;*one;one++) {
01077             free(*one);
01078         }
01079         free(all);
01080     }
01081     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01082         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01083         auto res=preset_loader->Remove(filename);
01084         return res?0:-1;
01085     }
01086     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01087         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01088         auto res=preset_loader->Rename(from,to);
01089         return res?0:-1;
01090     }
01091
01092 };
```

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 1542 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 1547 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 1557 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 1528 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 1533 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 1564 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 1569 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 1537 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 #else // !_WIN32
00045     #define fs_strdup strdup
00046 #endif // !_WIN32
00047
00048 // bridge C callbacks to CPresetLoader
00049 extern "C" {
00050     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00051     static void cloudx1_custom_reader_free(char *content, void* userdata);
00052     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00053     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00054     static fs_char** cloudx1_custom_getall(void* userdata);
00055     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00056     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00057     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00058 };
00059
00063 namespace sound4 {
00067 namespace cloudx1 {
00071 namespace dyn {
00072
00076     namespace helper {
00077
00083             template <typename T>
00084         class DynFuncHolder {
00085             // Storage definition, depending on OS
00086             #if UNIXLIKE
00087                 using DynFunc_t = void *;
00088             #else
00089                 using DynFunc_t = FARPROC;
00090             #endif
00091         private:
00092             DynFunc_t m_ptr = NULL;
00093         public:
00094             DynFuncHolder() = default;
00095             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00096             operator bool() const { return (m_ptr!=NULL); }
00097             bool IsOk() const { return m_ptr!=NULL; }
00098             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00099         };
00100
00105         class CDynLoader {
00106             // Storage definition, depending on OS
00107             #if UNIXLIKE
00108                 using DynLib_t = void *;
00109             #else
00110                 using DynLib_t = HMODULE;
00111             #endif
00112             private:
00113                 DynLib_t m_lib {};
00114             public:
00115                 CDynLoader() = default;
00116                 CDynLoader(CDynLoader&&) = default;
00117                 // No copy allowed
00118                 CDynLoader(const CDynLoader&) = delete;
00119                 CDynLoader& operator=(CDynLoader const&) = delete;
00120                 operator bool() const {
00121                     return (m_lib!=NULL);
00122                 }
00123                 bool IsOk() const {
00124                     return (m_lib!=NULL);
00125                 }
00126                 #if UNIXLIKE
00127                     void Close() {
00128                         if (IsOk()) {
00129                             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00130                             dlclose(m_lib);
00131                             #endif // __SANITIZE_ADDRESS__
00132                             m_lib=NULL;
00133                         }
00134                     }
00135                     bool Load(const std::filesystem::path& dynlib)
```

```
00136                     {
00137                         Close();
00138                         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00139                         return IsOk();
00140                     }
00141                         template <typename T>
00142                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00143                         auto ptr=dlsym(m_lib, name.c_str());
00144                         if (!ptr) {
00145                             throw std::runtime_error("Missing function in library");
00146                         }
00147                         return DynFuncHolder<T>(ptr);
00148                     }
00149                         template <typename T>
00150                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00151                         auto ptr=dlsym(m_lib, name.c_str());
00152                         return DynFuncHolder<T>(ptr);
00153                     }
00154                     static std::filesystem::path GetThisLibraryPath(void) {
00155                         static Dl_info info;
00156                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
00157                         auto rp=realpath(info.dli_fname, NULL);
00158                         std::filesystem::path p(rp);
00159                         free(rp);
00160                         return p;
00161                     }
00162                 #else
00163                     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
00164                     {
00165                         Close();
00166                         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00167                         return IsOk();
00168                     }
00169                     void Close() {
00170                         if (IsOk()) {
00171                             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00172                             // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00173                             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00174                                 FreeLibraryAndExitThread(m_lib, 0);
00175                             #elif defined(SOUND4_CALL_FREELIBRARY)
00176                                 FreeLibrary(m_lib);
00177                             #endif // SOUND4_CALL_FREELIBRARY
00178                             m_lib=NULL;
00179                         }
00180                     }
00181                         template <typename T>
00182                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00183                         auto ptr=GetProcAddress(m_lib, name.c_str());
00184                         if (!ptr) {
00185                             throw std::runtime_error("Missing function in library");
00186                         }
00187                         return DynFuncHolder<T>(ptr);
00188                     }
00189                         template <typename T>
00190                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00191                         auto ptr=GetProcAddress(m_lib, name.c_str());
00192                         return DynFuncHolder<T>(ptr);
00193                     }
00194                     static std::filesystem::path GetThisLibraryPath(void) {
00195                         static wchar_t path[MAX_PATH]={};
00196                         HMODULE hm = NULL;
00197                         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00198                                 GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00199                                 (LPCWSTR) &GetThisLibraryPath, &hm))
00200                         {
00201                             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00202                                 return std::filesystem::path(path);
00203                             }
00204                         }
00205                         return {};
00206                     }
00207                 #endif
00208                 virtual ~CDynLoader() {
00209                     Close();
00210                 }
00211         }; // class CDynLoader
00212     }; // namespace helper
00213
00220     class CDynLib {
00221     private:
00222         helper::CDynLoader m_lib;
00223         bool m_bOK = false;
00224     public:
00225         CDynLib() = default;
```

```
00226
00227         // All those wrapped functions have the same signature as there C cloudx1_XXX
00228         helper::DynFuncHolder< decltype(cloudx1_GetVersion              ) > GetVersion              ;
00229         helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames    ) > GetChunkSizeInFrames    ;
00230         helper::DynFuncHolder< decltype(cloudx1_GetChannelCount         ) > GetChannelCount         ;
00231         helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount      ) > GetAudioInputCount      ;
00232         helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount     ) > GetAudioOutputCount     ;
00233         helper::DynFuncHolder< decltype(cloudx1_GetSampleRate           ) > GetSampleRate           ;
00234         helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback       ) > SetLoggerCallback       ;
00235         helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity          ) > SetLogSeverity          ;
00236         helper::DynFuncHolder< decltype(cloudx1_NewParameters           ) > NewParameters           ;
00237         helper::DynFuncHolder< decltype(cloudx1_FreeParameters          ) > FreeParameters          ;
00238         helper::DynFuncHolder< decltype(cloudx1_SetParameter            ) > SetParameter            ;
00239         helper::DynFuncHolder< decltype(cloudx1_GetParameter            ) > GetParameter            ;
00240         helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue      ) > FreeParameterValue      ;
00241         helper::DynFuncHolder< decltype(cloudx1_InitProcess             ) > InitProcess             ;
00242         helper::DynFuncHolder< decltype(cloudx1_InitProcess2            ) > InitProcess2            ;
00243         helper::DynFuncHolder< decltype(cloudx1_TerminateProcess        ) > TerminateProcess        ;
00244         helper::DynFuncHolder< decltype(cloudx1_ExitProcess             ) > ExitProcess             ;
00245         helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread       ) > StartUpdateThread       ;
00246         helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread        ) > StopUpdateThread        ;
00247         helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady   ) > WaitUpdateThreadReady   ;
00248         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio            ) > ProcessAudio            ;
00249         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar     ) > ProcessAudio_Planar     ;
00250         helper::DynFuncHolder< decltype(cloudx1_GetBufferIn             ) > GetBufferIn             ;
00251         helper::DynFuncHolder< decltype(cloudx1_GetBufferOut            ) > GetBufferOut            ;
00252         helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay       ) > GetEstimatedDelay       ;
00253         helper::DynFuncHolder< decltype(cloudx1_GetFormatName           ) > GetFormatName           ;
00254         helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName       ) > GetFormatFromName       ;
00255         helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat      ) > GetBytesFromFormat      ;
00256         helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame       ) > GetMaxPacketFrame       ;
00257         helper::DynFuncHolder< decltype(cloudx1_AddAudio                ) > AddAudio                ;
00258         helper::DynFuncHolder< decltype(cloudx1_AddPadAudio             ) > AddPadAudio             ;
00259         helper::DynFuncHolder< decltype(cloudx1_GetOutputCount          ) > GetOutputCount          ;
00260         helper::DynFuncHolder< decltype(cloudx1_GetAudio                ) > GetAudio                ;
00261         helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom        ) > AudioConvertFrom        ;
00262         helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo          ) > AudioConvertTo          ;
00263         helper::DynFuncHolder< decltype(cloudx1_StereoToMono            ) > StereoToMono            ;
00264         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo            ) > MonoToStereo            ;
00265         helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo;
00266         helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo   ) > AudioMonoToLiveStereo   ;
00267         helper::DynFuncHolder< decltype(cloudx1_NewClient                ) > NewClient              ;
00268         helper::DynFuncHolder< decltype(cloudx1_DeleteClient             ) > DeleteClient           ;
00269         helper::DynFuncHolder< decltype(cloudx1_ProcessJson              ) > ProcessJson            ;
00270         helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer           ) > FreeJsonAnswer         ;
00271         helper::DynFuncHolder< decltype(cloudx1_SaveState                ) > SaveState              ;
00272 #if CLOUDX1_HAS_WEBSERVER
00273         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp            ) > Webserver_tcp          ;
00274         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2           ) > Webserver_tcp2         ;
00275         helper::DynFuncHolder< decltype(cloudx1_Webserver                ) > Webserver              ;
00276         helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop           ) > Webserver_Stop         ;
00277         helper::DynFuncHolder< decltype(cloudx1_Webserver_Status         ) > Webserver_Status       ;
00278 #endif // CLOUDX1_HAS_WEBSERVER
00279
00280         // added 2023-03-22
00281         helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar      ) > StereoToMono_Planar    ;
00282         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar      ) > MonoToStereo_Planar    ;
00283
00284         // added 2023-05-22
00285         helper::DynFuncHolder< decltype(cloudx1_SetMetadata              ) > SetMetadata            ;
00286         helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos         ) > GetMetadataInfos       ;
00287         helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos        ) > FreeMetadataInfos      ;
00288
00289         // added 2023-06-20
00290         helper::DynFuncHolder< decltype(cloudx1_SetPresetManager                ) > SetPresetManager
    ;
00291         helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange      ) >
    PresetManager_InformChange      ;
00292
00293         // added 2023-09-21
00294         helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames    ) >
    GetPossibleChunkSizeInFrames   ;
00295         helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames           ) >
    GetProcessChunkFrames   ;
00296         helper::DynFuncHolder< decltype(cloudx1_InitProcess3                    ) > InitProcess3
    ;
00297
00298 #if CLOUDX1_HAS_CLOUDBUS
00299         // added 2023-06-26
00300         helper::DynFuncHolder< decltype(cloudx1_NewBus                  ) > NewBus      ;
00301         helper::DynFuncHolder< decltype(cloudx1_FreeBus                 ) > FreeBus      ;
00302         helper::DynFuncHolder< decltype(cloudx1_SetInstanceBus          ) > SetInstanceBus      ;
00303 #endif // CLOUDX1_HAS_CLOUDBUS
00304
00305         // added 2024-02-19
00306         helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti        ) > SetMetadataMulti        ;
00307
```

```
00308 #if CLOUDX1_HAS_WEBSERVER
00309         // added 2024-05-23
00310         helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth    ) > Webserver_SetAppHealth
      ;
00311         helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth    ) > Webserver_GetAppHealth
      ;
00312         helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString      ) > Webserver_FreeString
      ;
00313 #endif // CLOUDX1_HAS_WEBSERVER
00314
00322         bool Load(const std::filesystem::path& path = {}) {
00323             std::string filename;
00324             #if defined(__APPLE__)
00325                 filename="libsound4.x1.cloud.dylib";
00326             #elif defined(__unix__)
00327                 filename="libsound4.x1.cloud.so";
00328             #else
00329                 filename="sound4.x1.cloud.dll";
00330             #endif
00331             m_lib.Close();
00332             // If a path is given, use it directly and do not try other path
00333             if (!path.empty() && !m_lib.Load(path / filename)) {
00334                 return false;
00335             } else if (!m_lib.IsOk()) {
00336                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00337                 #ifndef _WIN32
00338                 // Linux: if in a bin directory, try ../lib first
00339                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00340                     auto libdir = thisdir.parent_path() / "lib";
00341                     m_lib.Load(libdir / filename);
00342                 }
00343                 #endif // !_WIN32
00344                 if (!m_lib.IsOk()) {
00345                     // Search in the same directory this code is in
00346                     m_lib.Load(thisdir / filename);
00347                 }
00348                 if (!m_lib.IsOk()) {
00349                     // Try current path
00350                     std::error_code ec;
00351                     auto p = std::filesystem::current_path(ec);
00352                     if (!ec) {
00353                         m_lib.Load(p / filename);
00354                     }
00355                 }
00356                 if (!m_lib.IsOk()) {
00357                     return false;
00358                 }
00359             }
00360             // Load all C functions
00361             try {
00362                 GetVersion              = m_lib.GetSymbol< decltype(cloudx1_GetVersion             )
      >("cloudx1_GetVersion"              );
00363                 GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(cloudx1_GetChunkSizeInFrames   )
      >("cloudx1_GetChunkSizeInFrames"    );
00364                 GetChannelCount         = m_lib.GetSymbol< decltype(cloudx1_GetChannelCount        )
      >("cloudx1_GetChannelCount"         );
00365                 GetAudioInputCount      = m_lib.GetSymbol< decltype(cloudx1_GetAudioInputCount     )
      >("cloudx1_GetAudioInputCount"      );
00366                 GetAudioOutputCount     = m_lib.GetSymbol< decltype(cloudx1_GetAudioOutputCount    )
      >("cloudx1_GetAudioOutputCount"     );
00367                 GetSampleRate           = m_lib.GetSymbol< decltype(cloudx1_GetSampleRate          )
      >("cloudx1_GetSampleRate"           );
00368                 SetLoggerCallback       = m_lib.GetSymbol< decltype(cloudx1_SetLoggerCallback      )
      >("cloudx1_SetLoggerCallback"       );
00369                 SetLogSeverity          = m_lib.GetSymbol< decltype(cloudx1_SetLogSeverity         )
      >("cloudx1_SetLogSeverity"          );
00370                 NewParameters           = m_lib.GetSymbol< decltype(cloudx1_NewParameters          )
      >("cloudx1_NewParameters"           );
00371                 FreeParameters          = m_lib.GetSymbol< decltype(cloudx1_FreeParameters         )
      >("cloudx1_FreeParameters"          );
00372                 SetParameter            = m_lib.GetSymbol< decltype(cloudx1_SetParameter           )
      >("cloudx1_SetParameter"            );
00373                 GetParameter            = m_lib.GetSymbol< decltype(cloudx1_GetParameter           )
      >("cloudx1_GetParameter"            );
00374                 FreeParameterValue      = m_lib.GetSymbol< decltype(cloudx1_FreeParameterValue     )
      >("cloudx1_FreeParameterValue"      );
00375                 InitProcess             = m_lib.GetSymbol< decltype(cloudx1_InitProcess            )
      >("cloudx1_InitProcess"             );
00376                 InitProcess2            = m_lib.GetSymbol< decltype(cloudx1_InitProcess2           )
      >("cloudx1_InitProcess2"            );
00377                 TerminateProcess        = m_lib.GetSymbol< decltype(cloudx1_TerminateProcess       )
      >("cloudx1_TerminateProcess"        );
00378                 ExitProcess             = m_lib.GetSymbol< decltype(cloudx1_ExitProcess            )
      >("cloudx1_ExitProcess"             );
00379                 StartUpdateThread       = m_lib.GetSymbol< decltype(cloudx1_StartUpdateThread      )
      >("cloudx1_StartUpdateThread"       );
00380                 StopUpdateThread        = m_lib.GetSymbol< decltype(cloudx1_StopUpdateThread       )
```

```
>("cloudx1_StopUpdateThread"       );
00381                 WaitUpdateThreadReady    = m_lib.GetSymbol< decltype(cloudx1_WaitUpdateThreadReady  )
>("cloudx1_WaitUpdateThreadReady"  );
00382                 ProcessAudio             = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio           )
>("cloudx1_ProcessAudio"           );
00383                 ProcessAudio_Planar      = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio_Planar    )
>("cloudx1_ProcessAudio_Planar"    );
00384                 GetBufferIn              = m_lib.GetSymbol< decltype(cloudx1_GetBufferIn            )
>("cloudx1_GetBufferIn"            );
00385                 GetBufferOut             = m_lib.GetSymbol< decltype(cloudx1_GetBufferOut           )
>("cloudx1_GetBufferOut"           );
00386                 GetEstimatedDelay        = m_lib.GetSymbol< decltype(cloudx1_GetEstimatedDelay      )
>("cloudx1_GetEstimatedDelay"      );
00387                 GetFormatName            = m_lib.GetSymbol< decltype(cloudx1_GetFormatName          )
>("cloudx1_GetFormatName"          );
00388                 GetFormatFromName        = m_lib.GetSymbol< decltype(cloudx1_GetFormatFromName      )
>("cloudx1_GetFormatFromName"      );
00389                 GetBytesFromFormat       = m_lib.GetSymbol< decltype(cloudx1_GetBytesFromFormat     )
>("cloudx1_GetBytesFromFormat"     );
00390                 GetMaxPacketFrame        = m_lib.GetSymbol< decltype(cloudx1_GetMaxPacketFrame      )
>("cloudx1_GetMaxPacketFrame"      );
00391                 AddAudio                 = m_lib.GetSymbol< decltype(cloudx1_AddAudio               )
>("cloudx1_AddAudio"               );
00392                 AddPadAudio              = m_lib.GetSymbol< decltype(cloudx1_AddPadAudio            )
>("cloudx1_AddPadAudio"            );
00393                 GetOutputCount           = m_lib.GetSymbol< decltype(cloudx1_GetOutputCount         )
>("cloudx1_GetOutputCount"         );
00394                 GetAudio                 = m_lib.GetSymbol< decltype(cloudx1_GetAudio               )
>("cloudx1_GetAudio"               );
00395                 AudioConvertFrom         = m_lib.GetSymbol< decltype(cloudx1_AudioConvertFrom       )
>("cloudx1_AudioConvertFrom"       );
00396                 AudioConvertTo           = m_lib.GetSymbol< decltype(cloudx1_AudioConvertTo         )
>("cloudx1_AudioConvertTo"         );
00397                 StereoToMono             = m_lib.GetSymbol< decltype(cloudx1_StereoToMono           )
>("cloudx1_StereoToMono"           );
00398                 MonoToStereo             = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo           )
>("cloudx1_MonoToStereo"           );
00399                 AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(cloudx1_AudioMonoFromLiveStereo)
>("cloudx1_AudioMonoFromLiveStereo");
00400                 AudioMonoToLiveStereo    = m_lib.GetSymbol< decltype(cloudx1_AudioMonoToLiveStereo  )
>("cloudx1_AudioMonoToLiveStereo"  );
00401                 NewClient                = m_lib.GetSymbol< decltype(cloudx1_NewClient              )
>("cloudx1_NewClient"              );
00402                 DeleteClient             = m_lib.GetSymbol< decltype(cloudx1_DeleteClient           )
>("cloudx1_DeleteClient"           );
00403                 ProcessJson              = m_lib.GetSymbol< decltype(cloudx1_ProcessJson            )
>("cloudx1_ProcessJson"            );
00404                 FreeJsonAnswer           = m_lib.GetSymbol< decltype(cloudx1_FreeJsonAnswer          )
>("cloudx1_FreeJsonAnswer"         );
00405                 SaveState                = m_lib.GetSymbol< decltype(cloudx1_SaveState              )
>("cloudx1_SaveState"              );
00406 #if CLOUDX1_HAS_WEBSERVER
00407                 Webserver_tcp            = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp          )
>("cloudx1_Webserver_tcp"          );
00408                 Webserver_tcp2           = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp2         )
>("cloudx1_Webserver_tcp2"         );
00409                 Webserver                = m_lib.GetSymbol< decltype(cloudx1_Webserver              )
>("cloudx1_Webserver"              );
00410                 Webserver_Stop           = m_lib.GetSymbol< decltype(cloudx1_Webserver_Stop         )
>("cloudx1_Webserver_Stop"         );
00411                 Webserver_Status         = m_lib.GetSymbol< decltype(cloudx1_Webserver_Status       )
>("cloudx1_Webserver_Status"       );
00412 #endif // CLOUDX1_HAS_WEBSERVER
00413
00414                 // added 2023-03-22
00415                 StereoToMono_Planar      = m_lib.GetSymbol< decltype(cloudx1_StereoToMono_Planar    )
>("cloudx1_StereoToMono_Planar"    );
00416                 MonoToStereo_Planar      = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo_Planar    )
>("cloudx1_MonoToStereo_Planar"    );
00417
00418             } catch (std::runtime_error& ) {
00419                 return false;
00420             }
00421             // C functions allowed to be missing
00422             try {
00423                 // added 2023-05-22
00424                 SetMetadata              = m_lib.GetSymbol< decltype(cloudx1_SetMetadata            )
>("cloudx1_SetMetadata"            );
00425                 GetMetadataInfos         = m_lib.GetSymbol< decltype(cloudx1_GetMetadataInfos       )
>("cloudx1_GetMetadataInfos"       );
00426                 FreeMetadataInfos        = m_lib.GetSymbol< decltype(cloudx1_FreeMetadataInfos      )
>("cloudx1_FreeMetadataInfos"      );
00427                 // added 2023-06-20
00428                 SetPresetManager            = m_lib.GetSymbol< decltype(cloudx1_SetPresetManager
) >("cloudx1_SetPresetManager"              );
00429                 PresetManager_InformChange  = m_lib.GetSymbol<
decltype(cloudx1_PresetManager_InformChange     ) >("cloudx1_PresetManager_InformChange"    );
```

```
00430
00431                 // added 2023-09-21
00432                 GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
      decltype(cloudx1_GetPossibleChunkSizeInFrames  ) >("cloudx1_GetPossibleChunkSizeInFrames"  );
00433                 GetProcessChunkFrames        = m_lib.GetSymbol< decltype(cloudx1_GetProcessChunkFrames
      ) >("cloudx1_GetProcessChunkFrames"         );
00434                 InitProcess3                 = m_lib.GetSymbol< decltype(cloudx1_InitProcess3
      ) >("cloudx1_InitProcess3"                  );
00435
00436 #if CLOUDX1_HAS_CLOUDBUS
00437                 // added 2023-06-26
00438                 NewBus                   = m_lib.GetSymbol< decltype(cloudx1_NewBus                 )
      >("cloudx1_NewBus"                 );
00439                 FreeBus                  = m_lib.GetSymbol< decltype(cloudx1_FreeBus                )
      >("cloudx1_FreeBus"                );
00440                 SetInstanceBus           = m_lib.GetSymbol< decltype(cloudx1_SetInstanceBus         )
      >("cloudx1_SetInstanceBus"         );
00441 #endif // CLOUDX1_HAS_CLOUDBUS
00442
00443                 // added 2024-02-19
00444                 SetMetadataMulti         = m_lib.GetSymbol< decltype(cloudx1_SetMetadataMulti       )
      >("cloudx1_SetMetadataMulti"       );
00445
00446 #if CLOUDX1_HAS_WEBSERVER
00447                 // added 2024-05-23
00448                 Webserver_SetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_SetAppHealth
      ) >("Webserver_SetAppHealth"     );
00449                 Webserver_GetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_GetAppHealth
      ) >("Webserver_GetAppHealth"     );
00450                 Webserver_FreeString        = m_lib.GetSymbol< decltype(cloudx1_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00451 #endif // CLOUDX1_HAS_WEBSERVER
00452
00453             }  catch (std::runtime_error& ) {
00454                 // Ignored, handler will take care
00455             }
00456
00457             m_bOK=true;
00458             return true;
00459         }
00466         bool IsOk() const {
00467             return m_bOK;
00468         }
00469
00470     };
00471
00472
00473
00474     static constexpr const char *process_name = "SOUND4 x1.CLOUD";
00475     static constexpr const char *process_shortname = "x1";
00476     using SampleFormat = cloudx1_SampleFormat;
00481     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00486     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00491     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00496     const size_t InputSampleSize=ChunkFrames*InputChannels;
00501     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00506     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
00507
00511     namespace helper {
00512         #ifdef _WIN32
00518         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00519             if (wstr.empty()) return {};
00520             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00521             std::string str(size, 0);
00522             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00523             return str;
00524         }
00530         static inline std::wstring UTF8ToWString(const std::string& str) {
00531             if (str.empty()) return std::wstring();
00532             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00533             std::wstring wstr(size, 0);
00534             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00535             return wstr;
00536         }
00537         #endif // _WIN32
00538
00544             template<typename T>
00545         struct SampleFormat {
00546         };
00547             template<>
00548         struct SampleFormat<int16_t> {
00549             const cloudx1_SampleFormat format = S16_NATIVE;
00550         };
00551             template<>
00552         struct SampleFormat<int32_t> {
00553             const cloudx1_SampleFormat format = S32_NATIVE;
```

```
00554         };
00555             template<>
00556         struct SampleFormat<float> {
00557             const cloudx1_SampleFormat format = F32_NATIVE;
00558         };
00559
00560         // ------------------------------------------------------------
00572         static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00573             dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00574         }
00575
00587         static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00588             dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00589         }
00590
00601         static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00602             dynlib.StereoToMono(input, output, nFrame);
00603         }
00604
00615         static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00616             dynlib.MonoToStereo(input, output, nFrame);
00617         }
00618
00630         static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
      *inputR, float *output, size_t nFrame) {
00631             dynlib.StereoToMono_Planar(inputL, inputR, output, nFrame);
00632         }
00633
00645         static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
      float *outputR, size_t nFrame) {
00646             dynlib.MonoToStereo_Planar(input, outputL, outputR, nFrame);
00647         }
00648
00656         static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
      *output) {
00657             dynlib.AudioMonoFromLiveStereo(payload, output);
00658         }
00659
00667         static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
      *payload) {
00668             dynlib.AudioMonoToLiveStereo(input, payload);
00669         }
00670
00671     }; // namespace helper
00672
00679     static inline std::string GetVersion(CDynLib& dynlib)
00680         {
00681             return std::string(dynlib.GetVersion());
00682         }
00683
00695     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00696         {
00697             return dynlib.GetChunkSizeInFrames();
00698         }
00699
00709     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00710         {
00711             std::vector<unsigned int> list;
00712             if (dynlib.GetPossibleChunkSizeInFrames) {
00713                 for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00714                     list.push_back(*src);
00715                 }
00716             } else {
00717                 list.push_back(64);
00718             }
00719             return list;
00720         }
00721
00728     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00729         {
00730             return dynlib.GetChannelCount();
00731         }
00732
00741     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00742         {
00743             return dynlib.GetAudioInputCount();
00744         }
00745
00754     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00755         {
00756             return dynlib.GetAudioOutputCount();
00757         }
00758
```

```
00767     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00768         {
00769             return dynlib.GetSampleRate();
00770         }
00771
00779     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00780     {
00781         // Do some sanity checks
00782         if (dynlib.GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00783             throw std::runtime_error("Bad library sampling rate");
00784         }
00785         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
00786             throw std::runtime_error("Bad library frame size");
00787         }
00788         if (dynlib.GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00789             throw std::runtime_error("Bad library channel count");
00790         }
00791         if (dynlib.GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00792             throw std::runtime_error("Bad library input count");
00793         }
00794         if (dynlib.GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00795             throw std::runtime_error("Bad library output count");
00796         }
00797     }
00798
00806     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00807         {
00808             return std::string(dynlib.GetFormatName(fmt));
00809         }
00810
00818     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00819         {
00820             return dynlib.GetFormatFromName(name.c_str());
00821         }
00822
00830     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00831         {
00832             return dynlib.GetBytesFromFormat(fmt);
00833         }
00834
00838     using LogSeverity = cloudx1_LogSeverity;
00842     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00843
00847     static log_cb_t _log_cb;
00848     extern "C" {
00849         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00850             _log_cb(severity, std::string(c_msg));
00851         }
00852     }
00856     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00857         {
00858             dynlib.SetLogSeverity(severity);
00859         }
00863     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00864         {
00865             _log_cb=cb;
00866             dynlib.SetLoggerCallback(&_log_cb_c);
00867         }
00868
00869 #if CLOUDX1_HAS_CLOUDBUS
00875     class CBus {
00876         public:
00877             CBus(CDynLib& dynlib)
00878                 : m_dynlib(dynlib)
00879                 {
00880                     if (m_dynlib.NewBus) {
00881                         m_bus=m_dynlib.NewBus();
00882                     }
00883                 }
00884             ~CBus()
00885                 {
00886                     if (m_bus && m_dynlib.FreeBus) {
00887                         m_dynlib.FreeBus(m_bus);
00888                     }
00889                     m_bus = nullptr;
00890                 }
00891             cloudx1_CBus *Get() const { return m_bus; }
00892         private:
00893             CDynLib& m_dynlib;
00894             cloudx1_CBus *m_bus;
00895     };
00896 #endif // CLOUDX1_HAS_CLOUDBUS
00897
00903     class CPresetLoader {
00904     public:
00905         CPresetLoader() = default;
00906         virtual ~CPresetLoader() = default;
```

```
00907
00908         virtual bool IsReadOnly() = 0;
00909         virtual bool Exists(const std::filesystem::path &name) = 0;
00910         virtual bool Remove(const std::filesystem::path &name) = 0;
00911         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00912         virtual std::vector<std::filesystem::path> GetAll() = 0;
00913         virtual std::string Read(const std::filesystem::path &filename) = 0;
00914         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00915     };
00916     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00917
00923     class CInstance {
00924     public:
00928         CInstance(CDynLib& dynlib)
00929         : m_dynlib(dynlib)
00930             {
00931                 params = m_dynlib.NewParameters();
00932                 if (!params) throw std::bad_alloc();
00933             }
00934         // non-copyable
00935         CInstance( const CInstance& ) = delete;
00936         CInstance& operator=( const CInstance& ) = delete;
00940         virtual ~CInstance()
00941             {
00942                 Stop();
00943                 if (params) m_dynlib.FreeParameters(params);
00944             }
00948        bool IsOk() const { return instance!=nullptr; }
00958         void SetParam(const std::string& name, const std::string& value)
00959             {
00960                 assert(params);
00961                 m_dynlib.SetParameter(params, name.c_str(), value.c_str());
00962             }
00969         std::string GetParam(const std::string& name)
00970             {
00971                 assert(params);
00972                 auto c_value=m_dynlib.GetParameter(params, name.c_str());
00973                 std::string ret(c_value);
00974                 m_dynlib.FreeParameterValue(c_value);
00975                 return ret;
00976             }
00977 #if CLOUDX1_HAS_CLOUDBUS
00986             void SetBus(const CBus& bus) {
00987                 assert(params);
00988                 if (m_dynlib.SetInstanceBus) {
00989                     m_dynlib.SetInstanceBus(params,bus.Get());
00990                 }
00991             }
00992 #endif // CLOUDX1_HAS_CLOUDBUS
00993
01004         void SetPresetManager(CPresetLoader *preset_manager)
01005             {
01006                 if (!preset_manager) return;
01007                 if (!m_dynlib.SetPresetManager) return;
01008                 assert(params);
01009                 m_dynlib.SetPresetManager(params,
01010                     cloudx1_custom_reader,
01011                     cloudx1_custom_reader_free,
01012                     cloudx1_custom_writer,
01013                     cloudx1_custom_exists,
01014                     cloudx1_custom_getall,
01015                     cloudx1_custom_getall_free,
01016                     cloudx1_custom_remove,
01017                     cloudx1_custom_rename,
01018                     preset_manager->IsReadOnly(),
01019                     preset_manager
01020                 );
01021             }
01022
01045         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)
01046             {
01047                 assert(!instance);
01048                 std::string l_save_path;
01049                 if (!save_path.empty()) {
01050                     l_save_path = save_path.u8string();
01051                 }
01052                 if (m_dynlib.InitProcess3) {
01053                     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);
01054                 } else if (frames_per_chunk!=64) {
01055                     return false;
01056                 } else {
01057                     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);
01058                 }
```

```
01059                 if (!instance) return false;
01060                 if (!init_metadata.empty()) {
01061                     for (auto&& [key,value]: init_metadata) {
01062                         m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
01063                     }
01064                     init_metadata.clear();
01065                 }
01066                 update_thread = std::thread([this, json_port](){
01067                     OnUpdateThreadStart();
01068                     m_dynlib.StartUpdateThread(instance, json_port);
01069                     OnUpdateThreadStop();
01070                 });
01071                 int timeout=5000;
01072                 #ifdef DEBUG
01073                     timeout=60*1000; // For debugger pauses
01074                 #endif
01075                 if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
01076                     return false;
01077                 }
01078                 return true;
01079             }
01080
01088         unsigned int GetChunkFrames() {
01089             if (instance && m_dynlib.GetProcessChunkFrames) {
01090                 return m_dynlib.GetProcessChunkFrames(instance);
01091             }
01092             return 64;
01093         }
01094
01102         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01103             {
01104                 if (m_dynlib.PresetManager_InformChange) {
01105                     m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
      (cloudx1_PresetChange_Kind) change_kind);
01106                 }
01107             }
01108
01120         void SetMetadata(const std::string& key, const char* value)
01121             {
01122                 if (m_dynlib.SetMetadata) {
01123                     if (instance) {
01124                         m_dynlib.SetMetadata(instance, key.c_str(), value);
01125                     } else {
01126                         init_metadata.push_back( {key, value} );
01127                     }
01128                 }
01129             }
01130
01131
01141         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01142             {
01143                 if (instance) {
01144                     if (m_dynlib.SetMetadataMulti) {
01145                         typedef const char* pchar;
01146                         pchar* keyvalue=new pchar[2*list.size()+1];
01147                         size_t n=0;
01148                         for (auto&& [key,value]: list) {
01149                             keyvalue[2*n+0]=key.c_str();
01150                             keyvalue[2*n+1]=value;
01151                             n++;
01152                         }
01153                         keyvalue[2*n+0]=nullptr;
01154
01155                         cloudx1_SetMetadataMulti(instance, keyvalue);
01156
01157                         delete[] keyvalue;
01158                     } else {
01159                         for (auto&& [key,value]: list) {
01160                             SetMetadata(key, value);
01161                         }
01162                     }
01163                 } else {
01164                     for (auto&& [key,value]: list) {
01165                         if (value) {
01166                             init_metadata.push_back( {key, value} );
01167                         }
01168                     }
01169                 }
01170             }
01171
01177         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01178             {
01179                 std::vector< std::tuple<std::string,std::string> > values;
01180                 if (m_dynlib.GetMetadataInfos) {
01181                     const char** c_values = m_dynlib.GetMetadataInfos(instance);
01182                     if (c_values) {
```

```
01183                         for (const char** c_value=c_values; *c_value; ) {
01184                             std::string key(*c_value);
01185                             c_value++;
01186                             if (*c_value) {
01187                                 std::string descr(*c_value);
01188                                 c_value++;
01189                                 values.push_back( {key,descr} );
01190                             }
01191                         }
01192                         m_dynlib.FreeMetadataInfos(instance, c_values);
01193                     }
01194                 }
01195                 return values;
01196             }
01197
01201         virtual void OnUpdateThreadStart() {}
01205         virtual void OnUpdateThreadStop() {}
01206
01207 #if CLOUDX1_HAS_WEBSERVER
01221         bool StartWebServer(int http_port, int https_port=0)
01222             {
01223                 assert(instance);
01224                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01225                 webserver = m_dynlib.Webserver(http_port,https_port,instance);
01226                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01227                     return false;
01228                 }
01229                 int web_status=m_dynlib.Webserver_Status(webserver);
01230                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01231                     return false;
01232                 }
01233                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01234                     return false;
01235                 }
01236                 return true;
01237             }
01238
01246         void StopWebServer(int timeout_ms = 1000)
01247             {
01248                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01249                     m_dynlib.Webserver_Stop(webserver, timeout_ms);
01250                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01251                 }
01252             }
01253
01257         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
01258             {
01259                 assert(instance);
01260                 if (!m_dynlib.Webserver_SetAppHealth) return;
01261                 m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
    content.c_str());
01262             }
01266         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01267             {
01268                 assert(instance);
01269                 if (!m_dynlib.Webserver_GetAppHealth) return;
01270                 char* c_contenttype=nullptr;
01271                 char* c_content=nullptr;
01272                 m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01273                 contenttype=c_contenttype;
01274                 content=c_content;
01275                 m_dynlib.Webserver_FreeString(c_contenttype);
01276                 m_dynlib.Webserver_FreeString(c_content);
01277             }
01278 #endif // CLOUDX1_HAS_WEBSERVER
01279
01288         int Terminate()
01289             {
01290                 if (!instance) return 0;
01291                 return m_dynlib.TerminateProcess(instance);
01292             }
01293
01297         void Stop()
01298             {
01299                 if (!instance) return;
01300 #if CLOUDX1_HAS_WEBSERVER
01301                 StopWebServer();
01302 #endif // CLOUDX1_HAS_WEBSERVER
01303                 if (update_thread.joinable()) {
01304                     m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01305                     update_thread.join();
01306                 }
01307                 m_dynlib.ExitProcess(instance);
01308                 instance=nullptr;
01309             }
01310
```

```
01320         unsigned int GetEstimatedDelay()
01321             {
01322                 assert(instance);
01323                 return m_dynlib.GetEstimatedDelay(instance);
01324             }
01325
01338         std::array<float,InputSampleSize>& GetBufferIn()
01339             {
01340                 assert(instance);
01341                 float *buf=m_dynlib.GetBufferIn(instance);
01342                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
01343
01344             }
01356         std::array<float,OutputSampleSize>& GetBufferOut()
01357             {
01358                 assert(instance);
01359                 float *buf= m_dynlib.GetBufferOut(instance);
01360                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
01361             }
01365         void ProcessAudio()
01366             {
01367                 assert(instance);
01368                 m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
    m_dynlib.GetBufferOut(instance));
01369             }
01381         void ProcessAudio_Planar(float const * const *input, float * const *output)
01382             {
01383                 assert(instance);
01384                 m_dynlib.ProcessAudio_Planar(instance, input, output);
01385             }
01386
01398             template<typename T>
01399         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01400             {
01401                 assert(instance);
01402                 std::vector<T> output;
01403                 unsigned int out_offset=0;
01404                 unsigned int in_offset=0;
01405                 unsigned int todo = input.size();
01406                 while (todo>0) {
01407                     unsigned int left = AddAudio(&input[in_offset], todo);
01408                     unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01409                     output.resize(out_offset + out_avail);
01410                     GetAudio(&output[out_offset], out_avail);
01411                     out_offset+=out_avail;
01412                     in_offset += todo-left;
01413                     todo=left;
01414                 }
01415                 return output;
01416             }
01417
01423         class CClient {
01424             public:
01425                 CClient(cloudx1_CInstance* instance, CDynLib& dynlib)
01426                 : m_dynlib(dynlib)
01427                     {
01428                         assert(instance);
01429                         client=m_dynlib.NewClient(instance);
01430                         if (!client) throw std::bad_alloc();
01431                     }
01432                 // non-copyable
01433                 CClient( const CInstance& ) = delete;
01434                 CClient& operator=( const CInstance& ) = delete;
01435                 ~CClient()
01436                     {
01437                         if (client) {
01438                             m_dynlib.DeleteClient(client);
01439                             client=nullptr;
01440                         }
01441                     }
01449                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01450                     {
01451                         assert(client);
01452                         int need_save=0;
01453                         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
    &need_save);
01454                         if (!canswer) return {};
01455                         std::string answer(canswer);
01456                         m_dynlib.FreeJsonAnswer (canswer);
01457                         if (NeedSave) {
01458                             *NeedSave=(need_save!=0);
01459                         }
01460                         return answer;
01461                     }
01462             private:
01463                 CDynLib& m_dynlib;
01464                 cloudx1_CClientInstance *client = nullptr;
```

```
01465         };
01473         std::shared_ptr<CClient> NewClient()
01474             {
01475                 assert(instance);
01476                 return std::make_shared<CClient>(instance, m_dynlib);
01477             }
01478
01489         bool SaveState() {
01490             assert(instance);
01491             if (cloudx1_SaveState(instance)!=0) {
01492                 return false;
01493             }
01494             return true;
01495         }
01496
01497     protected:
01498             template<typename T>
01499         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01500             {
01501                 assert(instance);
01502                 return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01503             }
01504             template<typename T>
01505         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01506             {
01507                 assert(instance);
01508                 return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01509             }
01510
01511     private:
01512         CDynLib& m_dynlib;
01513         cloudx1_CParameters* params = nullptr;
01514         cloudx1_CInstance* instance = nullptr;
01515         std::thread update_thread;
01516         std::vector< std::pair<std::string,std::string> > init_metadata;
01517 #if CLOUDX1_HAS_WEBSERVER
01518         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01519 #endif // CLOUDX1_HAS_WEBSERVER
01520     };
01521
01522 }; // namespace dyn
01523 }; // namespace cloudx1
01524 }; // namespace sound4
01525
01526 // bridge C callbacks to CPresetLoader
01527 extern "C" {
01528     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01529         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01530         std::string content=preset_loader->Read(filename);
01531         return strdup(content.c_str());
01532     }
01533     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01534         if (!content) return;
01535         free(content);
01536     }
01537     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01538         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01539         auto res=preset_loader->Write(filename,content);
01540         return res?0:-1;
01541     }
01542     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01543         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01544         auto res=preset_loader->Exists(filename);
01545         return res?0:-1;
01546     }
01547     static fs_char** cloudx1_custom_getall(void* userdata) {
01548         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01549         auto res=preset_loader->GetAll();
01550         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01551         for (size_t n=0;n<res.size();n++) {
01552             all[n]=fs_strdup(res[n].c_str());
01553         }
01554         all[res.size()]=nullptr;
01555         return all;
01556     }
01557     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01558         if (!all) return;
01559         for (fs_char** one=all;*one;one++) {
01560             free(*one);
01561         }
01562         free(all);
```

```
01563     }
01564     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01565         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01566         auto res=preset_loader->Remove(filename);
01567         return res?0:-1;
01568     }
01569     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01570         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01571         auto res=preset_loader->Rename(from,to);
01572         return res?0:-1;
01573     }
01574
01575 };
```

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 #else
00032     #error "Unsupported OS"
00033 #endif
00034
00035 #include "sound4cl_cdef.h"
00036
00037     // bridge C callbacks to CPresetLoader
00038     extern "C" {
00039         static char *sound4cl_custom_reader(const fs_char *filename, void* userdata);
00040         static void sound4cl_custom_reader_free(char *content, void* userdata);
00041         static int sound4cl_custom_writer(const fs_char *filename, const char *content, void*
      userdata);
00042         static int sound4cl_custom_exists(const fs_char *filename, void* userdata);
00043         static fs_char** sound4cl_custom_getall(void* userdata);
00044         static void sound4cl_custom_getall_free(fs_char** all, void* userdata);
00045         static int sound4cl_custom_remove(const fs_char *filename, void* userdata);
00046         static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00047     };
00048
00052 namespace sound4 {
00053
00057     enum SampleFormat {
00058         INVALID_FORMAT=sound4cl_SampleFormat::INVALID_FORMAT,
00059         S16_LE        =sound4cl_SampleFormat::S16_LE,
00060         S16_BE        =sound4cl_SampleFormat::S16_BE,
00061         S24_LE        =sound4cl_SampleFormat::S24_LE,
00062         S24_BE        =sound4cl_SampleFormat::S24_BE,
00063         S32_LE        =sound4cl_SampleFormat::S32_LE,
00064         S32_BE        =sound4cl_SampleFormat::S32_BE,
00065         F32_LE        =sound4cl_SampleFormat::F32_LE,
00066         F32_BE        =sound4cl_SampleFormat::F32_BE,
00067         S16_NATIVE    =sound4cl_SampleFormat::S16_NATIVE,
00068         S24_NATIVE    =sound4cl_SampleFormat::S24_NATIVE,
00069         S32_NATIVE    =sound4cl_SampleFormat::S32_NATIVE,
00070         F32_NATIVE    =sound4cl_SampleFormat::F32_NATIVE,
00071     };
00072
00076     namespace helper {
00077         #ifdef _WIN32
00083         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00084             if (wstr.empty()) return {};
00085             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00086             std::string str(size, 0);
00087             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00088             return str;
00089         }
00095         static inline std::wstring UTF8ToWString(const std::string& str) {
00096             if (str.empty()) return std::wstring();
00097             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00098             std::wstring wstr(size, 0);
00099             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00100             return wstr;
00101         }
00102         #endif // _WIN32
00103
00109             template<typename T>
00110         struct SampleFormat {
```

```
00111         };
00112             template<>
00113         struct SampleFormat<int16_t> {
00114             const sound4::SampleFormat format = S16_NATIVE;
00115         };
00116             template<>
00117         struct SampleFormat<int32_t> {
00118             const sound4::SampleFormat format = S32_NATIVE;
00119         };
00120             template<>
00121         struct SampleFormat<float> {
00122             const sound4::SampleFormat format = F32_NATIVE;
00123         };
00124 
00130             template <typename T>
00131         class DynFuncHolder {
00132             // Storage definition, depending on OS
00133             #if UNIXLIKE
00134                 using DynFunc_t = void *;
00135             #else
00136                 using DynFunc_t = FARPROC;
00137             #endif
00138         private:
00139             DynFunc_t m_ptr = NULL;
00140         public:
00141             DynFuncHolder() = default;
00142             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00143             operator bool() const { return (m_ptr!=NULL); }
00144             bool IsOk() const { return m_ptr!=NULL; }
00145             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00146         };
00147 
00152         class CDynLoader {
00153             // Storage definition, depending on OS
00154             #if UNIXLIKE
00155                 using DynLib_t = void *;
00156             #else
00157                 using DynLib_t = HMODULE;
00158             #endif
00159             private:
00160                 DynLib_t m_lib {};
00161             public:
00162                 CDynLoader() = default;
00163                 CDynLoader(CDynLoader&&) = default;
00164                 // No copy allowed
00165                 CDynLoader(const CDynLoader&) = delete;
00166                 CDynLoader& operator=(CDynLoader const&) = delete;
00167                 operator bool() const {
00168                     return (m_lib!=NULL);
00169                 }
00170                 bool IsOk() const {
00171                     return (m_lib!=NULL);
00172                 }
00173                 #if UNIXLIKE
00174                     void Close() {
00175                         if (IsOk()) {
00176                             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00177                             dlclose(m_lib);
00178                             #endif // __SANITIZE_ADDRESS__
00179                             m_lib=NULL;
00180                         }
00181                     }
00182                     bool Load(const std::filesystem::path& dynlib)
00183                     {
00184                         Close();
00185                         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00186                         return IsOk();
00187                     }
00188                         template <typename T>
00189                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00190                         auto ptr=dlsym(m_lib, name.c_str());
00191                         if (!ptr) {
00192                             throw std::runtime_error("Missing function in library");
00193                         }
00194                         return DynFuncHolder<T>(ptr);
00195                     }
00196                         template <typename T>
00197                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00198                         auto ptr=dlsym(m_lib, name.c_str());
00199                         return DynFuncHolder<T>(ptr);
00200                     }
00201                     static std::filesystem::path GetThisLibraryPath(void) {
00202                         static Dl_info info;
00203                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
00204                         auto rp=realpath(info.dli_fname, NULL);
00205                         std::filesystem::path p(rp);
00206                         free(rp);
```

```
00207                       return p;
00208                   }
00209               #else
00210                   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
00211                   {
00212                       Close();
00213                       m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00214                       return IsOk();
00215                   }
00216                   void Close() {
00217                       if (IsOk()) {
00218                           // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00219                           // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00220                           #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00221                               FreeLibraryAndExitThread(m_lib, 0);
00222                           #elif defined(SOUND4_CALL_FREELIBRARY)
00223                               FreeLibrary(m_lib);
00224                           #endif // SOUND4_CALL_FREELIBRARY
00225                           m_lib=NULL;
00226                       }
00227                   }
00228                       template <typename T>
00229                   DynFuncHolder<T> GetSymbol(const std::string& name) {
00230                       auto ptr=GetProcAddress(m_lib, name.c_str());
00231                       if (!ptr) {
00232                           throw std::runtime_error("Missing function in library");
00233                       }
00234                       return DynFuncHolder<T>(ptr);
00235                   }
00236                       template <typename T>
00237                   DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00238                       auto ptr=GetProcAddress(m_lib, name.c_str());
00239                       return DynFuncHolder<T>(ptr);
00240                   }
00241                   static std::filesystem::path GetThisLibraryPath(void) {
00242                       static wchar_t path[MAX_PATH]={};
00243                       HMODULE hm = NULL;
00244                       if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00245                               GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00246                               (LPCWSTR) &GetThisLibraryPath, &hm))
00247                       {
00248                           if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00249                               return std::filesystem::path(path);
00250                           }
00251                       }
00252                       return {};
00253                   }
00254               #endif
00255               virtual ~CDynLoader() {
00256                   Close();
00257               }
00258       }; // class CDynLoader
00259   }; // namespace helper
00260
00261
00265   enum LogSeverity {
00266      none     = sound4cl_none,
00267      fatal    = sound4cl_fatal,
00268      error    = sound4cl_error,
00269      warning  = sound4cl_warning,
00270      info     = sound4cl_info,
00271      verbose  = sound4cl_verbose,
00272      verbose2 = sound4cl_verbose2,
00273      verbose3 = sound4cl_verbose3,
00274      verbose4 = sound4cl_verbose4,
00275      verbose5 = sound4cl_verbose5,
00276   };
00277
00281   using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00282
00286   static log_cb_t _log_cb;
00287   extern "C" {
00288       static inline void _log_cb_c(sound4cl_LogSeverity severity, const char *c_msg) {
00289           _log_cb((LogSeverity)severity, std::string(c_msg));
00290       }
00291   }
00292
00299   class CProcessor {
00300   private:
00301       helper::CDynLoader m_lib;
00302       bool m_bOK = false;
00303       std::string m_prefix; // The prefix of functions, depends on the library (cloud_, impact_...)
00304   protected:
00305       friend class CBus;
```

```
00306         friend class CInstance;
00307
00308         // All those wrapped functions have the same signature as there C sound4cl_XXX
00309         helper::DynFuncHolder< decltype(sound4cl_GetVersion             ) > fnGetVersion             ;
00310         helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames   ) > fnGetChunkSizeInFrames   ;
00311         helper::DynFuncHolder< decltype(sound4cl_GetChannelCount        ) > fnGetChannelCount        ;
00312         helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount     ) > fnGetAudioInputCount     ;
00313         helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount    ) > fnGetAudioOutputCount    ;
00314         helper::DynFuncHolder< decltype(sound4cl_GetSampleRate          ) > fnGetSampleRate          ;
00315         helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback      ) > fnSetLoggerCallback      ;
00316         helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity         ) > fnSetLogSeverity         ;
00317         helper::DynFuncHolder< decltype(sound4cl_NewParameters           ) > fnNewParameters          ;
00318         helper::DynFuncHolder< decltype(sound4cl_FreeParameters          ) > fnFreeParameters         ;
00319         helper::DynFuncHolder< decltype(sound4cl_SetParameter            ) > fnSetParameter           ;
00320         helper::DynFuncHolder< decltype(sound4cl_GetParameter            ) > fnGetParameter           ;
00321         helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue      ) > fnFreeParameterValue     ;
00322         helper::DynFuncHolder< decltype(sound4cl_InitProcess             ) > fnInitProcess            ;
00323         helper::DynFuncHolder< decltype(sound4cl_InitProcess2            ) > fnInitProcess2           ;
00324         helper::DynFuncHolder< decltype(sound4cl_TerminateProcess        ) > fnTerminateProcess       ;
00325         helper::DynFuncHolder< decltype(sound4cl_ExitProcess             ) > fnExitProcess            ;
00326         helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread       ) > fnStartUpdateThread      ;
00327         helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread        ) > fnStopUpdateThread       ;
00328         helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady   ) > fnWaitUpdateThreadReady  ;
00329         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio            ) > fnProcessAudio           ;
00330         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar     ) > fnProcessAudio_Planar    ;
00331         helper::DynFuncHolder< decltype(sound4cl_GetBufferIn             ) > fnGetBufferIn            ;
00332         helper::DynFuncHolder< decltype(sound4cl_GetBufferOut            ) > fnGetBufferOut           ;
00333         helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay       ) > fnGetEstimatedDelay      ;
00334         helper::DynFuncHolder< decltype(sound4cl_GetFormatName           ) > fnGetFormatName          ;
00335         helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName       ) > fnGetFormatFromName      ;
00336         helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat      ) > fnGetBytesFromFormat     ;
00337         helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame       ) > fnGetMaxPacketFrame      ;
00338         helper::DynFuncHolder< decltype(sound4cl_AddAudio                ) > fnAddAudio               ;
00339         helper::DynFuncHolder< decltype(sound4cl_AddPadAudio             ) > fnAddPadAudio            ;
00340         helper::DynFuncHolder< decltype(sound4cl_GetOutputCount          ) > fnGetOutputCount         ;
00341         helper::DynFuncHolder< decltype(sound4cl_GetAudio                ) > fnGetAudio               ;
00342         helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom        ) > fnAudioConvertFrom       ;
00343         helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo          ) > fnAudioConvertTo         ;
00344         helper::DynFuncHolder< decltype(sound4cl_StereoToMono            ) > fnStereoToMono           ;
00345         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo            ) > fnMonoToStereo           ;
00346         helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
00347         helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo   ) > fnAudioMonoToLiveStereo  ;
00348         helper::DynFuncHolder< decltype(sound4cl_NewClient               ) > fnNewClient              ;
00349         helper::DynFuncHolder< decltype(sound4cl_DeleteClient            ) > fnDeleteClient           ;
00350         helper::DynFuncHolder< decltype(sound4cl_ProcessJson             ) > fnProcessJson            ;
00351         helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer          ) > fnFreeJsonAnswer         ;
00352         helper::DynFuncHolder< decltype(sound4cl_SaveState               ) > fnSaveState              ;
00353
00354         // only for processes supporting webserver
00355         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp           ) > fnWebserver_tcp          ;
00356         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2          ) > fnWebserver_tcp2         ;
00357         helper::DynFuncHolder< decltype(sound4cl_Webserver               ) > fnWebserver              ;
00358         helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop          ) > fnWebserver_Stop         ;
00359         helper::DynFuncHolder< decltype(sound4cl_Webserver_Status        ) > fnWebserver_Status       ;
00360
00361         // added 2023-03-22
00362         helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar     ) > fnStereoToMono_Planar    ;
00363         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar     ) > fnMonoToStereo_Planar    ;
00364
00365         // added 2023-05-22
00366         helper::DynFuncHolder< decltype(sound4cl_SetMetadata             ) > fnSetMetadata            ;
00367         helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos        ) > fnGetMetadataInfos       ;
00368         helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos       ) > fnFreeMetadataInfos      ;
00369
00370         // added 2023-06-20
00371         helper::DynFuncHolder< decltype(sound4cl_SetPresetManager                ) >
      fnSetPresetManager      ;
00372         helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange      ) >
      fnPresetManager_InformChange      ;
00373
00374         // added 2023-09-21
00375         helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) >
      fnGetPossibleChunkSizeInFrames   ;
00376         helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) >
      fnGetProcessChunkFrames   ;
00377         helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3
      ;
00378
00379         // added 2023-06-26, only for processes supporting bus
00380         helper::DynFuncHolder< decltype(sound4cl_NewBus                  ) > fnNewBus      ;
00381         helper::DynFuncHolder< decltype(sound4cl_FreeBus                 ) > fnFreeBus      ;
00382         helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus          ) > fnSetInstanceBus      ;
00383
00384         // added 2024-02-19
00385         helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti        ) > fnSetMetadataMulti
      ;
00386
```

```
00387         // added 2024-05-23
00388         helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) >
      fnWebserver_SetAppHealth      ;
00389         helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) >
      fnWebserver_GetAppHealth      ;
00390         helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString
      ;
00391
00392             template <typename T>
00393         helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
00394             std::string prefix_name = m_prefix+name;
00395             return m_lib.GetSymbol<T>(prefix_name);
00396         }
00397
00398     public:
00399         CProcessor() = default;
00400
00408         bool Load(const std::filesystem::path& filepath) {
00409             m_lib.Close();
00410             // If a path is given, use it directly and do not try other path
00411             if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
00412                 return false;
00413             } else if (!m_lib.IsOk()) {
00414                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00415                 auto filename = filepath.filename();
00416                 #ifndef _WIN32
00417                 // Linux: if in a bin directory, try ../lib first
00418                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00419                     auto libdir = thisdir.parent_path() / "lib";
00420                     m_lib.Load(libdir / filename);
00421                 }
00422                 #endif // !_WIN32
00423                 if (!m_lib.IsOk()) {
00424                     // Search in the same directory this code is in
00425                     m_lib.Load(thisdir / filename);
00426                 }
00427                 if (!m_lib.IsOk()) {
00428                     // Try current path
00429                     std::error_code ec;
00430                     auto p = std::filesystem::current_path(ec);
00431                     if (!ec) {
00432                         m_lib.Load(p / filename);
00433                     }
00434                 }
00435                 if (!m_lib.IsOk()) {
00436                     return false;
00437                 }
00438             }
00439             // Get information from the library itself
00440             try {
00441                 auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             )
      >("SOUND4_GetProcessInfo"   );
00442                 auto info = GetInfo();
00443                 m_prefix = info->prefix;
00444             } catch (std::runtime_error& ) {
00445                 // Compatibility: search in known libraries without this function
00446                 #ifdef _WIN32
00447                     std::string filename = helper::WStringToUTF8(filepath.stem());
00448                 #else // !_WIN32
00449                     std::string filename = filepath.stem();
00450                     // remove lib prefix
00451                     filename=filename.substr(3);
00452                 #endif // ! _WIN32
00453                 if (filename=="sound4.x1.cloud") {
00454                     m_prefix="cloudx1_";
00455                 } else if (filename=="sound4.impact.cl") {
00456                     m_prefix="impact_";
00457                 } else if (filename=="sound4.bigvoice.cl") {
00458                     m_prefix="bigvoice_";
00459                 } else {
00460                     return false;
00461                 }
00462             }
00463
00464             // Load all C functions
00465             try {
00466                 fnGetVersion              = GetPrefixSymbol< decltype(sound4cl_GetVersion
      ) >("GetVersion"              );
00467                 fnGetChunkSizeInFrames    = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
      ) >("GetChunkSizeInFrames"   );
00468                 fnGetChannelCount         = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
      ) >("GetChannelCount"         );
00469                 fnGetAudioInputCount      = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
      ) >("GetAudioInputCount"     );
00470                 fnGetAudioOutputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
      ) >("GetAudioOutputCount"    );
00471                 fnGetSampleRate           = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
```

```
    ) >("GetSampleRate"          );
00472               fnSetLoggerCallback       = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
    ) >("SetLoggerCallback"      );
00473               fnSetLogSeverity          = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
    ) >("SetLogSeverity"         );
00474               fnNewParameters           = GetPrefixSymbol< decltype(sound4cl_NewParameters
    ) >("NewParameters"          );
00475               fnFreeParameters          = GetPrefixSymbol< decltype(sound4cl_FreeParameters
    ) >("FreeParameters"         );
00476               fnSetParameter            = GetPrefixSymbol< decltype(sound4cl_SetParameter
    ) >("SetParameter"           );
00477               fnGetParameter            = GetPrefixSymbol< decltype(sound4cl_GetParameter
    ) >("GetParameter"           );
00478               fnFreeParameterValue      = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
    ) >("FreeParameterValue"     );
00479               fnInitProcess             = GetPrefixSymbol< decltype(sound4cl_InitProcess
    ) >("InitProcess"            );
00480               fnInitProcess2            = GetPrefixSymbol< decltype(sound4cl_InitProcess2
    ) >("InitProcess2"           );
00481               fnTerminateProcess        = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
    ) >("TerminateProcess"       );
00482               fnExitProcess             = GetPrefixSymbol< decltype(sound4cl_ExitProcess
    ) >("ExitProcess"            );
00483               fnStartUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
    ) >("StartUpdateThread"      );
00484               fnStopUpdateThread        = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
    ) >("StopUpdateThread"       );
00485               fnWaitUpdateThreadReady   = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
    ) >("WaitUpdateThreadReady"  );
00486               fnProcessAudio            = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
    ) >("ProcessAudio"           );
00487               fnProcessAudio_Planar     = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
    ) >("ProcessAudio_Planar"    );
00488               fnGetBufferIn             = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
    ) >("GetBufferIn"            );
00489               fnGetBufferOut            = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
    ) >("GetBufferOut"           );
00490               fnGetEstimatedDelay       = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
    ) >("GetEstimatedDelay"      );
00491               fnGetFormatName           = GetPrefixSymbol< decltype(sound4cl_GetFormatName
    ) >("GetFormatName"          );
00492               fnGetFormatFromName       = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
    ) >("GetFormatFromName"      );
00493               fnGetBytesFromFormat      = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
    ) >("GetBytesFromFormat"     );
00494               fnGetMaxPacketFrame       = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
    ) >("GetMaxPacketFrame"      );
00495               fnAddAudio                = GetPrefixSymbol< decltype(sound4cl_AddAudio
    ) >("AddAudio"               );
00496               fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
    ) >("AddPadAudio"            );
00497               fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
    ) >("GetOutputCount"         );
00498               fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
    ) >("GetAudio"               );
00499               fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
    ) >("AudioConvertFrom"       );
00500               fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
    ) >("AudioConvertTo"         );
00501               fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
    ) >("StereoToMono"           );
00502               fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo
    ) >("MonoToStereo"           );
00503               fnAudioMonoFromLiveStereo = GetPrefixSymbol<
    decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
00504               fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo
    ) >("AudioMonoToLiveStereo"  );
00505               fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient
    ) >("NewClient"              );
00506               fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient
    ) >("DeleteClient"           );
00507               fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson
    ) >("ProcessJson"            );
00508               fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer
    ) >("FreeJsonAnswer"         );
00509               fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState
    ) >("SaveState"              );
00510           } catch (std::runtime_error& ) {
00511               return false;
00512           }
00513           // Webserver : not for all projects
00514           try {
00515               fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp
    ) >("Webserver_tcp"          );
00516               fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2
    ) >("Webserver_tcp2"         );
00517               fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver
```

```
    ) >("Webserver"              );
00518               fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop
    ) >("Webserver_Stop"         );
00519               fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status
    ) >("Webserver_Status"       );
00520           } catch (std::runtime_error& ) {
00521               // Ignored, handler will take care
00522           }
00523           // Stereo/Mono conversion : must be present
00524           try {
00525               // added 2023-03-22
00526               fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar
    ) >("StereoToMono_Planar"    );
00527               fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar
    ) >("MonoToStereo_Planar"    );
00528
00529           } catch (std::runtime_error& ) {
00530               return false;
00531           }
00532           // C functions allowed to be missing
00533           try {
00534               // added 2023-05-22
00535               fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata
    ) >("SetMetadata"            );
00536               fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos
    ) >("GetMetadataInfos"       );
00537               fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos
    ) >("FreeMetadataInfos"      );
00538               // added 2023-06-20
00539               fnSetPresetManager            = GetPrefixSymbol< decltype(sound4cl_SetPresetManager
    ) >("SetPresetManager"             );
00540               fnPresetManager_InformChange  = GetPrefixSymbol<
    decltype(sound4cl_PresetManager_InformChange     ) >("PresetManager_InformChange"    );
00541
00542               // added 2023-09-21
00543               fnGetPossibleChunkSizeInFrames = GetPrefixSymbol<
    decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
00544               fnGetProcessChunkFrames        = GetPrefixSymbol<
    decltype(sound4cl_GetProcessChunkFrames         ) >("GetProcessChunkFrames"         );
00545               fnInitProcess3                 = GetPrefixSymbol< decltype(sound4cl_InitProcess3
    ) >("InitProcess3"                 );
00546           }  catch (std::runtime_error& ) {
00547               // Ignored, handler will take care
00548           }
00549
00550           // bus: not for all projects
00551           try {
00552               // added 2023-06-26
00553               fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus
    ) >("NewBus"                 );
00554               fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus
    ) >("FreeBus"                );
00555               fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus
    ) >("SetInstanceBus"         );
00556           }  catch (std::runtime_error& ) {
00557               // Ignored, handler will take care
00558           }
00559
00560           // Other optional
00561           try {
00562               // added 2024-02-19
00563               fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti
    ) >("SetMetadataMulti"       );
00564
00565           }  catch (std::runtime_error& ) {
00566               // Ignored, handler will take care
00567           }
00568
00569           // Other optional, web server only
00570           try {
00571               // added 2024-05-23
00572               fnWebserver_SetAppHealth    = GetPrefixSymbol<
    decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
00573               fnWebserver_GetAppHealth    = GetPrefixSymbol<
    decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
00574               fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString
    ) >("Webserver_FreeString"       );
00575           }  catch (std::runtime_error& ) {
00576               // Ignored, handler will take care
00577           }
00578
00579           m_bOK=true;
00580           return true;
00581       }
00588       bool IsOk() const {
00589           return m_bOK;
00590       }
```

```
00591
00592         // ------------------------------------------------------------
00599         // ------------------------------------------------------------
00610         void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
00611             { fnAudioConvertFrom(payload, output, nSpl, (sound4cl_SampleFormat)fmt); }
00612
00623         void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
00624             { fnAudioConvertTo(input, payload, nSpl, (sound4cl_SampleFormat)fmt); }
00625
00635         void StereoToMono(const float *input, float *output, size_t nFrame)
00636             { fnStereoToMono(input, output, nFrame); }
00637
00647         void MonoToStereo(const float *input, float *output, size_t nFrame)
00648             { fnMonoToStereo(input, output, nFrame); }
00649
00660         void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame)
00661             { fnStereoToMono_Planar(inputL, inputR, output, nFrame); }
00662
00673         void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame)
00674             { fnMonoToStereo_Planar(input, outputL, outputR, nFrame); }
00675
00682         void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
00683             { fnAudioMonoFromLiveStereo(payload, output); }
00684
00691         void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
00692             { fnAudioMonoToLiveStereo(input, payload); }
00693
00701         std::string GetVersion()
00702             { return std::string(fnGetVersion()); }
00703
00714         unsigned int GetChunkSizeInFrames()
00715             { return fnGetChunkSizeInFrames(); }
00716
00725         std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00726             {
00727                 std::vector<unsigned int> list;
00728                 if (fnGetPossibleChunkSizeInFrames) {
00729                     for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
00730                         list.push_back(*src);
00731                     }
00732                 } else {
00733                     list.push_back(fnGetChunkSizeInFrames());
00734                 }
00735                 return list;
00736             }
00737
00743         unsigned int GetChannelCount()
00744             { return fnGetChannelCount(); }
00745
00753         unsigned int GetAudioInputCount()
00754             { return fnGetAudioInputCount(); }
00755
00763         unsigned int GetAudioOutputCount()
00764             { return fnGetAudioOutputCount(); }
00765
00773         unsigned int GetSampleRate()
00774             { return fnGetSampleRate(); }
00775
00782         std::string GetFormatName( const SampleFormat fmt)
00783             { return std::string(fnGetFormatName((sound4cl_SampleFormat)fmt)); }
00784
00791         SampleFormat GetFormatFromName( const std::string& name)
00792             { return SampleFormat(fnGetFormatFromName(name.c_str())); }
00793
00800         unsigned int GetBytesFromFormat( const SampleFormat fmt)
00801             { return fnGetBytesFromFormat((sound4cl_SampleFormat)fmt); }
00802
00806         void SetLogSeverity(LogSeverity severity)
00807             { fnSetLogSeverity((sound4cl_LogSeverity)severity); }
00808
00812         void SetLoggerCallback(log_cb_t cb)
00813             { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
00814     };
00815
00821     class CBus {
00822         public:
00823             CBus(CProcessor& dynlib)
00824                 : m_dynlib(dynlib)
00825                 {
00826                     if (m_dynlib.fnNewBus) {
00827                         m_bus=m_dynlib.fnNewBus();
00828                     }
00829                 }
00830             ~CBus()
00831                 {
00832                     if (m_bus && m_dynlib.fnFreeBus) {
```

```
00833                       m_dynlib.fnFreeBus(m_bus);
00834                   }
00835                   m_bus = nullptr;
00836               }
00837       protected:
00838           friend class CInstance;
00839           sound4cl_CBus *Get() const { return m_bus; }
00840       private:
00841           CProcessor& m_dynlib;
00842           sound4cl_CBus *m_bus;
00843   };
00844
00850   class CPresetLoader {
00851   public:
00852       CPresetLoader() = default;
00853       virtual ~CPresetLoader() = default;
00854
00855       virtual bool IsReadOnly() = 0;
00856       virtual bool Exists(const std::filesystem::path &name) = 0;
00857       virtual bool Remove(const std::filesystem::path &name) = 0;
00858       virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00859       virtual std::vector<std::filesystem::path> GetAll() = 0;
00860       virtual std::string Read(const std::filesystem::path &filename) = 0;
00861       virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00862   };
00863   enum PresetChange_Kind {
00864       change_kind_created  = sound4cl_PresetChange_Kind::change_kind_created,
00865       change_kind_modified = sound4cl_PresetChange_Kind::change_kind_modified,
00866       change_kind_deleted  = sound4cl_PresetChange_Kind::change_kind_deleted,
00867   };
00873   class CInstance {
00874   public:
00880       CInstance(CProcessor& dynlib)
00881       : m_dynlib(dynlib)
00882           {
00883               params = m_dynlib.fnNewParameters();
00884               if (!params) throw std::bad_alloc();
00885           }
00886
00895       CInstance(CProcessor& dynlib, void* _instance)
00896       : m_dynlib(dynlib)
00897       , instance(reinterpret_cast<sound4cl_CInstance*>(_instance))
00898       , m_bOwned(false)
00899           {
00900           }
00901       // non-copyable
00902       CInstance( const CInstance& ) = delete;
00903       CInstance& operator=( const CInstance& ) = delete;
00907       virtual ~CInstance()
00908           {
00909               Stop();
00910               if (params) m_dynlib.fnFreeParameters(params);
00911           }
00915      bool IsOk() const { return instance!=nullptr; }
00925       void SetParam(const std::string& name, const std::string& value)
00926           {
00927               assert(params);
00928               m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
00929           }
00936       std::string GetParam(const std::string& name)
00937           {
00938               assert(params);
00939               auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
00940               std::string ret(c_value);
00941               m_dynlib.fnFreeParameterValue(c_value);
00942               return ret;
00943           }
00944
00953       void SetBus(const CBus& bus) {
00954           assert(params);
00955           if (m_dynlib.fnSetInstanceBus) {
00956               m_dynlib.fnSetInstanceBus(params,bus.Get());
00957           }
00958       }
00959
00970       void SetPresetManager(CPresetLoader *preset_manager)
00971           {
00972               if (!preset_manager) return;
00973               if (!m_dynlib.fnSetPresetManager) return;
00974               assert(params);
00975               m_dynlib.fnSetPresetManager(params,
00976                   sound4cl_custom_reader,
00977                   sound4cl_custom_reader_free,
00978                   sound4cl_custom_writer,
00979                   sound4cl_custom_exists,
00980                   sound4cl_custom_getall,
00981                   sound4cl_custom_getall_free,
```

```
00982                     sound4cl_custom_remove,
00983                     sound4cl_custom_rename,
00984                     preset_manager->IsReadOnly(),
00985                     preset_manager
00986                 );
00987             }
00988
01011         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)
01012             {
01013                 assert(!instance);
01014                 std::string l_save_path;
01015                 if (!save_path.empty()) {
01016                     l_save_path = save_path.u8string();
01017                 }
01018                 if (frames_per_chunk==0) {
01019                     frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
01020                 }
01021                 if (m_dynlib.fnInitProcess3) {
01022                     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);
01023                 } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
01024                     return false;
01025                 } else {
01026                     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);
01027                 }
01028                 if (!instance) return false;
01029                 if (!init_metadata.empty()) {
01030                     for (auto&& [key,value]: init_metadata) {
01031                         m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
01032                     }
01033                     init_metadata.clear();
01034                 }
01035                 update_thread = std::thread([this, json_port](){
01036                     OnUpdateThreadStart();
01037                     m_dynlib.fnStartUpdateThread(instance, json_port);
01038                     OnUpdateThreadStop();
01039                 });
01040                 int timeout=5000;
01041                 #ifdef DEBUG
01042                     timeout=60*1000; // For debugger pauses
01043                 #endif
01044                 if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
01045                     return false;
01046                 }
01047                 return true;
01048             }
01049
01057         unsigned int GetChunkFrames() {
01058             if (instance && m_dynlib.fnGetProcessChunkFrames) {
01059                 return m_dynlib.fnGetProcessChunkFrames(instance);
01060             }
01061             return m_dynlib.fnGetChunkSizeInFrames();
01062         }
01063
01071         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01072             {
01073                 if (m_dynlib.fnPresetManager_InformChange) {
01074                     m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(),
      (sound4cl_PresetChange_Kind) change_kind);
01075                 }
01076             }
01077
01089         void SetMetadata(const std::string& key, const char* value)
01090             {
01091                 if (m_dynlib.fnSetMetadata) {
01092                     if (instance) {
01093                         m_dynlib.fnSetMetadata(instance, key.c_str(), value);
01094                     } else {
01095                         init_metadata.push_back( {key, value} );
01096                     }
01097                 }
01098             }
01099
01109         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01110             {
01111                 if (instance) {
01112                     if (m_dynlib.fnSetMetadataMulti) {
01113                         typedef const char* pchar;
01114                         pchar* keyvalue=new pchar[2*list.size()+1];
01115                         size_t n=0;
01116                         for (auto&& [key,value]: list) {
01117                             keyvalue[2*n+0]=key.c_str();
01118                             keyvalue[2*n+1]=value;
```

```
01119                           n++;
01120                       }
01121                       keyvalue[2*n+0]=nullptr;
01122
01123                       sound4cl_SetMetadataMulti(instance, keyvalue);
01124
01125                       delete[] keyvalue;
01126                   } else {
01127                       for (auto&& [key,value]: list) {
01128                           SetMetadata(key, value);
01129                       }
01130                   }
01131               } else {
01132                   for (auto&& [key,value]: list) {
01133                       if (value) {
01134                           init_metadata.push_back( {key, value} );
01135                       }
01136                   }
01137               }
01138           }
01139
01145       std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01146           {
01147               std::vector< std::tuple<std::string,std::string> > values;
01148               if (m_dynlib.fnGetMetadataInfos) {
01149                   const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
01150                   if (c_values) {
01151                       for (const char** c_value=c_values; *c_value; ) {
01152                           std::string key(*c_value);
01153                           c_value++;
01154                           if (*c_value) {
01155                               std::string descr(*c_value);
01156                               c_value++;
01157                               values.push_back( {key,descr} );
01158                           }
01159                       }
01160                       m_dynlib.fnFreeMetadataInfos(instance, c_values);
01161                   }
01162               }
01163               return values;
01164           }
01165
01169       virtual void OnUpdateThreadStart() {}
01173       virtual void OnUpdateThreadStop() {}
01174
01175
01189       bool StartWebServer(int http_port, int https_port=0)
01190           {
01191               assert(instance);
01192               assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01193               if (!m_dynlib.fnWebserver) {
01194                   return false;
01195               }
01196               webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
01197               if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01198                   return false;
01199               }
01200               int web_status=m_dynlib.fnWebserver_Status(webserver);
01201               if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01202                   return false;
01203               }
01204               if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01205                   return false;
01206               }
01207               return true;
01208           }
01209
01217           void StopWebServer(int timeout_ms = 1000)
01218           {
01219               if (!m_dynlib.fnWebserver_Stop) return;
01220               if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01221                   m_dynlib.fnWebserver_Stop(webserver, timeout_ms);
01222                   webserver = SOUND4_INVALID_WEBSERVER_ID;
01223               }
01224           }
01225
01229       void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
01230           {
01231               assert(instance);
01232               if (!m_dynlib.fnWebserver_SetAppHealth) return;
01233               m_dynlib.fnWebserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
    content.c_str());
01234           }
01238       void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01239           {
01240               assert(instance);
```

```
01241                 if (!m_dynlib.fnWebserver_GetAppHealth) return;
01242                 char* c_contenttype=nullptr;
01243                 char* c_content=nullptr;
01244                 m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01245                 contenttype=c_contenttype;
01246                 content=c_content;
01247                 m_dynlib.fnWebserver_FreeString(c_contenttype);
01248                 m_dynlib.fnWebserver_FreeString(c_content);
01249             }
01258         int Terminate()
01259             {
01260                 if (!instance) return 0;
01261                 return m_dynlib.fnTerminateProcess(instance);
01262             }
01263 
01269         void Stop()
01270             {
01271                 if (!instance) return;
01272                 if (m_bOwned) {
01273                     StopWebServer();
01274                     if (update_thread.joinable()) {
01275                         m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
01276                         update_thread.join();
01277                     }
01278                     m_dynlib.fnExitProcess(instance);
01279                 }
01280                 instance=nullptr;
01281             }
01282 
01292         unsigned int GetEstimatedDelay()
01293             {
01294                 assert(instance);
01295                 return m_dynlib.fnGetEstimatedDelay(instance);
01296             }
01297 
01310         float* GetBufferIn()
01311             {
01312                 assert(instance);
01313                 return m_dynlib.fnGetBufferIn(instance);
01314 
01315             }
01327         float* GetBufferOut()
01328             {
01329                 assert(instance);
01330                 return m_dynlib.fnGetBufferOut(instance);
01331             }
01335         void ProcessAudio()
01336             {
01337                 assert(instance);
01338                 m_dynlib.fnProcessAudio(instance, GetBufferIn(), GetBufferOut());
01339             }
01351         void ProcessAudio_Planar(float const * const *input, float * const *output)
01352             {
01353                 assert(instance);
01354                 m_dynlib.fnProcessAudio_Planar(instance, input, output);
01355             }
01356 
01368             template<typename T>
01369         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01370             {
01371                 assert(instance);
01372                 std::vector<T> output;
01373                 unsigned int out_offset=0;
01374                 unsigned int in_offset=0;
01375                 unsigned int todo = input.size();
01376                 while (todo>0) {
01377                     unsigned int left = AddAudio(&input[in_offset], todo);
01378                     unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
01379                     output.resize(out_offset + out_avail);
01380                     GetAudio(&output[out_offset], out_avail);
01381                     out_offset+=out_avail;
01382                     in_offset += todo-left;
01383                     todo=left;
01384                 }
01385                 return output;
01386             }
01387 
01393         class CClient {
01394             public:
01395                 CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
01396                 : m_dynlib(dynlib)
01397                     {
01398                         assert(instance);
01399                         client=m_dynlib.fnNewClient(instance);
01400                         if (!client) throw std::bad_alloc();
01401                     }
01402                 // non-copyable
```

```
01403                 CClient( const CInstance& ) = delete;
01404                 CClient& operator=( const CInstance& ) = delete;
01405                 ~CClient()
01406                     {
01407                         if (client) {
01408                             m_dynlib.fnDeleteClient(client);
01409                             client=nullptr;
01410                         }
01411                     }
01419                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01420                     {
01421                         assert(client);
01422                         int need_save=0;
01423                         const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(),
      &need_save);
01424                         if (!canswer) return {};
01425                         std::string answer(canswer);
01426                         m_dynlib.fnFreeJsonAnswer (canswer);
01427                         if (NeedSave) {
01428                             *NeedSave=(need_save!=0);
01429                         }
01430                         return answer;
01431                     }
01432             private:
01433                 CProcessor& m_dynlib;
01434                 sound4cl_CClientInstance *client = nullptr;
01435         };
01443         std::shared_ptr<CClient> NewClient()
01444             {
01445                 assert(instance);
01446                 return std::make_shared<CClient>(instance, m_dynlib);
01447             }
01448
01459         bool SaveState() {
01460             assert(instance);
01461             if (!m_dynlib.fnSaveState || m_dynlib.fnSaveState(instance)!=0) {
01462                 return false;
01463             }
01464             return true;
01465         }
01466
01467     protected:
01468             template<typename T>
01469         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01470             {
01471                 assert(instance);
01472                 return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01473             }
01474             template<typename T>
01475         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01476             {
01477                 assert(instance);
01478                 return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01479             }
01480
01481     private:
01482         CProcessor& m_dynlib;
01483         sound4cl_CParameters* params = nullptr;
01484         sound4cl_CInstance* instance = nullptr;
01485         std::thread update_thread;
01486         std::vector< std::pair<std::string,std::string> > init_metadata;
01487         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01488         bool m_bOwned = true;
01489     };
01490
01491 }; // namespace sound4
01492
01493
01494 // bridge C callbacks to CPresetLoader
01495 extern "C" {
01496     static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
01497         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01498         std::string content=preset_loader->Read(filename);
01499         return strdup(content.c_str());
01500     }
01501     static void sound4cl_custom_reader_free(char *content, void* userdata) {
01502         if (!content) return;
01503         free(content);
01504     }
01505     static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01506         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01507         auto res=preset_loader->Write(filename,content);
01508         return res?0:-1;
01509     }
01510     static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
```

```
01511         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01512         auto res=preset_loader->Exists(filename);
01513         return res?0:-1;
01514     }
01515     static fs_char** sound4cl_custom_getall(void* userdata) {
01516         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01517         auto res=preset_loader->GetAll();
01518         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01519         for (size_t n=0;n<res.size();n++) {
01520             all[n]=fs_strdup(res[n].c_str());
01521         }
01522         all[res.size()]=nullptr;
01523         return all;
01524     }
01525     static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
01526         if (!all) return;
01527         for (fs_char** one=all;*one;one++) {
01528             free(*one);
01529         }
01530         free(all);
01531     }
01532     static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
01533         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01534         auto res=preset_loader->Remove(filename);
01535         return res?0:-1;
01536     }
01537     static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01538         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01539         auto res=preset_loader->Rename(from,to);
01540         return res?0:-1;
01541     }
01542
01543 };
```

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

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 62 of file sound4cl_cdef.h.

15.27.1.2 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 63 of file sound4cl_cdef.h.

15.27.1.3 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 64 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.

15.28 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 #else // !_WIN32
00026     #define fs_strdup strdup
00027     typedef char fs_char;
00028 #endif // !_WIN32
00029
00030 #ifdef __cplusplus
00031 extern "C" {
00032 #endif // __cplusplus
00033
00034 struct sound4cl_SOUND4_ProcessInfo {
00035     size_t struct_size;
00036     const char *name;
00037     const char *prefix;
00038     const char *shortname;
00039     const char *version;
00040 };
00041 enum sound4cl_LogSeverity {
00042     sound4cl_none     = 0,
00043     sound4cl_fatal    = 1,
00044     sound4cl_error    = 2,
00045     sound4cl_warning  = 3,
00046     sound4cl_info     = 4,
00047     sound4cl_verbose  = 5,
00048     sound4cl_verbose2 = 6,
```

```
00049     sound4cl_verbose3 = 7,
00050     sound4cl_verbose4 = 8,
00051     sound4cl_verbose5 = 9,
00052 };
00053 struct sound4cl_CInstance;
00054 struct sound4cl_CParameters;
00055 struct sound4cl_CBus;
00056 struct sound4cl_CClientInstance;
00057 enum sound4cl_PresetChange_Kind {
00058     change_kind_created = 1,
00059     change_kind_modified   ,
00060     change_kind_deleted    ,
00061 };
00062 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
00063 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
00064 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
00065
00069 enum sound4cl_SampleFormat {
00070     INVALID_FORMAT=0,
00071     S16_LE,
00072     S16_BE,
00073     S24_LE,
00074     S24_BE,
00075     S32_LE,
00076     S32_BE,
00077     F32_LE,
00078     F32_BE,
00079
00080     // Native definitions, only links to the physical ones
00081     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00082         S16_NATIVE=S16_LE,
00083         S24_NATIVE=S24_LE,
00084         S32_NATIVE=S32_LE,
00085         F32_NATIVE=F32_LE,
00086     #else
00087         S16_NATIVE=S16_BE,
00088         S24_NATIVE=S24_BE,
00089         S32_NATIVE=S32_BE,
00090         F32_NATIVE=F32_BE,
00091     #endif
00092 };
00093
00094 const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
00095 const char *sound4cl_GetVersion();
00096 unsigned int sound4cl_GetChunkSizeInFrames();
00097 unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
00098 unsigned int sound4cl_GetChannelCount();
00099 unsigned int sound4cl_GetAudioInputCount();
00100 unsigned int sound4cl_GetAudioOutputCount();
00101 unsigned int sound4cl_GetSampleRate();
00102 typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
00103 void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
00104 void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
00105 struct sound4cl_CParameters *sound4cl_NewParameters();
00106 void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
00107 void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
00108 const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
00109 void sound4cl_FreeParameterValue(const char *value);
00110 struct sound4cl_CInstance *sound4cl_InitProcess(
00111         const char *LoginKey,
00112         const char *RadioName,
00113         const char *Access_Key_ID,
00114         const char *Access_Key_Secret,
00115         const char *save_path
00116     );
00117 struct sound4cl_CInstance *sound4cl_InitProcess2(
00118         const char *LoginKey,
00119         const char *RadioName,
00120         const char *Access_Key_ID,
00121         const char *Access_Key_Secret,
00122         const char *save_path,
00123         const struct sound4cl_CParameters *parameters
00124     );
00125 struct sound4cl_CInstance *sound4cl_InitProcess3(
00126         const char *LoginKey,
00127         const char *RadioName,
00128         const char *Access_Key_ID,
00129         const char *Access_Key_Secret,
00130         const char *save_path,
00131         const struct sound4cl_CParameters *parameters,
00132         unsigned int frames_per_chunk
00133     );
00134 unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
00135 int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
00136 void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
00137 struct sound4cl_CBus* sound4cl_NewBus();
00138 void sound4cl_FreeBus(struct sound4cl_CBus *bus);
```

```
00139 void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
00140 typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
00141 typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
00142 typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00143 typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
00144 typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
00145 typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
00146 typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
00147 typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00148 void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
00149     , sound4cl_storage_reader reader, sound4cl_storage_reader_free
00150     , sound4cl_storage_writer writer
00151     , sound4cl_storage_exists exists
00152     , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
00153     , sound4cl_storage_remove remove
00154     , sound4cl_storage_rename rename
00155     , int IsReadOnly
00156     , void* userdata
00157     );
00158 void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
      *relative_path, enum sound4cl_PresetChange_Kind change_kind);
00159 void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
00160 void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
00161 const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
00162 void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
00163 void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
00164 void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
00165 int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
00166 void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
00167 void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
00168     , float const * const *input
00169     , float * const *output);
00170 float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
00171 float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
00172 unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
00173 const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
00174 enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
00175 unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
00176 unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
00177 unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
      int nFrame, enum sound4cl_SampleFormat fmt );
00178 unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
00179 unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
00180 unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
      max_nFrame, enum sound4cl_SampleFormat fmt );
00181 void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00182 void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00183 void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
00184 void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
00185 void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame);
00186 void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
00187 void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00188 void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00189 struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
00190 void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
00191 const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
      *need_save);
00192 void sound4cl_FreeJsonAnswer(const char *json_str);
00193 int sound4cl_SaveState(struct sound4cl_CInstance *instance);
00194 uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port);
00195 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);
00196 uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
      sound4cl_CInstance *instance);
00197 int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
00198 int sound4cl_Webserver_Status(uint64_t id);
00199 void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
      contenttype, const char* content);
00200 void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
      contenttype, char** content);
00201 void sound4cl_Webserver_FreeString(char* str);
00202
00203 #ifdef __cplusplus
00204 } // extern "C"
00205 #endif // __cplusplus
```

Index

_log_cb
  sound4, 82
  sound4::cloudx1, 88
  sound4::cloudx1::dyn, 97
_log_cb_c
  sound4, 81
  sound4::cloudx1, 84
  sound4::cloudx1::dyn, 91
~CBus
  sound4::CBus, 111
~CClient
  sound4::CInstance::CClient, 113
  sound4::cloudx1::CInstance::CClient, 114
  sound4::cloudx1::dyn::CInstance::CClient, 116
~CDynLoader
  sound4::cloudx1::dyn::helper::CDynLoader, 129
  sound4::helper::CDynLoader, 131
~CInstance
  sound4::CInstance, 135
  sound4::cloudx1::CInstance, 147
  sound4::cloudx1::dyn::CInstance, 159
~CPresetLoader
  sound4::cloudx1::CPresetLoader, 170
  sound4::cloudx1::dyn::CPresetLoader, 172
  sound4::CPresetLoader, 174
~MyPresetManager
  MyPresetManager, 186

AddAudio
  sound4::CInstance, 135
  sound4::cloudx1::CInstance, 147
  sound4::cloudx1::dyn::CDynLib, 119
  sound4::cloudx1::dyn::CInstance, 159
AddPadAudio
  sound4::cloudx1::dyn::CDynLib, 119
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, 119
  sound4::cloudx1::dyn::helper, 100
  sound4::cloudx1::helper, 105
AudioConvertTo
  Audio convert helpers (C++), 34
  sound4::cloudx1::dyn::CDynLib, 119
  sound4::cloudx1::dyn::helper, 100
  sound4::cloudx1::helper, 105
AudioMonoFromLiveStereo
  Audio convert helpers (C++), 35
  sound4::cloudx1::dyn::CDynLib, 119
  sound4::cloudx1::dyn::helper, 101
  sound4::cloudx1::helper, 105
AudioMonoToLiveStereo
  Audio convert helpers (C++), 35
  sound4::cloudx1::dyn::CDynLib, 119
  sound4::cloudx1::dyn::helper, 101
  sound4::cloudx1::helper, 106

CBus
  sound4::CBus, 111
  sound4::CProcessor, 181

CClient
sound4::CInstance::CClient, 113
sound4::cloudx1::CInstance::CClient, 114
sound4::cloudx1::dyn::CInstance::CClient, 115
CDynLib
sound4::cloudx1::dyn::CDynLib, 118
CDynLoader
sound4::cloudx1::dyn::helper::CDynLoader, 129
sound4::helper::CDynLoader, 131
change_kind_created
Custom preset managagement, 48
sound4, 81
change_kind_deleted
Custom preset managagement, 48
sound4, 81
change_kind_modified
Custom preset managagement, 48
sound4, 81
CHANNEL_COUNT
sound4.x1.cloud-pipe.cpp, 203
CHUNK_SIZE
sound4.x1.cloud-pipe.cpp, 203
ChunkFrames
sound4::cloudx1, 88
sound4::cloudx1::dyn, 97
CInstance
sound4::CBus, 112
sound4::CInstance, 134, 135
sound4::cloudx1::CInstance, 147
sound4::cloudx1::dyn::CInstance, 159
sound4::CProcessor, 181
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, 129
sound4::helper::CDynLoader, 132
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, 168
CLOUDX1_CHANNEL_COUNT
Global setup, 54
cloudx1_CInstance, 169
cloudx1_CParameters, 169
cloudx1_custom_exists
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_getall
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_getall_free
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_reader
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_reader_free
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 233
cloudx1_custom_remove
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 234
cloudx1_custom_rename
sound4.x1.cloud.hpp, 222
sound4.x1.cloud_dyn.hpp, 234
cloudx1_custom_writer
sound4.x1.cloud.hpp, 223
sound4.x1.cloud_dyn.hpp, 234
CLOUDX1_DECL
sound4.x1.cloud.h, 215
cloudx1_DeleteClient
Client's JSon config interfacing, 42
cloudx1_ExitProcess
Process creation/destruction, 61
cloudx1_FreeJsonAnswer
Client's JSon config interfacing, 42
cloudx1_FreeMetadataInfos
Metadata Management, 59
cloudx1_FreeParameters
Process creation/destruction, 61
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, 59
cloudx1_GetOutputCount
Audio managing (any audio), 41
cloudx1_GetParameter
Process creation/destruction, 62
cloudx1_GetPossibleChunkSizeInFrames
Global setup, 57
cloudx1_GetProcessChunkFrames
Process creation/destruction, 62
cloudx1_GetSampleRate
Global setup, 57
cloudx1_GetVersion
Global setup, 57
CLOUDX1_HAS_CLOUDBUS
sound4.x1.cloud.h, 215
CLOUDX1_HAS_WEBSERVER
sound4.x1.cloud.h, 216
cloudx1_InitProcess
Process creation/destruction, 62
cloudx1_InitProcess2
Process creation/destruction, 63
cloudx1_InitProcess3
Process creation/destruction, 63
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, 64
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_SetLoggerCallback
Global setup, 58
cloudx1_SetLogSeverity
Global setup, 58
cloudx1_SetMetadata
Metadata Management, 60
cloudx1_SetMetadataMulti
Metadata Management, 60
cloudx1_SetParameter
Process creation/destruction, 64
cloudx1_SetPresetManager
Custom preset managagement, 49
cloudx1_StartUpdateThread
Update Management, 72
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, 64
cloudx1_WaitUpdateThreadReady
Update Management, 73
cloudx1_Webserver
Web server, 75
cloudx1_Webserver_FreeString
Web server, 75
cloudx1_Webserver_GetAppHealth
Web server, 75
cloudx1_Webserver_SetAppHealth
Web server, 76
cloudx1_Webserver_Status
Web server, 76
cloudx1_Webserver_Stop
Web server, 77
cloudx1_Webserver_tcp
Web server, 77
cloudx1_Webserver_tcp2

Web server, 77
CPresetLoader
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174
CProcessor
sound4::CProcessor, 176
Create
sound4::CInstance, 135
sound4::cloudx1::CInstance, 147
sound4::cloudx1::dyn::CInstance, 159
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, 120
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, 193
doc_gencpp.h, 193
doc_json.h, 193
doc_main.h, 194
doc_params.h, 194
DynFuncHolder
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 182
sound4::helper::DynFuncHolder< T >, 184

Environment variables and parameters, 13
error
Global setup, 55
sound4, 80
Examples, 52
Exists
MyPresetManager, 186
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174
ExitProcess
sound4::cloudx1::dyn::CDynLib, 120

F32_BE
Audio managing (any audio), 38
sound4, 81
F32_LE
Audio managing (any audio), 38
sound4, 81
F32_NATIVE
Audio managing (any audio), 38
sound4, 81
fatal
Global setup, 55
sound4, 80
FreeJsonAnswer
sound4::cloudx1::dyn::CDynLib, 120
FreeMetadataInfos
sound4::cloudx1::dyn::CDynLib, 120
FreeParameters
sound4::cloudx1::dyn::CDynLib, 120
FreeParameterValue
sound4::cloudx1::dyn::CDynLib, 120
fs_char
Custom preset managagement, 48
sound4cl_cdef.h, 264

Generic C++ access to SOUND4 .CL librairies, 3
Get
sound4::CBus, 112
GetAll
MyPresetManager, 186
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174
GetAudio
sound4::CInstance, 137
sound4::cloudx1::CInstance, 148
sound4::cloudx1::dyn::CDynLib, 120
sound4::cloudx1::dyn::CInstance, 160
GetAudioInputCount
sound4::cloudx1, 84
sound4::cloudx1::dyn, 91
sound4::cloudx1::dyn::CDynLib, 121
sound4::CProcessor, 176
GetAudioOutputCount
sound4::cloudx1, 84
sound4::cloudx1::dyn, 92
sound4::cloudx1::dyn::CDynLib, 121
sound4::CProcessor, 176
GetBufferIn
sound4::CInstance, 137
sound4::cloudx1::CInstance, 148
sound4::cloudx1::dyn::CDynLib, 121
sound4::cloudx1::dyn::CInstance, 160
GetBufferOut
sound4::CInstance, 137
sound4::cloudx1::CInstance, 149
sound4::cloudx1::dyn::CDynLib, 121
sound4::cloudx1::dyn::CInstance, 160
GetBytesFromFormat
sound4::cloudx1, 84

sound4::cloudx1::dyn, 92
sound4::cloudx1::dyn::CDynLib, 121
sound4::CProcessor, 177
GetChannelCount
sound4::cloudx1, 85
sound4::cloudx1::dyn, 92
sound4::cloudx1::dyn::CDynLib, 121
sound4::CProcessor, 178
GetChunkFrames
sound4::CInstance, 138
sound4::cloudx1::CInstance, 149
sound4::cloudx1::dyn::CInstance, 161
GetChunkSizeInFrames
sound4::cloudx1, 85
sound4::cloudx1::dyn, 93
sound4::cloudx1::dyn::CDynLib, 121
sound4::CProcessor, 178
GetEnv
sound4.x1.cloud-pipe-cxx.cpp, 194
sound4.x1.cloud-pipe-dyn.cpp, 198
sound4cl-pipe.cpp, 208
GetEstimatedDelay
sound4::CInstance, 138
sound4::cloudx1::CInstance, 149
sound4::cloudx1::dyn::CDynLib, 122
sound4::cloudx1::dyn::CInstance, 161
GetFormatFromName
sound4::cloudx1, 85
sound4::cloudx1::dyn, 93
sound4::cloudx1::dyn::CDynLib, 122
sound4::CProcessor, 178
GetFormatName
sound4::cloudx1, 86
sound4::cloudx1::dyn, 94
sound4::cloudx1::dyn::CDynLib, 122
sound4::CProcessor, 179
GetMaxPacketFrame
sound4::cloudx1::dyn::CDynLib, 122
GetMetadataInfos
sound4::CInstance, 138
sound4::cloudx1::CInstance, 150
sound4::cloudx1::dyn::CDynLib, 122
sound4::cloudx1::dyn::CInstance, 161
GetOutputCount
sound4::cloudx1::dyn::CDynLib, 122
GetParam
sound4::CInstance, 139
sound4::cloudx1::CInstance, 150
sound4::cloudx1::dyn::CInstance, 162
GetParameter
sound4::cloudx1::dyn::CDynLib, 122
GetPossibleChunkSizeInFrames
sound4::cloudx1, 86
sound4::cloudx1::dyn, 94
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 179
GetPrefixSymbol
sound4::CProcessor, 179
GetProcessChunkFrames
sound4::cloudx1::dyn::CDynLib, 123
GetSampleRate
sound4::cloudx1, 86
sound4::cloudx1::dyn, 95
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 180
GetSymbol
sound4::cloudx1::dyn::helper::CDynLoader, 129
sound4::helper::CDynLoader, 132
GetSymbol_safe
sound4::cloudx1::dyn::helper::CDynLoader, 129
sound4::helper::CDynLoader, 132
GetThisLibraryPath
sound4::cloudx1::dyn::helper::CDynLoader, 129
sound4::helper::CDynLoader, 132
GetVersion
sound4::cloudx1, 87
sound4::cloudx1::dyn, 95
sound4::cloudx1::dyn::CDynLib, 123
sound4::CProcessor, 180
GetWebServerAppHealth
sound4::CInstance, 139
sound4::cloudx1::CInstance, 150
sound4::cloudx1::dyn::CInstance, 162
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, 80
InitProcess
sound4::cloudx1::dyn::CDynLib, 123

InitProcess2
sound4::cloudx1::dyn::CDynLib, 123
InitProcess3
sound4::cloudx1::dyn::CDynLib, 123
InputChannels
sound4::cloudx1, 88
sound4::cloudx1::dyn, 97
InputSampleSize
sound4::cloudx1, 88
sound4::cloudx1::dyn, 98
INVALID_FORMAT
Audio managing (any audio), 38
sound4, 81
IsOk
sound4::CInstance, 139
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CDynLib, 118
sound4::cloudx1::dyn::CInstance, 162
sound4::cloudx1::dyn::helper::CDynLoader, 130
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 182
sound4::CProcessor, 180
sound4::helper::CDynLoader, 132
sound4::helper::DynFuncHolder< T >, 184
IsReadOnly
MyPresetManager, 187
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174

JSON syntax, 5
JSON syntax for internal setup, 9

Load
sound4::cloudx1::dyn::CDynLib, 118
sound4::cloudx1::dyn::helper::CDynLoader, 130
sound4::CProcessor, 180
sound4::helper::CDynLoader, 132
log_cb_t
sound4, 80
sound4::cloudx1, 83
sound4::cloudx1::dyn, 91
LogSeverity
sound4, 80
sound4::cloudx1, 83
sound4::cloudx1::dyn, 91

main
sound4.x1.cloud-pipe-cxx.cpp, 194
sound4.x1.cloud-pipe-dyn.cpp, 198
sound4.x1.cloud-pipe.cpp, 204
sound4cl-pipe.cpp, 208
Metadata Management, 59
cloudx1_FreeMetadataInfos, 59
cloudx1_GetMetadataInfos, 59
cloudx1_SetMetadata, 60
cloudx1_SetMetadataMulti, 60
MonoToStereo
Audio convert helpers (C++), 35
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::helper, 101
sound4::cloudx1::helper, 106
MonoToStereo_Planar
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::helper, 102
sound4::cloudx1::helper, 107
MyLogger
sound4.x1.cloud-pipe-cxx.cpp, 195
sound4.x1.cloud-pipe-dyn.cpp, 199
sound4.x1.cloud-pipe.cpp, 204
MyPresetManager, 185
~MyPresetManager, 186
Exists, 186
GetAll, 186
IsReadOnly, 187
MyPresetManager, 186
Read, 187
Remove, 187
Rename, 188
SetupPath, 188
Write, 188

name
SOUND4_ProcessInfo, 192
NewClient
sound4::CInstance, 139
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 162
NewParameters
sound4::cloudx1::dyn::CDynLib, 124
none
Global setup, 55
sound4, 80

OnUpdateThreadStart
sound4::CInstance, 140
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CInstance, 163
OnUpdateThreadStop
sound4::CInstance, 140
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CInstance, 163
operator bool
sound4::cloudx1::dyn::helper::CDynLoader, 130
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 182
sound4::helper::CDynLoader, 132
sound4::helper::DynFuncHolder< T >, 184
operator T*
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 183
sound4::helper::DynFuncHolder< T >, 184
operator=
sound4::CInstance, 140
sound4::CInstance::CClient, 113
sound4::cloudx1::CInstance, 151

sound4::cloudx1::CInstance::CClient, 114
sound4::cloudx1::dyn::CInstance, 163
sound4::cloudx1::dyn::CInstance::CClient, 116
sound4::cloudx1::dyn::helper::CDynLoader, 130
sound4::helper::CDynLoader, 133
OutputChannels
sound4::cloudx1, 88
sound4::cloudx1::dyn, 98
OutputSampleSize
sound4::cloudx1, 89
sound4::cloudx1::dyn, 98

prefix
SOUND4_ProcessInfo, 192
PresetChange_Kind
sound4, 80
sound4::cloudx1, 83
sound4::cloudx1::dyn, 91
PresetManager_InformChange
sound4::CInstance, 140
sound4::cloudx1::CInstance, 151
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 163
Process creation/destruction, 61
cloudx1_ExitProcess, 61
cloudx1_FreeParameters, 61
cloudx1_FreeParameterValue, 62
cloudx1_GetParameter, 62
cloudx1_GetProcessChunkFrames, 62
cloudx1_InitProcess, 62
cloudx1_InitProcess2, 63
cloudx1_InitProcess3, 63
cloudx1_NewParameters, 64
cloudx1_SetParameter, 64
cloudx1_TerminateProcess, 64
process_name
sound4::cloudx1, 89
sound4::cloudx1::dyn, 98
process_shortname
sound4::cloudx1, 89
sound4::cloudx1::dyn, 98
ProcessAnyAudio
sound4::CInstance, 141
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CInstance, 164
ProcessAudio
sound4::CInstance, 141
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 164
ProcessAudio_Planar
sound4::CInstance, 141
sound4::cloudx1::CInstance, 152
sound4::cloudx1::dyn::CDynLib, 124
sound4::cloudx1::dyn::CInstance, 164
ProcessJson
sound4::CInstance::CClient, 113
sound4::cloudx1::CInstance::CClient, 114
sound4::cloudx1::dyn::CDynLib, 125
sound4::cloudx1::dyn::CInstance::CClient, 116
ProcessLoop
sound4.x1.cloud-pipe-cxx.cpp, 195
sound4.x1.cloud-pipe-dyn.cpp, 199
sound4.x1.cloud-pipe.cpp, 204
sound4cl-pipe.cpp, 209

Read
MyPresetManager, 187
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174
Remove
MyPresetManager, 187
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174
Rename
MyPresetManager, 188
sound4::cloudx1::CPresetLoader, 170
sound4::cloudx1::dyn::CPresetLoader, 172
sound4::CPresetLoader, 174

S16_BE
Audio managing (any audio), 38
sound4, 81
S16_LE
Audio managing (any audio), 38
sound4, 81
S16_NATIVE
Audio managing (any audio), 38
sound4, 81
S24_BE
Audio managing (any audio), 38
sound4, 81
S24_LE
Audio managing (any audio), 38
sound4, 81
S24_NATIVE
Audio managing (any audio), 38
sound4, 81
S32_BE
Audio managing (any audio), 38
sound4, 81
S32_LE
Audio managing (any audio), 38
sound4, 81
S32_NATIVE
Audio managing (any audio), 38
sound4, 81
Sample using C interface, 65
Sample using C++ interface wrapper, 66
Sample using C++ interface wrapper with dynamic library loading, 67
Sample using Generic C++ interface (multi-process compatible), 69
SAMPLE_RATE
sound4.x1.cloud-pipe.cpp, 203
SampleFormat

sound4, 81
sound4::cloudx1, 83
sound4::cloudx1::dyn, 91
SampleRate
sound4::cloudx1, 89
sound4::cloudx1::dyn, 99
SanityCheck
sound4::cloudx1, 87
sound4::cloudx1::dyn, 95
SaveState
sound4::CInstance, 142
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CDynLib, 125
sound4::cloudx1::dyn::CInstance, 165
SetBus
sound4::CInstance, 142
SetLoggerCallback
sound4::cloudx1, 87
sound4::cloudx1::dyn, 97
sound4::cloudx1::dyn::CDynLib, 125
sound4::CProcessor, 181
SetLogSeverity
sound4::cloudx1, 87
sound4::cloudx1::dyn, 97
sound4::cloudx1::dyn::CDynLib, 125
sound4::CProcessor, 181
SetMetadata
sound4::CInstance, 142
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CDynLib, 125
sound4::cloudx1::dyn::CInstance, 165
SetMetadataMulti
sound4::CInstance, 143
sound4::cloudx1::CInstance, 154
sound4::cloudx1::dyn::CDynLib, 125
sound4::cloudx1::dyn::CInstance, 165
SetParam
sound4::CInstance, 143
sound4::cloudx1::CInstance, 155
sound4::cloudx1::dyn::CInstance, 166
SetParameter
sound4::cloudx1::dyn::CDynLib, 125
SetPresetManager
sound4::CInstance, 143
sound4::cloudx1::CInstance, 155
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::CInstance, 166
SetupPath
MyPresetManager, 188
SetWebServerAppHealth
sound4::CInstance, 144
sound4::cloudx1::CInstance, 155
sound4::cloudx1::dyn::CInstance, 166
shortname
SOUND4_ProcessInfo, 192
sound4, 79
_log_cb, 82
_log_cb_c, 81
change_kind_created, 81
change_kind_deleted, 81
change_kind_modified, 81
error, 80
F32_BE, 81
F32_LE, 81
F32_NATIVE, 81
fatal, 80
info, 80
INVALID_FORMAT, 81
log_cb_t, 80
LogSeverity, 80
none, 80
PresetChange_Kind, 80
S16_BE, 81
S16_LE, 81
S16_NATIVE, 81
S24_BE, 81
S24_LE, 81
S24_NATIVE, 81
S32_BE, 81
S32_LE, 81
S32_NATIVE, 81
SampleFormat, 81
verbose, 80
verbose2, 80
verbose3, 80
verbose4, 80
verbose5, 80
warning, 80
SOUND4 x1.CLOUD Library, 1
sound4.x1.cloud-pipe-cxx.cpp, 194, 196
GetEnv, 194
main, 194
MyLogger, 195
ProcessLoop, 195
Usage, 195
sound4.x1.cloud-pipe-dyn.cpp, 198, 200
GetEnv, 198
main, 198
MyLogger, 199
ProcessLoop, 199
Usage, 199
sound4.x1.cloud-pipe.cpp, 203, 205
CHANNEL_COUNT, 203
CHUNK_SIZE, 203
main, 204
MyLogger, 204
ProcessLoop, 204
SAMPLE_RATE, 203
Usage, 204
sound4.x1.cloud.h, 213
CLOUDX1_DECL, 215
CLOUDX1_HAS_CLOUDBUS, 215
CLOUDX1_HAS_WEBSERVER, 216
sound4.x1.cloud.hpp, 220
cloudx1_custom_exists, 222
cloudx1_custom_getall, 222

cloudx1_custom_getall_free, 222
cloudx1_custom_reader, 222
cloudx1_custom_reader_free, 222
cloudx1_custom_remove, 222
cloudx1_custom_rename, 222
cloudx1_custom_writer, 223
sound4.x1.cloud_dyn.hpp, 231
cloudx1_custom_exists, 233
cloudx1_custom_getall, 233
cloudx1_custom_getall_free, 233
cloudx1_custom_reader, 233
cloudx1_custom_reader_free, 233
cloudx1_custom_remove, 234
cloudx1_custom_rename, 234
cloudx1_custom_writer, 234
UNIXLIKE, 233
sound4::CBus, 111
~CBus, 111
CBus, 111
CInstance, 112
Get, 112
sound4::CInstance, 133
~CInstance, 135
AddAudio, 135
CInstance, 134, 135
Create, 135
GetAudio, 137
GetBufferIn, 137
GetBufferOut, 137
GetChunkFrames, 138
GetEstimatedDelay, 138
GetMetadataInfos, 138
GetParam, 139
GetWebServerAppHealth, 139
IsOk, 139
NewClient, 139
OnUpdateThreadStart, 140
OnUpdateThreadStop, 140
operator=, 140
PresetManager_InformChange, 140
ProcessAnyAudio, 141
ProcessAudio, 141
ProcessAudio_Planar, 141
SaveState, 142
SetBus, 142
SetMetadata, 142
SetMetadataMulti, 143
SetParam, 143
SetPresetManager, 143
SetWebServerAppHealth, 144
StartWebServer, 144
Stop, 145
StopWebServer, 145
Terminate, 145
sound4::CInstance::CClient, 112
~CClient, 113
CClient, 113
operator=, 113
ProcessJson, 113
sound4::cloudx1, 82
_log_cb, 88
_log_cb_c, 84
ChunkFrames, 88
GetAudioInputCount, 84
GetAudioOutputCount, 84
GetBytesFromFormat, 84
GetChannelCount, 85
GetChunkSizeInFrames, 85
GetFormatFromName, 85
GetFormatName, 86
GetPossibleChunkSizeInFrames, 86
GetSampleRate, 86
GetVersion, 87
InputChannels, 88
InputSampleSize, 88
log_cb_t, 83
LogSeverity, 83
OutputChannels, 88
OutputSampleSize, 89
PresetChange_Kind, 83
process_name, 89
process_shortname, 89
SampleFormat, 83
SampleRate, 89
SanityCheck, 87
SetLoggerCallback, 87
SetLogSeverity, 87
sound4::cloudx1::CInstance, 146
~CInstance, 147
AddAudio, 147
CInstance, 147
Create, 147
GetAudio, 148
GetBufferIn, 148
GetBufferOut, 149
GetChunkFrames, 149
GetEstimatedDelay, 149
GetMetadataInfos, 150
GetParam, 150
GetWebServerAppHealth, 150
IsOk, 151
NewClient, 151
OnUpdateThreadStart, 151
OnUpdateThreadStop, 151
operator=, 151
PresetManager_InformChange, 151
ProcessAnyAudio, 152
ProcessAudio, 152
ProcessAudio_Planar, 152
SaveState, 154
SetMetadata, 154
SetMetadataMulti, 154
SetParam, 155
SetPresetManager, 155
SetWebServerAppHealth, 155
StartWebServer, 156

Stop, 156
StopWebServer, 156
Terminate, 157
sound4::cloudx1::CInstance::CClient, 114
~CClient, 114
CClient, 114
operator=, 114
ProcessJson, 114
sound4::cloudx1::CPresetLoader, 169
~CPresetLoader, 170
CPresetLoader, 170
Exists, 170
GetAll, 170
IsReadOnly, 170
Read, 170
Remove, 170
Rename, 170
Write, 171
sound4::cloudx1::dyn, 90
_log_cb, 97
_log_cb_c, 91
ChunkFrames, 97
GetAudioInputCount, 91
GetAudioOutputCount, 92
GetBytesFromFormat, 92
GetChannelCount, 92
GetChunkSizeInFrames, 93
GetFormatFromName, 93
GetFormatName, 94
GetPossibleChunkSizeInFrames, 94
GetSampleRate, 95
GetVersion, 95
InputChannels, 97
InputSampleSize, 98
log_cb_t, 91
LogSeverity, 91
OutputChannels, 98
OutputSampleSize, 98
PresetChange_Kind, 91
process_name, 98
process_shortname, 98
SampleFormat, 91
SampleRate, 99
SanityCheck, 95
SetLoggerCallback, 97
SetLogSeverity, 97
sound4::cloudx1::dyn::CDynLib, 116
AddAudio, 119
AddPadAudio, 119
AudioConvertFrom, 119
AudioConvertTo, 119
AudioMonoFromLiveStereo, 119
AudioMonoToLiveStereo, 119
CDynLib, 118
DeleteClient, 120
ExitProcess, 120
FreeJsonAnswer, 120
FreeMetadataInfos, 120
FreeParameters, 120
FreeParameterValue, 120
GetAudio, 120
GetAudioInputCount, 121
GetAudioOutputCount, 121
GetBufferIn, 121
GetBufferOut, 121
GetBytesFromFormat, 121
GetChannelCount, 121
GetChunkSizeInFrames, 121
GetEstimatedDelay, 122
GetFormatFromName, 122
GetFormatName, 122
GetMaxPacketFrame, 122
GetMetadataInfos, 122
GetOutputCount, 122
GetParameter, 122
GetPossibleChunkSizeInFrames, 123
GetProcessChunkFrames, 123
GetSampleRate, 123
GetVersion, 123
InitProcess, 123
InitProcess2, 123
InitProcess3, 123
IsOk, 118
Load, 118
MonoToStereo, 124
MonoToStereo_Planar, 124
NewClient, 124
NewParameters, 124
PresetManager_InformChange, 124
ProcessAudio, 124
ProcessAudio_Planar, 124
ProcessJson, 125
SaveState, 125
SetLoggerCallback, 125
SetLogSeverity, 125
SetMetadata, 125
SetMetadataMulti, 125
SetParameter, 125
SetPresetManager, 126
StartUpdateThread, 126
StereoToMono, 126
StereoToMono_Planar, 126
StopUpdateThread, 126
TerminateProcess, 126
WaitUpdateThreadReady, 126
Webserver, 127
Webserver_FreeString, 127
Webserver_GetAppHealth, 127
Webserver_SetAppHealth, 127
Webserver_Status, 127
Webserver_Stop, 127
Webserver_tcp, 127
Webserver_tcp2, 128
sound4::cloudx1::dyn::CInstance, 157
~CInstance, 159
AddAudio, 159

CInstance, 159
Create, 159
GetAudio, 160
GetBufferIn, 160
GetBufferOut, 160
GetChunkFrames, 161
GetEstimatedDelay, 161
GetMetadataInfos, 161
GetParam, 162
GetWebServerAppHealth, 162
IsOk, 162
NewClient, 162
OnUpdateThreadStart, 163
OnUpdateThreadStop, 163
operator=, 163
PresetManager_InformChange, 163
ProcessAnyAudio, 164
ProcessAudio, 164
ProcessAudio_Planar, 164
SaveState, 165
SetMetadata, 165
SetMetadataMulti, 165
SetParam, 166
SetPresetManager, 166
SetWebServerAppHealth, 166
StartWebServer, 167
Stop, 167
StopWebServer, 167
Terminate, 168
sound4::cloudx1::dyn::CInstance::CClient, 115
~CClient, 116
CClient, 115
operator=, 116
ProcessJson, 116
sound4::cloudx1::dyn::CPresetLoader, 171
~CPresetLoader, 172
CPresetLoader, 172
Exists, 172
GetAll, 172
IsReadOnly, 172
Read, 172
Remove, 172
Rename, 172
Write, 173
sound4::cloudx1::dyn::helper, 99
AudioConvertFrom, 100
AudioConvertTo, 100
AudioMonoFromLiveStereo, 101
AudioMonoToLiveStereo, 101
MonoToStereo, 101
MonoToStereo_Planar, 102
StereoToMono, 102
StereoToMono_Planar, 103
UTF8ToWString, 103
WStringToUTF8, 104
sound4::cloudx1::dyn::helper::CDynLoader, 128
~CDynLoader, 129
CDynLoader, 129
Close, 129
GetSymbol, 129
GetSymbol_safe, 129
GetThisLibraryPath, 129
IsOk, 130
Load, 130
operator bool, 130
operator=, 130
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 182
DynFuncHolder, 182
IsOk, 182
operator bool, 182
operator T*, 183
sound4::cloudx1::dyn::helper::SampleFormat< T >, 189
sound4::cloudx1::helper, 104
AudioConvertFrom, 105
AudioConvertTo, 105
AudioMonoFromLiveStereo, 105
AudioMonoToLiveStereo, 106
MonoToStereo, 106
MonoToStereo_Planar, 107
StereoToMono, 107
StereoToMono_Planar, 107
UTF8ToWString, 108
WStringToUTF8, 108
sound4::cloudx1::helper::SampleFormat< T >, 189
sound4::CPresetLoader, 173
~CPresetLoader, 174
CPresetLoader, 174
Exists, 174
GetAll, 174
IsReadOnly, 174
Read, 174
Remove, 174
Rename, 174
Write, 175
sound4::CProcessor, 175
CBus, 181
CInstance, 181
CProcessor, 176
GetAudioInputCount, 176
GetAudioOutputCount, 176
GetBytesFromFormat, 177
GetChannelCount, 178
GetChunkSizeInFrames, 178
GetFormatFromName, 178
GetFormatName, 179
GetPossibleChunkSizeInFrames, 179
GetPrefixSymbol, 179
GetSampleRate, 180
GetVersion, 180
IsOk, 180
Load, 180
SetLoggerCallback, 181
SetLogSeverity, 181
sound4::helper, 109

UTF8ToWString, 109
WStringToUTF8, 109
sound4::helper::CDynLoader, 130
~CDynLoader, 131
CDynLoader, 131
Close, 132
GetSymbol, 132
GetSymbol_safe, 132
GetThisLibraryPath, 132
IsOk, 132
Load, 132
operator bool, 132
operator=, 133
sound4::helper::DynFuncHolder< T >, 183
DynFuncHolder, 184
IsOk, 184
operator bool, 184
operator T*, 184
sound4::helper::SampleFormat< T >, 191
SOUND4_GetProcessInfo
Global setup, 58
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 263
Web server, 74
SOUND4_ProcessInfo, 191
name, 192
prefix, 192
shortname, 192
struct_size, 192
version, 192
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 263
Web server, 74
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 264
Web server, 74
sound4cl-pipe.cpp, 208, 210
GetEnv, 208
main, 208
ProcessLoop, 209
sound4cl.hpp, 248
UNIXLIKE, 249
sound4cl_cdef.h, 263
fs_char, 264
SOUND4_INVALID_WEBSERVER_ID, 263
SOUND4_WEBSERVER_HTTP_OK, 263
SOUND4_WEBSERVER_HTTPS_OK, 264
StartUpdateThread
sound4::cloudx1::dyn::CDynLib, 126
StartWebServer
sound4::CInstance, 144
sound4::cloudx1::CInstance, 156
sound4::cloudx1::dyn::CInstance, 167
StereoToMono
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::helper, 102
sound4::cloudx1::helper, 107
StereoToMono_Planar
Audio convert helpers (C++), 37
sound4::cloudx1::dyn::CDynLib, 126
sound4::cloudx1::dyn::helper, 103
sound4::cloudx1::helper, 107
Stop
sound4::CInstance, 145
sound4::cloudx1::CInstance, 156
sound4::cloudx1::dyn::CInstance, 167
StopUpdateThread
sound4::cloudx1::dyn::CDynLib, 126
StopWebServer
sound4::CInstance, 145
sound4::cloudx1::CInstance, 156
sound4::cloudx1::dyn::CInstance, 167
struct_size
SOUND4_ProcessInfo, 192

Terminate
sound4::CInstance, 145
sound4::cloudx1::CInstance, 157
sound4::cloudx1::dyn::CInstance, 168
TerminateProcess
sound4::cloudx1::dyn::CDynLib, 126

UNIXLIKE
sound4.x1.cloud_dyn.hpp, 233
sound4cl.hpp, 249
Update Management, 72
cloudx1_StartUpdateThread, 72
cloudx1_StopUpdateThread, 73
cloudx1_WaitUpdateThreadReady, 73
Usage
sound4.x1.cloud-pipe-cxx.cpp, 195
sound4.x1.cloud-pipe-dyn.cpp, 199
sound4.x1.cloud-pipe.cpp, 204
UTF8ToWString
sound4::cloudx1::dyn::helper, 103
sound4::cloudx1::helper, 108
sound4::helper, 109

verbose
Global setup, 55
sound4, 80
verbose2
Global setup, 55
sound4, 80
verbose3
Global setup, 55
sound4, 80
verbose4
Global setup, 55
sound4, 80
verbose5
Global setup, 55
sound4, 80
version
SOUND4_ProcessInfo, 192

WaitUpdateThreadReady
sound4::cloudx1::dyn::CDynLib, 126
warning
Global setup, 55
sound4, 80
Web server, 73
cloudx1_Webserver, 75
cloudx1_Webserver_FreeString, 75
cloudx1_Webserver_GetAppHealth, 75
cloudx1_Webserver_SetAppHealth, 76
cloudx1_Webserver_Status, 76
cloudx1_Webserver_Stop, 77
cloudx1_Webserver_tcp, 77
cloudx1_Webserver_tcp2, 77
SOUND4_INVALID_WEBSERVER_ID, 74
SOUND4_WEBSERVER_HTTP_OK, 74
SOUND4_WEBSERVER_HTTPS_OK, 74
Webserver
sound4::cloudx1::dyn::CDynLib, 127
Webserver_FreeString
sound4::cloudx1::dyn::CDynLib, 127
Webserver_GetAppHealth
sound4::cloudx1::dyn::CDynLib, 127
Webserver_SetAppHealth
sound4::cloudx1::dyn::CDynLib, 127
Webserver_Status
sound4::cloudx1::dyn::CDynLib, 127
Webserver_Stop
sound4::cloudx1::dyn::CDynLib, 127
Webserver_tcp
sound4::cloudx1::dyn::CDynLib, 127
Webserver_tcp2
sound4::cloudx1::dyn::CDynLib, 128
Write
MyPresetManager, 188
sound4::cloudx1::CPresetLoader, 171
sound4::cloudx1::dyn::CPresetLoader, 173
sound4::CPresetLoader, 175
WStringToUTF8
sound4::cloudx1::dyn::helper, 104
sound4::cloudx1::helper, 108
sound4::helper, 109