SOUND4 x1.CL Library [2.0.0]

Generated by Doxygen 1.9.8

1 SOUND4 x1.CL Library 1

1.1 General library information . . . 1

1.2 General C usage . . . 1

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

2.1 General usage . . . 3

2.2 Communicating with the instance . . . 3

2.3 Metadata . . . 4

2.4 Advanced tasks . . . 4

2.4.1 Custom storage . . . 4

2.4.2 Using a bus . . . 4

2.4.3 Convert helpers . . . 4

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

3 JSON syntax 5

3.1 Commands . . . 5

3.2 Answers . . . 6

3.3 Examples . . . 6

3.3.1 Get status . . . 6

3.3.2 Get all changes since last request . . . 6

3.3.3 Check license . . . 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
5.4 Workaround for known problems . . . 15
5.4.1 .NET (C#) . . . 15

6 Deprecated List **17**

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

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

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

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

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

12 Topic Documentation **29**
12.1 Audio convert helpers . . . 29
12.1.1 Detailed Description . . . 30
12.1.2 Enumeration Type Documentation . . . 30
12.1.2.1 cloudx1_ConverterTargetMode . . . 30

12.1.3 Function Documentation . . . 30
12.1.3.1 cloudx1_AudioConverter_From() . . . 30
12.1.3.2 cloudx1_AudioConverter_To() . . . 30
12.1.3.3 cloudx1_AudioConvertFrom() . . . 31
12.1.3.4 cloudx1_AudioConvertTo() . . . 31
12.1.3.5 cloudx1_AudioMonoFromLiveStereo() . . . 32
12.1.3.6 cloudx1_AudioMonoToLiveStereo() . . . 32
12.1.3.7 cloudx1_FreeAudioConverter() . . . 32
12.1.3.8 cloudx1_GetAudioConverter() . . . 32
12.1.3.9 cloudx1_MonoToStereo() . . . 33
12.1.3.10 cloudx1_MonoToStereo_Planar() . . . 34
12.1.3.11 cloudx1_StereoToMono() . . . 34
12.1.3.12 cloudx1_StereoToMono_Planar() . . . 34
12.2 Audio convert helpers (C++) . . . 35
12.2.1 Detailed Description . . . 35
12.2.2 Enumeration Type Documentation . . . 36
12.2.2.1 ConverterTargetMode . . . 36
12.2.3 Function Documentation . . . 36
12.2.3.1 AudioConvertFrom() . . . 36
12.2.3.2 AudioConvertTo() . . . 36
12.2.3.3 AudioMonoFromLiveStereo() . . . 37
12.2.3.4 AudioMonoToLiveStereo() . . . 37
12.2.3.5 MonoToStereo() . . . 38
12.2.3.6 MonoToStereo_Planar() . . . 38
12.2.3.7 StereoToMono() . . . 38
12.2.3.8 StereoToMono_Planar() . . . 39
12.3 Audio managing (any audio) . . . 39
12.3.1 Detailed Description . . . 40
12.3.2 Enumeration Type Documentation . . . 40
12.3.2.1 cloudx1_SampleFormat . . . 40
12.3.3 Function Documentation . . . 41
12.3.3.1 cloudx1_AddAudio() . . . 41
12.3.3.2 cloudx1_AddPadAudio() . . . 41
12.3.3.3 cloudx1_GetAudio() . . . 41
12.3.3.4 cloudx1_GetBytesFromFormat() . . . 43
12.3.3.5 cloudx1_GetFormatFromName() . . . 43
12.3.3.6 cloudx1_GetFormatName() . . . 43
12.3.3.7 cloudx1_GetMaxPacketFrame() . . . 44
12.3.3.8 cloudx1_GetOutputCount() . . . 44
12.4 Auxiliary outputs . . . 44
12.4.1 Detailed Description . . . 45
12.4.2 Function Documentation . . . 45

12.4.2.1 cloudx1_ActivateAuxiliaryAudioID() . . . 45
12.4.2.2 cloudx1_FreeAuxiliaryAudioInfos() . . . 45
12.4.2.3 cloudx1_GetAuxiliaryAudioEstimatedDelay() . . . 46
12.4.2.4 cloudx1_GetAuxiliaryAudioOutputInfos() . . . 46
12.5 Client's JSon config interfacing . . . 46
12.5.1 Detailed Description . . . 47
12.5.2 Function Documentation . . . 47
12.5.2.1 cloudx1_DeleteClient() . . . 47
12.5.2.2 cloudx1_FreeJsonAnswer() . . . 47
12.5.2.3 cloudx1_NewClient() . . . 47
12.5.2.4 cloudx1_ProcessJson() . . . 48
12.5.2.5 cloudx1_SaveState() . . . 48
12.6 Custom preset managagement . . . 49
12.6.1 Detailed Description . . . 50
12.6.2 Typedef Documentation . . . 50
12.6.2.1 cloudx1_storage_exists . . . 50
12.6.2.2 cloudx1_storage_getall . . . 50
12.6.2.3 cloudx1_storage_getall_free . . . 51
12.6.2.4 cloudx1_storage_reader . . . 51
12.6.2.5 cloudx1_storage_reader_free . . . 51
12.6.2.6 cloudx1_storage_remove . . . 52
12.6.2.7 cloudx1_storage_rename . . . 52
12.6.2.8 cloudx1_storage_writer . . . 52
12.6.2.9 fs_char . . . 53
12.6.3 Enumeration Type Documentation . . . 53
12.6.3.1 cloudx1_PresetChange_Kind . . . 53
12.6.4 Function Documentation . . . 53
12.6.4.1 cloudx1_PresetManager_InformChange() . . . 53
12.6.4.2 cloudx1_SetPresetManager() . . . 54
12.7 Direct processing . . . 54
12.7.1 Detailed Description . . . 55
12.7.2 Function Documentation . . . 55
12.7.2.1 cloudx1_GetBufferIn() . . . 55
12.7.2.2 cloudx1_GetBufferOut() . . . 55
12.7.2.3 cloudx1_GetEstimatedDelay() . . . 56
12.7.2.4 cloudx1_ProcessAudio() . . . 56
12.7.2.5 cloudx1_ProcessAudio_Planar() . . . 57
12.8 Examples . . . 57
12.9 Global setup . . . 58
12.9.1 Detailed Description . . . 58
12.9.2 Macro Definition Documentation . . . 59
12.9.2.1 CLOUDX1_AUDIO_INPUT_COUNT . . . 59

12.9.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT . . . 59
12.9.2.3 CLOUDX1_AUDIOFRAME_COUNT . . . 59
12.9.2.4 CLOUDX1_CHANNEL_COUNT . . . 59
12.9.2.5 CLOUDX1_SAMPLE_RATE . . . 60
12.9.3 Typedef Documentation . . . 60
12.9.3.1 cloudx1_loggerfn . . . 60
12.9.4 Enumeration Type Documentation . . . 60
12.9.4.1 cloudx1_LogSeverity . . . 60
12.9.5 Function Documentation . . . 61
12.9.5.1 cloudx1_GetAudioInputCount() . . . 61
12.9.5.2 cloudx1_GetAudioOutputCount() . . . 61
12.9.5.3 cloudx1_GetChannelCount() . . . 61
12.9.5.4 cloudx1_GetChunkSizeInFrames() . . . 62
12.9.5.5 cloudx1_GetPossibleChunkSizeInFrames() . . . 62
12.9.5.6 cloudx1_GetSampleRate() . . . 62
12.9.5.7 cloudx1_GetVersion() . . . 63
12.9.5.8 cloudx1_SetLoggerCallback() . . . 63
12.9.5.9 cloudx1_SetLogSeverity() . . . 63
12.9.5.10 SOUND4_GetProcessInfo() . . . 63
12.10 Instance tracing (Windows only) . . . 64
12.10.1 Detailed Description . . . 64
12.10.2 Function Documentation . . . 64
12.10.2.1 cloudx1_SetInstanceTracing() . . . 64
12.10.2.2 cloudx1_SetInstanceTracingProcessActivity() . . . 64
12.11 Metadata Management . . . 65
12.11.1 Detailed Description . . . 65
12.11.2 Function Documentation . . . 65
12.11.2.1 cloudx1_FreeMetadataInfos() . . . 65
12.11.2.2 cloudx1_GetMetadataInfos() . . . 65
12.11.2.3 cloudx1_SetMetadata() . . . 66
12.11.2.4 cloudx1_SetMetadataMulti() . . . 66
12.12 Process creation/destruction . . . 66
12.12.1 Detailed Description . . . 67
12.12.2 Function Documentation . . . 67
12.12.2.1 cloudx1_ExitProcess() . . . 67
12.12.2.2 cloudx1_FreeParameters() . . . 67
12.12.2.3 cloudx1_FreeParameterValue() . . . 68
12.12.2.4 cloudx1_GetParameter() . . . 68
12.12.2.5 cloudx1_GetProcessChunkFrames() . . . 68
12.12.2.6 cloudx1_InitProcess() . . . 68
12.12.2.7 cloudx1_InitProcess2() . . . 69
12.12.2.8 cloudx1_InitProcess3() . . . 69

12.12.2.9 cloudx1_NewParameters() . . . 70
12.12.2.10 cloudx1_SetParameter() . . . 70
12.12.2.11 cloudx1_TerminateProcess() . . . 70
12.13 Sample using C interface . . . 71
12.13.1 Sample to create the processing instance . . . 71
12.13.2 Sample processing loop . . . 72
12.13.3 Sample to stop the processing instance . . . 72
12.14 Sample using C++ interface wrapper . . . 72
12.14.1 Setup and sanity checks . . . 73
12.14.2 Sample to create the processing instance . . . 73
12.14.3 Communicating with the instance . . . 73
12.14.4 Starting a web-server . . . 73
12.14.5 Sample processing loop . . . 73
12.15 Sample using C++ interface wrapper with dynamic library loading . . . 73
12.15.1 Dynamic loading . . . 73
12.15.2 Setup and sanity checks . . . 74
12.15.3 Sample to create the processing instance . . . 74
12.15.4 Communicating with the instance . . . 74
12.15.5 Starting a web-server . . . 74
12.15.6 Sample processing loop . . . 74
12.15.7 Custom preset manager . . . 74
12.15.8 Custom preset manager loading . . . 75
12.16 Sample using Generic C++ interface (multi-process compatible) . . . 75
12.16.1 Loading the library . . . 76
12.16.2 Setup and configure . . . 76
12.16.3 Create the processing instance . . . 76
12.16.4 Processing loop . . . 76
12.16.5 Starting a web-server . . . 76
12.16.6 Communicating with the instance . . . 76
12.16.7 Custom preset manager example . . . 77
12.16.8 Custom preset manager loading . . . 77
12.17 Update Management . . . 78
12.17.1 Detailed Description . . . 78
12.17.2 Function Documentation . . . 78
12.17.2.1 cloudx1_StartUpdateThread() . . . 78
12.17.2.2 cloudx1_StopUpdateThread() . . . 78
12.17.2.3 cloudx1_WaitUpdateThreadReady() . . . 79
12.18 Web server . . . 79
12.18.1 Detailed Description . . . 80
12.18.2 Macro Definition Documentation . . . 80
12.18.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 80
12.18.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 80

12.18.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 80
12.18.3 Function Documentation . . . 80
12.18.3.1 cloudx1_Webserver() . . . 80
12.18.3.2 cloudx1_Webserver_FreeString() . . . 81
12.18.3.3 cloudx1_Webserver_GetAppHealth() . . . 81
12.18.3.4 cloudx1_Webserver_SetAppHealth() . . . 81
12.18.3.5 cloudx1_Webserver_Status() . . . 83
12.18.3.6 cloudx1_Webserver_Stop() . . . 83
12.18.3.7 cloudx1_Webserver_tcp() . . . 84
12.18.3.8 cloudx1_Webserver_tcp2() . . . 84

13 Namespace Documentation **87**

13.1 sound4 Namespace Reference . . . 87
13.1.1 Detailed Description . . . 88
13.1.2 Typedef Documentation . . . 88
13.1.2.1 log_cb_t . . . 88
13.1.3 Enumeration Type Documentation . . . 88
13.1.3.1 LogSeverity . . . 88
13.1.3.2 PresetChange_Kind . . . 89
13.1.3.3 SampleFormat . . . 89
13.1.4 Function Documentation . . . 89
13.1.4.1 _log_cb_c() . . . 89
13.1.5 Variable Documentation . . . 90
13.1.5.1 _log_cb . . . 90
13.2 sound4::cloudx1 Namespace Reference . . . 90
13.2.1 Detailed Description . . . 91
13.2.2 Typedef Documentation . . . 91
13.2.2.1 log_cb_t . . . 91
13.2.2.2 LogSeverity . . . 91
13.2.2.3 PresetChange_Kind . . . 91
13.2.2.4 SampleFormat . . . 91
13.2.3 Function Documentation . . . 92
13.2.3.1 _log_cb_c() . . . 92
13.2.3.2 GetAudioInputCount() . . . 92
13.2.3.3 GetAudioOutputCount() . . . 92
13.2.3.4 GetBytesFromFormat() . . . 92
13.2.3.5 GetChannelCount() . . . 93
13.2.3.6 GetChunkSizeInFrames() . . . 93
13.2.3.7 GetFormatFromName() . . . 93
13.2.3.8 GetFormatName() . . . 94
13.2.3.9 GetPossibleChunkSizeInFrames() . . . 94
13.2.3.10 GetSampleRate() . . . 95

13.2.3.11 GetVersion() . . . 95
13.2.3.12 SanityCheck() . . . 95
13.2.3.13 SetLoggerCallback() . . . 95
13.2.3.14 SetLogSeverity() . . . 96
13.2.4 Variable Documentation . . . 96
13.2.4.1 _log_cb . . . 96
13.2.4.2 ChunkFrames . . . 96
13.2.4.3 InputChannels . . . 96
13.2.4.4 InputSampleSize . . . 96
13.2.4.5 OutputChannels . . . 97
13.2.4.6 OutputSampleSize . . . 97
13.2.4.7 process_name . . . 97
13.2.4.8 process_shortname . . . 97
13.2.4.9 SampleRate . . . 97
13.3 sound4::cloudx1::dyn Namespace Reference . . . 98
13.3.1 Detailed Description . . . 99
13.3.2 Typedef Documentation . . . 99
13.3.2.1 log_cb_t . . . 99
13.3.2.2 LogSeverity . . . 99
13.3.2.3 PresetChange_Kind . . . 99
13.3.2.4 SampleFormat . . . 99
13.3.3 Function Documentation . . . 99
13.3.3.1 _log_cb_c() . . . 99
13.3.3.2 GetAudioInputCount() . . . 99
13.3.3.3 GetAudioOutputCount() . . . 100
13.3.3.4 GetBytesFromFormat() . . . 100
13.3.3.5 GetChannelCount() . . . 101
13.3.3.6 GetChunkSizeInFrames() . . . 101
13.3.3.7 GetFormatFromName() . . . 101
13.3.3.8 GetFormatName() . . . 102
13.3.3.9 GetPossibleChunkSizeInFrames() . . . 102
13.3.3.10 GetSampleRate() . . . 103
13.3.3.11 GetVersion() . . . 103
13.3.3.12 SanityCheck() . . . 103
13.3.3.13 SetLoggerCallback() . . . 105
13.3.3.14 SetLogSeverity() . . . 105
13.3.4 Variable Documentation . . . 105
13.3.4.1 _log_cb . . . 105
13.3.4.2 ChunkFrames . . . 105
13.3.4.3 InputChannels . . . 106
13.3.4.4 InputSampleSize . . . 106
13.3.4.5 OutputChannels . . . 106

13.3.4.6 OutputSampleSize . . . 106
13.3.4.7 process_name . . . 106
13.3.4.8 process_shortname . . . 107
13.3.4.9 SampleRate . . . 107
13.4 sound4::cloudx1::dyn::helper Namespace Reference . . . 107
13.4.1 Detailed Description . . . 108
13.4.2 Function Documentation . . . 108
13.4.2.1 AudioConvertFrom() . . . 108
13.4.2.2 AudioConvertTo() . . . 108
13.4.2.3 AudioMonoFromLiveStereo() . . . 109
13.4.2.4 AudioMonoToLiveStereo() . . . 109
13.4.2.5 MonoToStereo() . . . 109
13.4.2.6 MonoToStereo_Planar() . . . 110
13.4.2.7 StereoToMono() . . . 110
13.4.2.8 StereoToMono_Planar() . . . 111
13.4.2.9 UTF8ToWString() . . . 111
13.4.2.10 WStringToUTF8() . . . 112
13.5 sound4::cloudx1::helper Namespace Reference . . . 112
13.5.1 Detailed Description . . . 112
13.5.2 Function Documentation . . . 113
13.5.2.1 AudioConvertFrom() . . . 113
13.5.2.2 AudioConvertTo() . . . 113
13.5.2.3 AudioMonoFromLiveStereo() . . . 114
13.5.2.4 AudioMonoToLiveStereo() . . . 114
13.5.2.5 MonoToStereo() . . . 114
13.5.2.6 MonoToStereo_Planar() . . . 115
13.5.2.7 StereoToMono() . . . 115
13.5.2.8 StereoToMono_Planar() . . . 116
13.5.2.9 UTF8ToWString() . . . 116
13.5.2.10 WStringToUTF8() . . . 116
13.6 sound4::helper Namespace Reference . . . 117
13.6.1 Detailed Description . . . 117
13.6.2 Function Documentation . . . 117
13.6.2.1 UTF8ToWString() . . . 117
13.6.2.2 WStringToUTF8() . . . 117

14 Class Documentation **119**

14.1 sound4::AuxiliaryOutputInfos Struct Reference . . . 119
14.1.1 Detailed Description . . . 119
14.1.2 Member Data Documentation . . . 119
14.1.2.1 buffer . . . 119
14.1.2.2 channel_count . . . 120

14.1.2.3 description . . . 120
14.1.2.4 id . . . 120
14.1.2.5 sample_rate . . . 120
14.2 sound4::CAudioConverter Class Reference . . . 120
14.2.1 Detailed Description . . . 121
14.2.2 Constructor & Destructor Documentation . . . 121
14.2.2.1 CAudioConverter() . . . 121
14.2.2.2 ~CAudioConverter() . . . 122
14.2.3 Member Function Documentation . . . 122
14.2.3.1 Configure() . . . 122
14.2.3.2 ConvertFrom() . . . 123
14.2.3.3 ConvertTo() . . . 123
14.2.3.4 Delete() . . . 123
14.2.3.5 IsOK() . . . 124
14.3 sound4::cloudx1::CAudioConverter Class Reference . . . 124
14.3.1 Detailed Description . . . 124
14.3.2 Constructor & Destructor Documentation . . . 124
14.3.2.1 CAudioConverter() . . . 124
14.3.2.2 ~CAudioConverter() . . . 124
14.3.3 Member Function Documentation . . . 125
14.3.3.1 Configure() . . . 125
14.3.3.2 ConvertFrom() . . . 125
14.3.3.3 ConvertTo() . . . 126
14.3.3.4 Delete() . . . 126
14.3.3.5 IsOK() . . . 126
14.4 sound4::cloudx1::dyn::CAudioConverter Class Reference . . . 126
14.4.1 Detailed Description . . . 127
14.4.2 Constructor & Destructor Documentation . . . 127
14.4.2.1 CAudioConverter() . . . 127
14.4.2.2 ~CAudioConverter() . . . 127
14.4.3 Member Function Documentation . . . 127
14.4.3.1 Configure() . . . 127
14.4.3.2 ConvertFrom() . . . 129
14.4.3.3 ConvertTo() . . . 129
14.4.3.4 Delete() . . . 130
14.4.3.5 IsOK() . . . 130
14.5 sound4::CBus Class Reference . . . 130
14.5.1 Detailed Description . . . 131
14.5.2 Constructor & Destructor Documentation . . . 131
14.5.2.1 CBus() . . . 131
14.5.2.2 ~CBus() . . . 131
14.5.3 Member Function Documentation . . . 131

14.5.3.1 Get() . . . 131
14.5.4 Friends And Related Symbol Documentation . . . 131
14.5.4.1 CInstance . . . 131
14.6 sound4::CInstance::CClient Class Reference . . . 131
14.6.1 Detailed Description . . . 132
14.6.2 Constructor & Destructor Documentation . . . 132
14.6.2.1 CClient() [1/2] . . . 132
14.6.2.2 CClient() [2/2] . . . 132
14.6.2.3 ~CClient() . . . 132
14.6.3 Member Function Documentation . . . 132
14.6.3.1 operator=() . . . 132
14.6.3.2 ProcessJson() . . . 132
14.7 sound4::cloudx1::CInstance::CClient Class Reference . . . 133
14.7.1 Detailed Description . . . 133
14.7.2 Constructor & Destructor Documentation . . . 133
14.7.2.1 CClient() [1/2] . . . 133
14.7.2.2 CClient() [2/2] . . . 134
14.7.2.3 ~CClient() . . . 134
14.7.3 Member Function Documentation . . . 134
14.7.3.1 operator=() . . . 134
14.7.3.2 ProcessJson() . . . 134
14.8 sound4::cloudx1::dyn::CInstance::CClient Class Reference . . . 134
14.8.1 Detailed Description . . . 135
14.8.2 Constructor & Destructor Documentation . . . 135
14.8.2.1 CClient() [1/2] . . . 135
14.8.2.2 CClient() [2/2] . . . 135
14.8.2.3 ~CClient() . . . 135
14.8.3 Member Function Documentation . . . 135
14.8.3.1 operator=() . . . 135
14.8.3.2 ProcessJson() . . . 135
14.9 sound4::cloudx1::dyn::CDynLib Class Reference . . . 136
14.9.1 Detailed Description . . . 138
14.9.2 Constructor & Destructor Documentation . . . 138
14.9.2.1 CDynLib() . . . 138
14.9.3 Member Function Documentation . . . 138
14.9.3.1 IsOk() . . . 138
14.9.3.2 Load() . . . 138
14.9.4 Member Data Documentation . . . 138
14.9.4.1 AddAudio . . . 138
14.9.4.2 AddPadAudio . . . 139
14.9.4.3 AudioConverter_From . . . 139
14.9.4.4 AudioConverter_To . . . 139

14.9.4.5 AudioConvertFrom . . . 139
14.9.4.6 AudioConvertTo . . . 139
14.9.4.7 AudioMonoFromLiveStereo . . . 139
14.9.4.8 AudioMonoToLiveStereo . . . 139
14.9.4.9 DeleteClient . . . 140
14.9.4.10 ExitProcess . . . 140
14.9.4.11 FreeAudioConverter . . . 140
14.9.4.12 FreeJsonAnswer . . . 140
14.9.4.13 FreeMetadataInfos . . . 140
14.9.4.14 FreeParameters . . . 140
14.9.4.15 FreeParameterValue . . . 140
14.9.4.16 GetAudio . . . 141
14.9.4.17 GetAudioConverter . . . 141
14.9.4.18 GetAudioInputCount . . . 141
14.9.4.19 GetAudioOutputCount . . . 141
14.9.4.20 GetBufferIn . . . 141
14.9.4.21 GetBufferOut . . . 141
14.9.4.22 GetBytesFromFormat . . . 141
14.9.4.23 GetChannelCount . . . 142
14.9.4.24 GetChunkSizeInFrames . . . 142
14.9.4.25 GetEstimatedDelay . . . 142
14.9.4.26 GetFormatFromName . . . 142
14.9.4.27 GetFormatName . . . 142
14.9.4.28 GetMaxPacketFrame . . . 142
14.9.4.29 GetMetadataInfos . . . 142
14.9.4.30 GetOutputCount . . . 143
14.9.4.31 GetParameter . . . 143
14.9.4.32 GetPossibleChunkSizeInFrames . . . 143
14.9.4.33 GetProcessChunkFrames . . . 143
14.9.4.34 GetSampleRate . . . 143
14.9.4.35 GetVersion . . . 143
14.9.4.36 InitProcess . . . 143
14.9.4.37 InitProcess2 . . . 144
14.9.4.38 InitProcess3 . . . 144
14.9.4.39 MonoToStereo . . . 144
14.9.4.40 MonoToStereo_Planar . . . 144
14.9.4.41 NewClient . . . 144
14.9.4.42 NewParameters . . . 144
14.9.4.43 PresetManager_InformChange . . . 144
14.9.4.44 ProcessAudio . . . 145
14.9.4.45 ProcessAudio_Planar . . . 145
14.9.4.46 ProcessJson . . . 145

14.9.4.47 SaveState . . . 145
14.9.4.48 SetInstanceTracing . . . 145
14.9.4.49 SetInstanceTracingProcessActivity . . . 145
14.9.4.50 SetLoggerCallback . . . 145
14.9.4.51 SetLogSeverity . . . 146
14.9.4.52 SetMetadata . . . 146
14.9.4.53 SetMetadataMulti . . . 146
14.9.4.54 SetParameter . . . 146
14.9.4.55 SetPresetManager . . . 146
14.9.4.56 StartUpdateThread . . . 146
14.9.4.57 StereoToMono . . . 146
14.9.4.58 StereoToMono_Planar . . . 147
14.9.4.59 StopUpdateThread . . . 147
14.9.4.60 TerminateProcess . . . 147
14.9.4.61 WaitUpdateThreadReady . . . 147
14.9.4.62 Webserver . . . 147
14.9.4.63 Webserver_FreeString . . . 147
14.9.4.64 Webserver_GetAppHealth . . . 147
14.9.4.65 Webserver_SetAppHealth . . . 148
14.9.4.66 Webserver_Status . . . 148
14.9.4.67 Webserver_Stop . . . 148
14.9.4.68 Webserver_tcp . . . 148
14.9.4.69 Webserver_tcp2 . . . 148
14.10 sound4::cloudx1::dyn::helper::CDynLoader Class Reference . . . 148
14.10.1 Detailed Description . . . 149
14.10.2 Constructor & Destructor Documentation . . . 149
14.10.2.1 CDynLoader() [1/3] . . . 149
14.10.2.2 CDynLoader() [2/3] . . . 149
14.10.2.3 CDynLoader() [3/3] . . . 149
14.10.2.4 ∼CDynLoader() . . . 149
14.10.3 Member Function Documentation . . . 150
14.10.3.1 Close() . . . 150
14.10.3.2 GetSymbol() . . . 150
14.10.3.3 GetSymbol_safe() . . . 150
14.10.3.4 GetThisLibraryPath() . . . 150
14.10.3.5 IsOk() . . . 150
14.10.3.6 Load() . . . 150
14.10.3.7 operator bool() . . . 151
14.10.3.8 operator=() [1/2] . . . 151
14.10.3.9 operator=() [2/2] . . . 151
14.11 sound4::helper::CDynLoader Class Reference . . . 151
14.11.1 Detailed Description . . . 152

14.11.2 Constructor & Destructor Documentation . . . 152
14.11.2.1 CDynLoader() [1/3] . . . 152
14.11.2.2 CDynLoader() [2/3] . . . 152
14.11.2.3 CDynLoader() [3/3] . . . 152
14.11.2.4 ~CDynLoader() . . . 152
14.11.3 Member Function Documentation . . . 152
14.11.3.1 Close() . . . 152
14.11.3.2 GetSymbol() . . . 152
14.11.3.3 GetSymbol_safe() . . . 153
14.11.3.4 GetThisLibraryPath() . . . 153
14.11.3.5 IsOk() . . . 153
14.11.3.6 Load() . . . 153
14.11.3.7 operator bool() . . . 153
14.11.3.8 operator=() [1/2] . . . 153
14.11.3.9 operator=() [2/2] . . . 154
14.12 sound4::CInstance Class Reference . . . 154
14.12.1 Detailed Description . . . 155
14.12.2 Constructor & Destructor Documentation . . . 155
14.12.2.1 CInstance() [1/3] . . . 155
14.12.2.2 CInstance() [2/3] . . . 155
14.12.2.3 CInstance() [3/3] . . . 156
14.12.2.4 ~CInstance() . . . 156
14.12.3 Member Function Documentation . . . 156
14.12.3.1 AddAudio() . . . 156
14.12.3.2 Create() . . . 156
14.12.3.3 GetAudio() . . . 157
14.12.3.4 GetBufferIn() . . . 157
14.12.3.5 GetBufferOut() . . . 158
14.12.3.6 GetChunkFrames() . . . 158
14.12.3.7 GetEstimatedDelay() . . . 158
14.12.3.8 GetMetadataInfos() . . . 159
14.12.3.9 GetParam() . . . 159
14.12.3.10 GetWebServerAppHealth() . . . 159
14.12.3.11 IsOk() . . . 159
14.12.3.12 NewClient() . . . 160
14.12.3.13 OnUpdateThreadStart() . . . 160
14.12.3.14 OnUpdateThreadStop() . . . 160
14.12.3.15 operator=() . . . 160
14.12.3.16 PresetManager_InformChange() . . . 160
14.12.3.17 ProcessAnyAudio() . . . 161
14.12.3.18 ProcessAudio() . . . 161
14.12.3.19 ProcessAudio_Planar() . . . 161

14.12.3.20 SaveState() . . . 162
14.12.3.21 SetBus() . . . 162
14.12.3.22 SetInstanceTracing() . . . 162
14.12.3.23 SetInstanceTracingProcessActivity() . . . 163
14.12.3.24 SetMetadata() . . . 163
14.12.3.25 SetMetadataMulti() . . . 163
14.12.3.26 SetParam() . . . 164
14.12.3.27 SetPresetManager() . . . 164
14.12.3.28 SetWebServerAppHealth() . . . 164
14.12.3.29 StartWebServer() . . . 165
14.12.3.30 Stop() . . . 165
14.12.3.31 StopWebServer() . . . 165
14.12.3.32 Terminate() . . . 166
14.13 sound4::cloudx1::CInstance Class Reference . . . 166
14.13.1 Detailed Description . . . 167
14.13.2 Constructor & Destructor Documentation . . . 168
14.13.2.1 CInstance() [1/2] . . . 168
14.13.2.2 CInstance() [2/2] . . . 168
14.13.2.3 ~CInstance() . . . 168
14.13.3 Member Function Documentation . . . 168
14.13.3.1 ActivateAuxiliaryAudioID() . . . 168
14.13.3.2 AddAudio() . . . 168
14.13.3.3 Create() . . . 169
14.13.3.4 GetAudio() . . . 169
14.13.3.5 GetAuxiliaryAudioOutputInfos() . . . 170
14.13.3.6 GetBufferIn() . . . 170
14.13.3.7 GetBufferOut() . . . 170
14.13.3.8 GetChunkFrames() . . . 171
14.13.3.9 GetEstimatedDelay() . . . 171
14.13.3.10 GetMetadataInfos() . . . 171
14.13.3.11 GetParam() . . . 171
14.13.3.12 GetWebServerAppHealth() . . . 172
14.13.3.13 IsOk() . . . 172
14.13.3.14 NewClient() . . . 172
14.13.3.15 OnUpdateThreadStart() . . . 172
14.13.3.16 OnUpdateThreadStop() . . . 173
14.13.3.17 operator=() . . . 173
14.13.3.18 PresetManager_InformChange() . . . 173
14.13.3.19 ProcessAnyAudio() . . . 173
14.13.3.20 ProcessAudio() . . . 174
14.13.3.21 ProcessAudio_Planar() . . . 174
14.13.3.22 SaveState() . . . 174

14.13.3.23 SetInstanceTracing() . . . 175
14.13.3.24 SetInstanceTracingProcessActivity() . . . 175
14.13.3.25 SetMetadata() . . . 175
14.13.3.26 SetMetadataMulti() . . . 176
14.13.3.27 SetParam() . . . 176
14.13.3.28 SetPresetManager() . . . 176
14.13.3.29 SetWebServerAppHealth() . . . 177
14.13.3.30 StartWebServer() . . . 177
14.13.3.31 Stop() . . . 178
14.13.3.32 StopWebServer() . . . 178
14.13.3.33 Terminate() . . . 178
14.14 sound4::cloudx1::dyn::CInstance Class Reference . . . 178
14.14.1 Detailed Description . . . 180
14.14.2 Constructor & Destructor Documentation . . . 180
14.14.2.1 CInstance() [1/2] . . . 180
14.14.2.2 CInstance() [2/2] . . . 180
14.14.2.3 ∼CInstance() . . . 180
14.14.3 Member Function Documentation . . . 180
14.14.3.1 AddAudio() . . . 180
14.14.3.2 Create() . . . 181
14.14.3.3 GetAudio() . . . 181
14.14.3.4 GetBufferIn() . . . 182
14.14.3.5 GetBufferOut() . . . 182
14.14.3.6 GetChunkFrames() . . . 182
14.14.3.7 GetEstimatedDelay() . . . 183
14.14.3.8 GetMetadataInfos() . . . 183
14.14.3.9 GetParam() . . . 183
14.14.3.10 GetWebServerAppHealth() . . . 184
14.14.3.11 IsOk() . . . 184
14.14.3.12 NewClient() . . . 184
14.14.3.13 OnUpdateThreadStart() . . . 184
14.14.3.14 OnUpdateThreadStop() . . . 184
14.14.3.15 operator=() . . . 184
14.14.3.16 PresetManager_InformChange() . . . 184
14.14.3.17 ProcessAnyAudio() . . . 185
14.14.3.18 ProcessAudio() . . . 185
14.14.3.19 ProcessAudio_Planar() . . . 186
14.14.3.20 SaveState() . . . 187
14.14.3.21 SetInstanceTracing() . . . 187
14.14.3.22 SetInstanceTracingProcessActivity() . . . 187
14.14.3.23 SetMetadata() . . . 188
14.14.3.24 SetMetadataMulti() . . . 188

14.14.3.25 SetParam() . . . 188
14.14.3.26 SetPresetManager() . . . 189
14.14.3.27 SetWebServerAppHealth() . . . 189
14.14.3.28 StartWebServer() . . . 189
14.14.3.29 Stop() . . . 190
14.14.3.30 StopWebServer() . . . 190
14.14.3.31 Terminate() . . . 190
14.15 cloudx1_CAudioConverter Struct Reference . . . 191
14.15.1 Detailed Description . . . 191
14.16 cloudx1_CClientInstance Struct Reference . . . 191
14.16.1 Detailed Description . . . 191
14.17 cloudx1_CInstance Struct Reference . . . 191
14.17.1 Detailed Description . . . 191
14.18 cloudx1_CParameters Struct Reference . . . 191
14.18.1 Detailed Description . . . 192
14.19 sound4::cloudx1::CPresetLoader Class Reference . . . 192
14.19.1 Detailed Description . . . 192
14.19.2 Constructor & Destructor Documentation . . . 192
14.19.2.1 CPresetLoader() . . . 192
14.19.2.2 ∼CPresetLoader() . . . 192
14.19.3 Member Function Documentation . . . 193
14.19.3.1 Exists() . . . 193
14.19.3.2 GetAll() . . . 193
14.19.3.3 IsReadOnly() . . . 193
14.19.3.4 Read() . . . 193
14.19.3.5 Remove() . . . 193
14.19.3.6 Rename() . . . 193
14.19.3.7 Write() . . . 193
14.20 sound4::cloudx1::dyn::CPresetLoader Class Reference . . . 194
14.20.1 Detailed Description . . . 194
14.20.2 Constructor & Destructor Documentation . . . 194
14.20.2.1 CPresetLoader() . . . 194
14.20.2.2 ∼CPresetLoader() . . . 194
14.20.3 Member Function Documentation . . . 194
14.20.3.1 Exists() . . . 194
14.20.3.2 GetAll() . . . 195
14.20.3.3 IsReadOnly() . . . 195
14.20.3.4 Read() . . . 195
14.20.3.5 Remove() . . . 195
14.20.3.6 Rename() . . . 195
14.20.3.7 Write() . . . 195
14.21 sound4::CPresetLoader Class Reference . . . 196

14.21.1 Detailed Description . . . 196
14.21.2 Constructor & Destructor Documentation . . . 196
14.21.2.1 CPresetLoader() . . . 196
14.21.2.2 ∼CPresetLoader() . . . 196
14.21.3 Member Function Documentation . . . 196
14.21.3.1 Exists() . . . 196
14.21.3.2 GetAll() . . . 197
14.21.3.3 IsReadOnly() . . . 197
14.21.3.4 Read() . . . 197
14.21.3.5 Remove() . . . 197
14.21.3.6 Rename() . . . 197
14.21.3.7 Write() . . . 197
14.22 sound4::CProcessor Class Reference . . . 198
14.22.1 Detailed Description . . . 198
14.22.2 Constructor & Destructor Documentation . . . 199
14.22.2.1 CProcessor() . . . 199
14.22.3 Member Function Documentation . . . 199
14.22.3.1 GetAudioInputCount() . . . 199
14.22.3.2 GetAudioOutputCount() . . . 199
14.22.3.3 GetBytesFromFormat() . . . 199
14.22.3.4 GetChannelCount() . . . 200
14.22.3.5 GetChunkSizeInFrames() . . . 200
14.22.3.6 GetFormatFromName() . . . 200
14.22.3.7 GetFormatName() . . . 201
14.22.3.8 GetPossibleChunkSizeInFrames() . . . 201
14.22.3.9 GetPrefixSymbol() . . . 202
14.22.3.10 GetSampleRate() . . . 202
14.22.3.11 GetVersion() . . . 202
14.22.3.12 IsOk() . . . 202
14.22.3.13 Load() . . . 202
14.22.3.14 SetLoggerCallback() . . . 203
14.22.3.15 SetLogSeverity() . . . 203
14.22.4 Friends And Related Symbol Documentation . . . 203
14.22.4.1 CAudioConverter . . . 203
14.22.4.2 CBus . . . 203
14.22.4.3 CInstance . . . 204
14.23 sound4::cloudx1::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 204
14.23.1 Detailed Description . . . 204
14.23.2 Constructor & Destructor Documentation . . . 204
14.23.2.1 DynFuncHolder() [1/2] . . . 204
14.23.2.2 DynFuncHolder() [2/2] . . . 205
14.23.3 Member Function Documentation . . . 205

14.23.3.1 IsOk() . . . 205
14.23.3.2 operator bool() . . . 205
14.23.3.3 operator T∗() . . . 205
14.24 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 205
14.24.1 Detailed Description . . . 205
14.24.2 Constructor & Destructor Documentation . . . 206
14.24.2.1 DynFuncHolder() [1/2] . . . 206
14.24.2.2 DynFuncHolder() [2/2] . . . 206
14.24.3 Member Function Documentation . . . 206
14.24.3.1 IsOk() . . . 206
14.24.3.2 operator bool() . . . 206
14.24.3.3 operator T∗() . . . 206
14.25 MyPresetManager Class Reference . . . 207
14.25.1 Detailed Description . . . 207
14.25.2 Constructor & Destructor Documentation . . . 208
14.25.2.1 MyPresetManager() [1/2] . . . 208
14.25.2.2 ∼MyPresetManager() [1/2] . . . 208
14.25.2.3 MyPresetManager() [2/2] . . . 208
14.25.2.4 ∼MyPresetManager() [2/2] . . . 208
14.25.3 Member Function Documentation . . . 208
14.25.3.1 Exists() [1/2] . . . 208
14.25.3.2 Exists() [2/2] . . . 208
14.25.3.3 GetAll() [1/2] . . . 208
14.25.3.4 GetAll() [2/2] . . . 209
14.25.3.5 IsReadOnly() [1/2] . . . 209
14.25.3.6 IsReadOnly() [2/2] . . . 209
14.25.3.7 Read() [1/2] . . . 209
14.25.3.8 Read() [2/2] . . . 209
14.25.3.9 Remove() [1/2] . . . 209
14.25.3.10 Remove() [2/2] . . . 210
14.25.3.11 Rename() [1/2] . . . 210
14.25.3.12 Rename() [2/2] . . . 210
14.25.3.13 SetupPath() [1/2] . . . 210
14.25.3.14 SetupPath() [2/2] . . . 210
14.25.3.15 Write() [1/2] . . . 210
14.25.3.16 Write() [2/2] . . . 211
14.26 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference . . . 211
14.26.1 Detailed Description . . . 211
14.27 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference . . . 211
14.27.1 Detailed Description . . . 212
14.28 sound4::helper::SampleFormat< T > Struct Template Reference . . . 213
14.28.1 Detailed Description . . . 213

14.29 SOUND4_AuxiliaryOutputInfos Struct Reference . . . 213
14.29.1 Detailed Description . . . 214
14.29.2 Member Data Documentation . . . 214
14.29.2.1 buffer . . . 214
14.29.2.2 channel_count . . . 214
14.29.2.3 description . . . 214
14.29.2.4 id . . . 214
14.29.2.5 sample_rate . . . 214
14.30 SOUND4_ProcessInfo Struct Reference . . . 214
14.30.1 Detailed Description . . . 215
14.30.2 Member Data Documentation . . . 215
14.30.2.1 name . . . 215
14.30.2.2 prefix . . . 215
14.30.2.3 shortname . . . 215
14.30.2.4 struct_size . . . 215
14.30.2.5 version . . . 215

15 File Documentation **217**

15.1 doc_examples.h File Reference . . . 217
15.2 doc_examples.h . . . 217
15.3 doc_gencpp.h File Reference . . . 217
15.4 doc_gencpp.h . . . 217
15.5 doc_json.h File Reference . . . 217
15.6 doc_json.h . . . 217
15.7 doc_main.h File Reference . . . 218
15.8 doc_main.h . . . 218
15.9 doc_params.h File Reference . . . 218
15.10 doc_params.h . . . 218
15.11 sound4.x1.cloud-pipe-cxx.cpp File Reference . . . 218
15.11.1 Function Documentation . . . 218
15.11.1.1 GetEnv() . . . 218
15.11.1.2 main() . . . 219
15.11.1.3 MyLogger() . . . 219
15.11.1.4 ProcessLoop() . . . 219
15.11.1.5 Usage() . . . 219
15.12 sound4.x1.cloud-pipe-cxx.cpp . . . 220
15.13 sound4.x1.cloud-pipe-dyn.cpp File Reference . . . 222
15.13.1 Function Documentation . . . 222
15.13.1.1 GetEnv() . . . 222
15.13.1.2 main() . . . 223
15.13.1.3 MyLogger() . . . 223
15.13.1.4 ProcessLoop() . . . 223

15.13.1.5 Usage() . . . 223
15.14 sound4.x1.cloud-pipe-dyn.cpp . . . 224
15.15 sound4.x1.cloud-pipe.cpp File Reference . . . 227
15.15.1 Macro Definition Documentation . . . 227
15.15.1.1 CHANNEL_COUNT . . . 227
15.15.1.2 CHUNK_SIZE . . . 227
15.15.1.3 SAMPLE_RATE . . . 228
15.15.2 Function Documentation . . . 228
15.15.2.1 main() . . . 228
15.15.2.2 MyLogger() . . . 228
15.15.2.3 ProcessLoop() . . . 228
15.15.2.4 Usage() . . . 228
15.16 sound4.x1.cloud-pipe.cpp . . . 229
15.17 sound4cl-pipe.cpp File Reference . . . 232
15.17.1 Function Documentation . . . 232
15.17.1.1 GetEnv() . . . 232
15.17.1.2 main() . . . 233
15.17.1.3 ProcessLoop() . . . 233
15.18 sound4cl-pipe.cpp . . . 234
15.19 sound4.x1.cloud.h File Reference . . . 237
15.19.1 Macro Definition Documentation . . . 240
15.19.1.1 CLOUDX1_DECL . . . 240
15.19.1.2 CLOUDX1_HAS_CLOUDBUS . . . 240
15.19.1.3 CLOUDX1_HAS_WEBSERVER . . . 240
15.19.2 Typedef Documentation . . . 240
15.19.2.1 TraceLoggingHProvider . . . 240
15.20 sound4.x1.cloud.h . . . 241
15.21 sound4.x1.cloud.hpp File Reference . . . 245
15.21.1 Function Documentation . . . 247
15.21.1.1 cloudx1_custom_exists() . . . 247
15.21.1.2 cloudx1_custom_getall() . . . 247
15.21.1.3 cloudx1_custom_getall_free() . . . 248
15.21.1.4 cloudx1_custom_reader() . . . 248
15.21.1.5 cloudx1_custom_reader_free() . . . 248
15.21.1.6 cloudx1_custom_remove() . . . 248
15.21.1.7 cloudx1_custom_rename() . . . 248
15.21.1.8 cloudx1_custom_writer() . . . 248
15.22 sound4.x1.cloud.hpp . . . 249
15.23 sound4.x1.cloud_dyn.hpp File Reference . . . 257
15.23.1 Detailed Description . . . 259
15.23.2 Macro Definition Documentation . . . 259
15.23.2.1 UNIXLIKE . . . 259

15.23.3 Function Documentation . . . 260
15.23.3.1 cloudx1_custom_exists() . . . 260
15.23.3.2 cloudx1_custom_getall() . . . 260
15.23.3.3 cloudx1_custom_getall_free() . . . 260
15.23.3.4 cloudx1_custom_reader() . . . 260
15.23.3.5 cloudx1_custom_reader_free() . . . 260
15.23.3.6 cloudx1_custom_remove() . . . 260
15.23.3.7 cloudx1_custom_rename() . . . 261
15.23.3.8 cloudx1_custom_writer() . . . 261
15.24 sound4.x1.cloud_dyn.hpp . . . 261
15.25 sound4cl.hpp File Reference . . . 276
15.25.1 Detailed Description . . . 277
15.25.2 Macro Definition Documentation . . . 277
15.25.2.1 UNIXLIKE . . . 277
15.26 sound4cl.hpp . . . 278
15.27 sound4cl_cdef.h File Reference . . . 292
15.27.1 Detailed Description . . . 293
15.27.2 Macro Definition Documentation . . . 293
15.27.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 293
15.27.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 293
15.27.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 293
15.27.3 Typedef Documentation . . . 293
15.27.3.1 fs_char . . . 293
15.27.3.2 TraceLoggingHProvider . . . 294
15.28 sound4cl_cdef.h . . . 294

Index **297**

Chapter 1

SOUND4 x1.CL Library

1.1 General library information

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

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

Note

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

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

Have a look at our Examples.

1.2 General C usage

You may check library configuration and setup logging Global setup and Environment variables and parameters

You have to create/destroy an instance Process creation/destruction.

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

You need to start a thread calling 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.

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

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

load the preset
```
{ "save" : "`presetname`" }
```

save the preset.
If null is given, save onair preset
```
{ "delete" : "`presetname`" }
```

delete the preset.
can also be an array of presetnames
```
{ "factoryload" : "`presetname`" }
```

Create a preset from factory and load it
```
{ "rename" : { "preset" : "`presetname`", "name": "`newname`" } }
```

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

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

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

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

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

Redo last(s) `count` preset change

3.2 Answers

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

For other commands, if there is no error the answer is
```
{ "OK" : {"command" : "original command" } }
```

If the command generates warnings, the answer also lists them:
```
{ "OK" : {"command" : "import", "warnings" : ["detailed warning"] } }
```

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 SameSite mode [default|strict|lax|none]
- `TRUSTED_ORIGINS=origins` : Extra origins allowed to send requests and open websockets, as `scheme://host`[:port], separated by commas or spaces (e.g. a reverse proxy public URL, when the proxy sends neither `X-Forwarded-Host` nor `Forwarded`)
- `JSON_DIGEST=boolvalue` : if "false", do not accept HTTP Digest authentication on `/json` (default: accepted, e.g. `curl --digest -u user:password http://host/json/get/presetlist`)
- `CSRF_CHECK=boolvalue` : if "false", disable the cross-origin check of requests, websockets and `/json` GET commands (last resort, logged)

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.

5.4 Workaround for known problems

5.4.1 .NET (C#)

If you are using the library in a .NET program, you may trigger a SEGV due to .NET incompatibility with Go, referenced in `dotnet issue 110368` (present at least up to .NET 10.0.107).

This library has a mitigation to avoid the crash. You need to set the variable `SOUND4_DOTNET_MITIGATION=1` in environment before loading the library to activate it. You can check it is activated by setting `SOUND4_DOTNET_MITIGATION_VERBOSE=1` the same way, which should output information on stderr.

If you intend to attach a debugger (managed or native) to the host, set `SOUND4_DOTNET_MITIGATION_DEBUGGER=1` instead. It supersedes `SOUND4_DOTNET_MITIGATION` (no need to set both) and additionally clears `SA_ONSTACK` on `SIGTRAP` so the debugger attach does not kill the process.

Note

calling `Environment.SetEnvironmentVariable` from C# `is not sufficient on Linux`, you need to call `setenv` from libc.

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++) 35
Audio managing (any audio) 39
Auxiliary outputs 44
Client's JSon config interfacing 46
Custom preset managagement 49
Direct processing 54
Examples 57
Global setup 58
Instance tracing (Windows only) 64
Metadata Management 65
Process creation/destruction 66
Sample using C interface 71
Sample using C++ interface wrapper 72
Sample using C++ interface wrapper with dynamic library loading 73
Sample using Generic C++ interface (multi-process compatible) 75
Update Management 78
Web server 79

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

sound4 87
sound4::cloudx1 90
sound4::cloudx1::dyn 98
sound4::cloudx1::dyn::helper 107
sound4::cloudx1::helper 112
sound4::helper 117

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

sound4::AuxiliaryOutputInfos . . . 119
sound4::CAudioConverter . . . 120
sound4::cloudx1::CAudioConverter . . . 124
sound4::cloudx1::dyn::CAudioConverter . . . 126
sound4::CBus . . . 130
sound4::CInstance::CClient . . . 131
sound4::cloudx1::CInstance::CClient . . . 133
sound4::cloudx1::dyn::CInstance::CClient . . . 134
sound4::cloudx1::dyn::CDynLib . . . 136
sound4::cloudx1::dyn::helper::CDynLoader . . . 148
sound4::helper::CDynLoader . . . 151
sound4::CInstance . . . 154
sound4::cloudx1::CInstance . . . 166
sound4::cloudx1::dyn::CInstance . . . 178
cloudx1_CAudioConverter . . . 191
cloudx1_CClientInstance . . . 191
cloudx1_CInstance . . . 191
cloudx1_CParameters . . . 191
sound4::cloudx1::CPresetLoader . . . 192
sound4::cloudx1::dyn::CPresetLoader . . . 194
MyPresetManager . . . 207
sound4::CPresetLoader . . . 196
MyPresetManager . . . 207
sound4::CProcessor . . . 198
sound4::cloudx1::dyn::helper::DynFuncHolder< T > . . . 204
sound4::helper::DynFuncHolder< T > . . . 205
sound4::cloudx1::dyn::helper::SampleFormat< T > . . . 211
sound4::cloudx1::helper::SampleFormat< T > . . . 211
sound4::helper::SampleFormat< T > . . . 213
SOUND4_AuxiliaryOutputInfos . . . 213
SOUND4_ProcessInfo . . . 214

Chapter 10

Class Index

10.1 Class List

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

sound4::AuxiliaryOutputInfos . . . 119
sound4::CAudioConverter
    Audio converter helper . . . 120
sound4::cloudx1::CAudioConverter
    Audio converter helper . . . 124
sound4::cloudx1::dyn::CAudioConverter
    Audio converter helper . . . 126
sound4::CBus
    Shared Bus . . . 130
sound4::CInstance::CClient . . . 131
sound4::cloudx1::CInstance::CClient . . . 133
sound4::cloudx1::dyn::CInstance::CClient . . . 134
sound4::cloudx1::dyn::CDynLib
    Dynamic library interface . . . 136
sound4::cloudx1::dyn::helper::CDynLoader
    Helper to load dynamic library . . . 148
sound4::helper::CDynLoader
    Helper to load dynamic library . . . 151
sound4::CInstance
    Instance handling class . . . 154
sound4::cloudx1::CInstance
    Instance handling class . . . 166
sound4::cloudx1::dyn::CInstance
    Instance handling class . . . 178
cloudx1_CAudioConverter . . . 191
cloudx1_CClientInstance . . . 191
cloudx1_CInstance . . . 191
cloudx1_CParameters . . . 191
sound4::cloudx1::CPresetLoader
    Custom preset handler helper . . . 192
sound4::cloudx1::dyn::CPresetLoader
    Custom preset handler helper . . . 194
sound4::CPresetLoader
    Custom preset handler helper . . . 196
sound4::CProcessor
    Dynamic library interface . . . 198

sound4::cloudx1::dyn::helper::DynFuncHolder< T > 204
Helper to wrap a function pointer
sound4::helper::DynFuncHolder< T > 205
Helper to wrap a function pointer
MyPresetManager 207
[Custom preset manager]
sound4::cloudx1::dyn::helper::SampleFormat< T > 211
Helper for Sample format types
sound4::cloudx1::helper::SampleFormat< T > 211
Helper for Sample format types
sound4::helper::SampleFormat< T > 213
Helper for Sample format types
SOUND4_AuxiliaryOutputInfos 213
SOUND4_ProcessInfo 214

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

doc_examples.h . . . 217
doc_gencpp.h . . . 217
doc_json.h . . . 217
doc_main.h . . . 218
doc_params.h . . . 218
sound4.x1.cloud-pipe-cxx.cpp
C++ sample using C++ interface wrapper . . . 218
sound4.x1.cloud-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading . . . 222
sound4.x1.cloud-pipe.cpp
C++ sample using only C interface . . . 227
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided . . . 232
sound4.x1.cloud.h
C interface for library . . . 237
sound4.x1.cloud.hpp
C++ wrapper for library . . . 245
sound4.x1.cloud_dyn.hpp
C++ wrapper for dynamic loading library . . . 257
sound4cl.hpp
C++ wrapper for dynamic loading library . . . 276
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries . . . 292

Chapter 12

Topic Documentation

12.1 Audio convert helpers

Helpers to convert audio.

Classes

- struct cloudx1_CAudioConverter

Enumerations

- enum cloudx1_ConverterTargetMode { ConverterTargetMode_AsInput = 0 , ConverterTargetMode_Mono , ConverterTargetMode_Stereo , ConverterTargetMode_StereoToMono , ConverterTargetMode_MonoToStereo }

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)
- struct cloudx1_CAudioConverter * cloudx1_GetAudioConverter (enum cloudx1_SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, enum cloudx1_ConverterTargetMode target_mode)
- void cloudx1_FreeAudioConverter (struct cloudx1_CAudioConverter *converter)
- void cloudx1_AudioConverter_From (struct cloudx1_CAudioConverter *converter, const uint8_t *payload, float *output)
- void cloudx1_AudioConverter_To (struct cloudx1_CAudioConverter *converter, const float *input, uint8_↩ t *payload)

12.1.1 Detailed Description

Helpers to convert audio in an optimized way

12.1.2 Enumeration Type Documentation

12.1.2.1 cloudx1_ConverterTargetMode

```
enum cloudx1_ConverterTargetMode
```

Kind of process-side samples for CAudioConverter

Enumerator

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

Definition at line 1084 of file sound4.x1.cloud.h.

12.1.3 Function Documentation

12.1.3.1 cloudx1_AudioConverter_From()

```
void cloudx1_AudioConverter_From (
            struct cloudx1_CAudioConverter * converter,
            const uint8_t * payload,
            float * output )
```

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

Warning

Output buffer must be 64-byte aligned

Parameters

converter	a converter created with cloudx1_GetAudioConverter
payload	the binary raw audio buffer
output	audio output buffer (64 byte aligned)

12.1.3.2 cloudx1_AudioConverter_To()

```
void cloudx1_AudioConverter_To (
```

```
            struct cloudx1_CAudioConverter * converter,
            const float * input,
            uint8_t * payload )
```

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

Warning

Input buffer must be 64-byte aligned

Parameters

converter	a converter created with cloudx1_GetAudioConverter
input	audio input buffer (64 byte aligned)
payload	the binary raw audio buffer

12.1.3.3 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

12.1.3.4 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

12.1.3.5 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)

12.1.3.6 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

12.1.3.7 cloudx1_FreeAudioConverter()

```
void cloudx1_FreeAudioConverter (
            struct cloudx1_CAudioConverter * converter )
```

Free the audio converter created with cloudx1_GetAudioConverter.

12.1.3.8 cloudx1_GetAudioConverter()

```
struct cloudx1_CAudioConverter * cloudx1_GetAudioConverter (
            enum cloudx1_SampleFormat fmt,
            size_t nFrame,
```

```
            size_t payl_ch,
            size_t payl_ch_offset,
            enum cloudx1_ConverterTargetMode target_mode )
```

Prepare an audio converter for extract/insert and convert.

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

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

It can be used with cloudx1_AudioConverter_From and cloudx1_AudioConverter_To.

The converter will have to be freed with cloudx1_FreeAudioConverter.

Parameters

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

Returns

The converter or null if not possible

Note

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

12.1.3.9 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

12.1.3.10 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

12.1.3.11 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

12.1.3.12 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++)

Classes

- class sound4::CAudioConverter

  Audio converter helper.

Enumerations

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

Functions

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

12.2.1 Detailed Description

Helpers to convert audio in an optimized way

12.2.2 Enumeration Type Documentation

12.2.2.1 ConverterTargetMode

```
enum sound4::ConverterTargetMode
```

Kind of process-side samples for CAudioConverter

Enumerator

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

Definition at line 1576 of file sound4cl.hpp.

12.2.3 Function Documentation

12.2.3.1 AudioConvertFrom()

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

Convert the payload to floating-point array

Warning

Output buffer must be 64-byte aligned

Parameters

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

Definition at line 660 of file sound4cl.hpp.

12.2.3.2 AudioConvertTo()

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

```
            uint8_t * payload,
            size_t nSpl,
            SampleFormat fmt )  [inline]
```

Convert floating-point array to payload

Warning

Input buffer must be 64-byte aligned

Parameters

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

Definition at line 673 of file sound4cl.hpp.

12.2.3.3 AudioMonoFromLiveStereo()

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

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

Parameters

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

Definition at line 732 of file sound4cl.hpp.

12.2.3.4 AudioMonoToLiveStereo()

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

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

Parameters

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

Definition at line 741 of file sound4cl.hpp.

12.2.3.5 MonoToStereo()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 697 of file sound4cl.hpp.

12.2.3.6 MonoToStereo_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 723 of file sound4cl.hpp.

12.2.3.7 StereoToMono()

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

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 685 of file sound4cl.hpp.

12.2.3.8 StereoToMono_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

Definition at line 710 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 880 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 Auxiliary outputs

Classes

- struct SOUND4_AuxiliaryOutputInfos

Functions

- struct SOUND4_AuxiliaryOutputInfos ** cloudx1_GetAuxiliaryAudioOutputInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeAuxiliaryAudioInfos (struct SOUND4_AuxiliaryOutputInfos **infos)
- int cloudx1_ActivateAuxiliaryAudioID (struct cloudx1_CInstance *instance, const char *id)
- unsigned int cloudx1_GetAuxiliaryAudioEstimatedDelay (struct cloudx1_CInstance *instance, const char *id)

12.4.1 Detailed Description

For advanced processing with multiple simultaneous outputs.

Auxiliary outputs are filled at each call to cloudx1_ProcessAudio or cloudx1_ProcessAudio_Planar.

You can use their buffer to retreive their output.

12.4.2 Function Documentation

12.4.2.1 cloudx1_ActivateAuxiliaryAudioID()

```
int cloudx1_ActivateAuxiliaryAudioID (
            struct cloudx1_CInstance * instance,
            const char * id )
```

Activate an auxiliary output.

This is required to activate the output generation

Parameters

instance	the processing instance
id	the auxiliary output ID to activate

Returns

0 is success

12.4.2.2 cloudx1_FreeAuxiliaryAudioInfos()

```
void cloudx1_FreeAuxiliaryAudioInfos (
            struct SOUND4_AuxiliaryOutputInfos ** infos )
```

Frees the information returned by cloudx1_GetAuxiliaryAudioOutputInfos.

Parameters

infos	the infos to free

12.4.2.3 cloudx1_GetAuxiliaryAudioEstimatedDelay()

```
unsigned int cloudx1_GetAuxiliaryAudioEstimatedDelay (
            struct cloudx1_CInstance * instance,
            const char * id )
```

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

Remarks

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

But it is sufficient for file processing adjustements.

Changing the preset and parameters may change the returned value.

Parameters

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

Returns

Estimated delay in frames, or 0 if unknown id.

12.4.2.4 cloudx1_GetAuxiliaryAudioOutputInfos()

```
struct SOUND4_AuxiliaryOutputInfos ** cloudx1_GetAuxiliaryAudioOutputInfos (
            struct cloudx1_CInstance * instance )
```

Get the list of possible auxiliary outputs.

Parameters

instance	the processing instance

Returns

a list of SOUND4_AuxiliaryOutputInfos, null terminated, of auxiliary outputs. May be null. Should be freed with cloudx1_FreeAuxiliaryAudioInfos

12.5 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)

12.5.1 Detailed Description

Allow low-level JSON configuration. see also JSON syntax

12.5.2 Function Documentation

12.5.2.1 cloudx1_DeleteClient()

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

Deletes a client

Parameters

client	the client instance

12.5.2.2 cloudx1_FreeJsonAnswer()

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

Free a JSON answer returned by cloudx1_ProcessJson

Parameters

json_str	the JSON request

12.5.2.3 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

12.5.2.4 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)

12.5.2.5 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

12.6 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)

12.6.1 Detailed Description

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

By default, the library store data on disk or in memory (see `save_path` parameter in 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)

12.6.2 Typedef Documentation

12.6.2.1 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 516 of file sound4.x1.cloud.h.

12.6.2.2 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 525 of file sound4.x1.cloud.h.

12.6.2.3 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 533 of file sound4.x1.cloud.h.

12.6.2.4 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 489 of file sound4.x1.cloud.h.

12.6.2.5 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 497 of file sound4.x1.cloud.h.

12.6.2.6 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 542 of file sound4.x1.cloud.h.

12.6.2.7 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 552 of file sound4.x1.cloud.h.

12.6.2.8 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 507 of file sound4.x1.cloud.h.

12.6.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 477 of file sound4.x1.cloud.h.

12.6.3 Enumeration Type Documentation

12.6.3.1 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 585 of file sound4.x1.cloud.h.

12.6.4 Function Documentation

12.6.4.1 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

12.6.4.2 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

12.7 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)

12.7.1 Detailed Description

Optimized processing with direct in->out.

12.7.2 Function Documentation

12.7.2.1 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.

12.7.2.2 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.

12.7.2.3 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

12.7.2.4 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

12.7.2.5 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)

12.8 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

12.9 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)

12.9.1 Detailed Description

Get information on the library. Set logging mode.

12.9.2 Macro Definition Documentation

12.9.2.1 CLOUDX1_AUDIO_INPUT_COUNT

```
#define CLOUDX1_AUDIO_INPUT_COUNT 1
```

This version number of inputs for this process.

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

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

12.9.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT

```
#define CLOUDX1_AUDIO_OUTPUT_COUNT 1
```

This version number of outputs for this process.

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

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

12.9.2.3 CLOUDX1_AUDIOFRAME_COUNT

```
#define CLOUDX1_AUDIOFRAME_COUNT 64
```

This version number of frames in a process chunk.

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

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

12.9.2.4 CLOUDX1_CHANNEL_COUNT

```
#define CLOUDX1_CHANNEL_COUNT 2
```

This version number of channels.

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

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

12.9.2.5 CLOUDX1_SAMPLE_RATE

```
#define CLOUDX1_SAMPLE_RATE 48000
```

This input/output sample-rate.

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

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

12.9.3 Typedef Documentation

12.9.3.1 cloudx1_loggerfn

```
typedef void(* cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *)
```

The logging callback format

Warning

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

Note

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

Definition at line 185 of file sound4.x1.cloud.h.

12.9.4 Enumeration Type Documentation

12.9.4.1 cloudx1_LogSeverity

```
enum cloudx1_LogSeverity
```

The logging levels

Enumerator

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

Definition at line 166 of file sound4.x1.cloud.h.

12.9.5 Function Documentation

12.9.5.1 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

12.9.5.2 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

12.9.5.3 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

12.9.5.4 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

12.9.5.5 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

12.9.5.6 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

12.9.5.7 cloudx1_GetVersion()

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

Get the process version string

Returns

version string

12.9.5.8 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

The message pointer is only valid for the duration of the call: copy the string if you need to keep it

Parameters

logger	A C callback called for logging.

12.9.5.9 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

12.9.5.10 SOUND4_GetProcessInfo()

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

Get this processor description

Common to all SOUND4 processors.

Returns

the process description

12.10 Instance tracing (Windows only)

This allows instances to use windows event tracing.

Functions

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

12.10.1 Detailed Description

This allows instances to use windows event tracing

12.10.2 Function Documentation

12.10.2.1 cloudx1_SetInstanceTracing()

```
void cloudx1_SetInstanceTracing (
            struct cloudx1_CParameters * params,
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid )
```

Set the instance's tracing provider before creating instance

Parameters

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

12.10.2.2 cloudx1_SetInstanceTracingProcessActivity()

```
void cloudx1_SetInstanceTracingProcessActivity (
            struct cloudx1_CInstance * instance,
            GUID activity_guid )
```

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

Parameters

instance	the instance to configure
activity_guid	the activity guid

12.11 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)

12.11.1 Detailed Description

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

12.11.2 Function Documentation

12.11.2.1 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

12.11.2.2 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

12.11.2.3 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.

12.11.2.4 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.

12.12 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)

12.12.1 Detailed Description

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

12.12.2 Function Documentation

12.12.2.1 cloudx1_ExitProcess()

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

Quit and destroy the instance

Parameters

instance	the instance to destroy

12.12.2.2 cloudx1_FreeParameters()

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

Free the parameters created with cloudx1_NewParameters.

12.12.2.3 cloudx1_FreeParameterValue()

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

Free a parameter value returned by cloudx1_GetParameter

Parameters

value	the returned value

12.12.2.4 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

12.12.2.5 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

12.12.2.6 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

12.12.2.7 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

12.12.2.8 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

12.12.2.9 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).

12.12.2.10 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

12.12.2.11 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

12.13 Sample using C interface

C example.

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

12.13.1 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);
     }*/
```

12.13.2 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;
        }
    }
```

12.13.3 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);
```

12.14 Sample using C++ interface wrapper

C++ example (static link)

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

12.14.1 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");
```

12.14.2 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;
}
```

12.14.3 Communicating with the instance

See also JSON syntax

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

12.14.4 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;
}
```

12.14.5 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;
    }
}
```

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

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

12.15.1 Dynamic loading

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

12.15.2 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");
```

12.15.3 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;
}
```

12.15.4 Communicating with the instance

See also JSON syntax

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

12.15.5 Starting a web-server

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

12.15.6 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;
    }
}
```

12.15.7 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;
        }
};
```

12.15.8 Custom preset manager loading

```
    MyPresetManager preset_manager;
    if (!store_path.empty()) {
        if (!preset_manager.SetupPath(store_path)) {
            std::cerr « "Failed to setup the custom preset manager on path " « (const
      char*)store_path.u8string().c_str() « std::endl;
            return 1;
        }
        instance.SetPresetManager(&preset_manager);
    }
```

12.16 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

12.16.1 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;
}
```

12.16.2 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");
```

12.16.3 Create the processing instance

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

12.16.4 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;
    }
}
```

12.16.5 Starting a web-server

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

12.16.6 Communicating with the instance

See also JSON syntax

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

12.16.7 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;
        }
};
```

12.16.8 Custom preset manager loading

```
    MyPresetManager preset_manager;
```

```
    if (!store_path.empty()) {
        if (!preset_manager.SetupPath(store_path)) {
            std::cerr « "Failed to setup the custom preset manager on path " « (const
      char*)store_path.u8string().c_str() « std::endl;
            return 1;
        }
        instance.SetPresetManager(&preset_manager);
    }
```

12.17 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)

12.17.1 Detailed Description

Manage the process update and communications.

12.17.2 Function Documentation

12.17.2.1 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

12.17.2.2 cloudx1_StopUpdateThread()

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

Stop the update thread

Parameters

instance	the processing instance

12.17.2.3 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

12.18 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)

12.18.1 Detailed Description

Run a web server to configure the processor

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

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

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

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

12.18.2 Macro Definition Documentation

12.18.2.1 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 1339 of file sound4.x1.cloud.h.

12.18.2.2 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 1344 of file sound4.x1.cloud.h.

12.18.2.3 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 1348 of file sound4.x1.cloud.h.

12.18.3 Function Documentation

12.18.3.1 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.

The web server keeps a reference to the instance for its whole lifetime: stop it with cloudx1_Webserver_Stop before calling ExitProcess on this instance

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

12.18.3.2 cloudx1_Webserver_FreeString()

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

Free a a string returned by cloudx1_Webserver_GetAppHealth

Parameters

str	the string to free

12.18.3.3 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)

12.18.3.4 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

12.18.3.5 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

12.18.3.6 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

12.18.3.7 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

12.18.3.8 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

- struct AuxiliaryOutputInfos
- class CAudioConverter

  Audio converter helper.
- class CBus

  Shared Bus.
- class CInstance

  Instance handling class.
- class CPresetLoader

  Custom preset handler helper.
- class CProcessor

  Dynamic library interface.

Typedefs

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

Enumerations

- enum SampleFormat {
  INVALID_FORMAT , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE , S24_NATIVE , S32_NATIVE ,
  F32_NATIVE }
- enum LogSeverity {
  none , fatal , error , warning ,
  info , verbose , verbose2 , verbose3 ,
  verbose4 , verbose5 }
- enum PresetChange_Kind { change_kind_created , change_kind_modified , change_kind_deleted }
- enum ConverterTargetMode {
  ConverterTargetMode_AsInput = sound4cl_ConverterTargetMode_AsInput , ConverterTargetMode_Mono =
  sound4cl_ConverterTargetMode_Mono , ConverterTargetMode_Stereo = sound4cl_ConverterTargetMode↩
  _Stereo , ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_StereoToMono ,
  ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo }

Functions

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

Variables

- static log_cb_t _log_cb

13.1.1 Detailed Description

SOUND4 namespace

13.1.2 Typedef Documentation

13.1.2.1 log_cb_t

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

Log callback type

Definition at line 296 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 280 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 913 of file sound4cl.hpp.

13.1.3.3 SampleFormat

```
enum sound4::SampleFormat
```

Supported formats for audio I/O

Enumerator

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

Definition at line 60 of file sound4cl.hpp.

13.1.4 Function Documentation

13.1.4.1 _log_cb_c()

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

Definition at line 303 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 301 of file sound4cl.hpp.

13.2 sound4::cloudx1 Namespace Reference

Namespaces

- namespace dyn
- namespace helper

Classes

- class CAudioConverter

  Audio converter helper.
- 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.CL"
- 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 405 of file sound4.x1.cloud.hpp.

13.2.2.2 LogSeverity

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

Log Severity levels

Definition at line 401 of file sound4.x1.cloud.hpp.

13.2.2.3 PresetChange_Kind

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

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

13.2.2.4 SampleFormat

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

Definition at line 63 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 412 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 311 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 323 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 393 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 299 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 272 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 382 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 371 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 285 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 335 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 257 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 345 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 426 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 419 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 410 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 68 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 73 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 83 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 78 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 88 of file sound4.x1.cloud.hpp.

13.2.4.7 process_name

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

Definition at line 61 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 62 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 93 of file sound4.x1.cloud.hpp.

13.3 sound4::cloudx1::dyn Namespace Reference

Namespaces

- namespace helper

Classes

- class CAudioConverter

  Audio converter helper.
- 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.CL"
- 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 881 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 877 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 955 of file sound4.x1.cloud_dyn.hpp.

13.3.2.4 SampleFormat

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

Definition at line 515 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 888 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 780 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 793 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 869 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 767 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 734 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 857 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 845 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 748 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 806 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 718 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 818 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 902 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 895 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 886 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 520 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 525 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 535 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 530 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 540 of file sound4.x1.cloud_dyn.hpp.

13.3.4.7 process_name

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

Definition at line 513 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 514 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 545 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 611 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 626 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 695 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 706 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 654 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 684 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 640 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 669 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 569 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 557 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 158 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 172 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 236 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 246 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 198 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 226 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 185 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 212 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 117 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 105 of file sound4.x1.cloud.hpp.

13.6 sound4::helper Namespace Reference

Classes

- class CDynLoader

  Helper to load dynamic library.
- class DynFuncHolder

  Helper to wrap a function pointer.
- struct SampleFormat

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

Functions

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

13.6.1 Detailed Description

helper functions for unicode strings in Windows

13.6.2 Function Documentation

13.6.2.1 UTF8ToWString()

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

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

Attention

For Windows only

Definition at line 98 of file sound4cl.hpp.

13.6.2.2 WStringToUTF8()

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

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

Attention

For Windows only

Definition at line 86 of file sound4cl.hpp.

Chapter 14

Class Documentation

14.1 sound4::AuxiliaryOutputInfos Struct Reference

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

Public Attributes

- std::string id

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

  The description of the output.
- unsigned int sample_rate

  The sampling rate of this output.
- unsigned int channel_count

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

  The output buffer of the output. See sound4::cloudx1::CInstance::ProcessAudio for format.

14.1.1 Detailed Description

Definition at line 50 of file sound4.x1.cloud.hpp.

14.1.2 Member Data Documentation

14.1.2.1 buffer

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

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

14.1.2.2 channel_count

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

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

14.1.2.3 description

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

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

14.1.2.4 id

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

Definition at line 51 of file sound4.x1.cloud.hpp.

14.1.2.5 sample_rate

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

Definition at line 53 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.2 sound4::CAudioConverter Class Reference

Audio converter helper.

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

Public Member Functions

- CAudioConverter (CProcessor &dynlib)

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

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

14.2.1 Detailed Description

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

Definition at line 1591 of file sound4cl.hpp.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 CAudioConverter()

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

Parameters

dynlib	the processor dynamic library

Definition at line 1598 of file sound4cl.hpp.

14.2.2.2 ~CAudioConverter()

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

Definition at line 1605 of file sound4cl.hpp.

14.2.3 Member Function Documentation

14.2.3.1 Configure()

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

Configure the converter.

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

Parameters

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

Returns

true if was created

Note

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

Definition at line 1625 of file sound4cl.hpp.

14.2.3.2 ConvertFrom()

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

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

Warning

Output buffer must be 64-byte aligned

Parameters

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

Definition at line 1649 of file sound4cl.hpp.

14.2.3.3 ConvertTo()

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

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

Warning

Input buffer must be 64-byte aligned

Parameters

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

Definition at line 1660 of file sound4cl.hpp.

14.2.3.4 Delete()

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

Definition at line 1632 of file sound4cl.hpp.

14.2.3.5 IsOK()

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

Definition at line 1639 of file sound4cl.hpp.

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

- sound4cl.hpp

14.3 sound4::cloudx1::CAudioConverter Class Reference

Audio converter helper.

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

Public Member Functions

- CAudioConverter ()=default

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

  Destroy the CAudioConverter object.
- bool Configure (SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, cloudx1_ConverterTargetMode target_mode)
- void Delete ()
- bool IsOK () const
- void ConvertFrom (const uint8_t *payload, float *output)
- void ConvertTo (const float *input, uint8_t *payload)

14.3.1 Detailed Description

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

Definition at line 1108 of file sound4.x1.cloud.hpp.

14.3.2 Constructor & Destructor Documentation

14.3.2.1 CAudioConverter()

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

14.3.2.2 ~CAudioConverter()

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

Definition at line 1118 of file sound4.x1.cloud.hpp.

14.3.3 Member Function Documentation

14.3.3.1 Configure()

```
bool sound4::cloudx1::CAudioConverter::Configure (
            SampleFormat fmt,
            size_t nFrame,
            size_t payl_ch,
            size_t payl_ch_offset,
            cloudx1_ConverterTargetMode target_mode )  [inline]
```

Prepare an audio converter for extract/insert and convert.

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

Parameters

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

Returns

true if was created

Note

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

Definition at line 1138 of file sound4.x1.cloud.hpp.

14.3.3.2 ConvertFrom()

```
void sound4::cloudx1::CAudioConverter::ConvertFrom (
            const uint8_t * payload,
            float * output )  [inline]
```

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

Warning

Output buffer must be 64-byte aligned

Parameters

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

Definition at line 1161 of file sound4.x1.cloud.hpp.

14.3.3.3 ConvertTo()

```
void sound4::cloudx1::CAudioConverter::ConvertTo (
            const float * input,
            uint8_t * payload )  [inline]
```

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

Warning

Input buffer must be 64-byte aligned

Parameters

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

Definition at line 1172 of file sound4.x1.cloud.hpp.

14.3.3.4 Delete()

```
void sound4::cloudx1::CAudioConverter::Delete ( )  [inline]
```

Definition at line 1144 of file sound4.x1.cloud.hpp.

14.3.3.5 IsOK()

```
bool sound4::cloudx1::CAudioConverter::IsOK ( ) const  [inline]
```

Definition at line 1151 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::CAudioConverter Class Reference

Audio converter helper.

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

Public Member Functions

- CAudioConverter (CDynLib &dynlib)

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

  Destroy the CAudioConverter object.
- bool Configure (SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, cloudx1_ConverterTargetMode target_mode)
- void Delete ()
- bool IsOK () const
- void ConvertFrom (const uint8_t *payload, float *output)
- void ConvertTo (const float *input, uint8_t *payload)

14.4.1 Detailed Description

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

Definition at line 1595 of file sound4.x1.cloud_dyn.hpp.

14.4.2 Constructor & Destructor Documentation

14.4.2.1 CAudioConverter()

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

Parameters

dynlib	the processor dynamic library

Definition at line 1602 of file sound4.x1.cloud_dyn.hpp.

14.4.2.2 ~CAudioConverter()

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

Definition at line 1609 of file sound4.x1.cloud_dyn.hpp.

14.4.3 Member Function Documentation

14.4.3.1 Configure()

```
bool sound4::cloudx1::dyn::CAudioConverter::Configure (
            SampleFormat fmt,
            size_t nFrame,
            size_t payl_ch,
```

```
        size_t payl_ch_offset,
        cloudx1_ConverterTargetMode target_mode )  [inline]
```

Prepare an audio converter for extract/insert and convert.

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

Parameters

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

Returns

true if was created

Note

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

Definition at line 1629 of file sound4.x1.cloud_dyn.hpp.

14.4.3.2 ConvertFrom()

```
void sound4::cloudx1::dyn::CAudioConverter::ConvertFrom (
            const uint8_t * payload,
            float * output )  [inline]
```

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

Warning

Output buffer must be 64-byte aligned

Parameters

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

Definition at line 1653 of file sound4.x1.cloud_dyn.hpp.

14.4.3.3 ConvertTo()

```
void sound4::cloudx1::dyn::CAudioConverter::ConvertTo (
            const float * input,
            uint8_t * payload )  [inline]
```

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

Warning

Input buffer must be 64-byte aligned

Parameters

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

Definition at line 1664 of file sound4.x1.cloud_dyn.hpp.

14.4.3.4 Delete()

```
void sound4::cloudx1::dyn::CAudioConverter::Delete ( )  [inline]
```

Definition at line 1636 of file sound4.x1.cloud_dyn.hpp.

14.4.3.5 IsOK()

```
bool sound4::cloudx1::dyn::CAudioConverter::IsOK ( ) const  [inline]
```

Definition at line 1643 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::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

14.5.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 871 of file sound4cl.hpp.

14.5.2 Constructor & Destructor Documentation

14.5.2.1 CBus()

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

Definition at line 873 of file sound4cl.hpp.

14.5.2.2 ~CBus()

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

Definition at line 880 of file sound4cl.hpp.

14.5.3 Member Function Documentation

14.5.3.1 Get()

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

Definition at line 889 of file sound4cl.hpp.

14.5.4 Friends And Related Symbol Documentation

14.5.4.1 CInstance

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

Definition at line 888 of file sound4cl.hpp.

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

- sound4cl.hpp

14.6 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.

14.6.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1447 of file sound4cl.hpp.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 CClient() [1/2]

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

Definition at line 1449 of file sound4cl.hpp.

14.6.2.2 CClient() [2/2]

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

14.6.2.3 ~CClient()

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

Definition at line 1459 of file sound4cl.hpp.

14.6.3 Member Function Documentation

14.6.3.1 operator=()

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

14.6.3.2 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 1473 of file sound4cl.hpp.

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

- sound4cl.hpp

14.7 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.

14.7.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 986 of file sound4.x1.cloud.hpp.

14.7.2 Constructor & Destructor Documentation

14.7.2.1 CClient() [1/2]

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

Definition at line 988 of file sound4.x1.cloud.hpp.

14.7.2.2 CClient() [2/2]

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

14.7.2.3 ~CClient()

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

Definition at line 997 of file sound4.x1.cloud.hpp.

14.7.3 Member Function Documentation

14.7.3.1 operator=()

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

14.7.3.2 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 1011 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.8 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.

14.8.1 Detailed Description

Low-level JSON client.

See also JSON syntax

Definition at line 1466 of file sound4.x1.cloud_dyn.hpp.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 CClient() [1/2]

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

Definition at line 1468 of file sound4.x1.cloud_dyn.hpp.

14.8.2.2 CClient() [2/2]

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

14.8.2.3 ~CClient()

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

Definition at line 1478 of file sound4.x1.cloud_dyn.hpp.

14.8.3 Member Function Documentation

14.8.3.1 operator=()

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

14.8.3.2 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 1492 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.9 sound4::cloudx1::dyn::CDynLib Class Reference

Dynamic library interface.

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

Public Member Functions

- CDynLib ()=default
- bool Load (const std::filesystem::path &path={})

  Loads the library.
- bool IsOk () const

  Check if the library was loaded correctly.

Public Attributes

- helper::DynFuncHolder< decltype(cloudx1_GetVersion) > GetVersion
- helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames) > GetChunkSizeInFrames
- helper::DynFuncHolder< decltype(cloudx1_GetChannelCount) > GetChannelCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount) > GetAudioInputCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount) > GetAudioOutputCount
- helper::DynFuncHolder< decltype(cloudx1_GetSampleRate) > GetSampleRate
- helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback) > SetLoggerCallback
- helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity) > SetLogSeverity
- helper::DynFuncHolder< decltype(cloudx1_NewParameters) > NewParameters
- helper::DynFuncHolder< decltype(cloudx1_FreeParameters) > FreeParameters
- helper::DynFuncHolder< decltype(cloudx1_SetParameter) > SetParameter
- helper::DynFuncHolder< decltype(cloudx1_GetParameter) > GetParameter
- helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue) > FreeParameterValue
- helper::DynFuncHolder< decltype(cloudx1_InitProcess) > InitProcess
- helper::DynFuncHolder< decltype(cloudx1_InitProcess2) > InitProcess2
- helper::DynFuncHolder< decltype(cloudx1_TerminateProcess) > TerminateProcess
- helper::DynFuncHolder< decltype(cloudx1_ExitProcess) > ExitProcess
- helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread) > StartUpdateThread

- helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread) > StopUpdateThread
- helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady) > WaitUpdateThreadReady
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio) > ProcessAudio
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar) > ProcessAudio_Planar
- helper::DynFuncHolder< decltype(cloudx1_GetBufferIn) > GetBufferIn
- helper::DynFuncHolder< decltype(cloudx1_GetBufferOut) > GetBufferOut
- helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay) > GetEstimatedDelay
- helper::DynFuncHolder< decltype(cloudx1_GetFormatName) > GetFormatName
- helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName) > GetFormatFromName
- helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat) > GetBytesFromFormat
- helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame) > GetMaxPacketFrame
- helper::DynFuncHolder< decltype(cloudx1_AddAudio) > AddAudio
- helper::DynFuncHolder< decltype(cloudx1_AddPadAudio) > AddPadAudio
- helper::DynFuncHolder< decltype(cloudx1_GetOutputCount) > GetOutputCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudio) > GetAudio
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom) > AudioConvertFrom
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo) > AudioConvertTo
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono) > StereoToMono
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo) > MonoToStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo) > AudioMonoToLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_NewClient) > NewClient
- helper::DynFuncHolder< decltype(cloudx1_DeleteClient) > DeleteClient
- helper::DynFuncHolder< decltype(cloudx1_ProcessJson) > ProcessJson
- helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer) > FreeJsonAnswer
- helper::DynFuncHolder< decltype(cloudx1_SaveState) > SaveState
- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp) > Webserver_tcp
- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2) > Webserver_tcp2
- helper::DynFuncHolder< decltype(cloudx1_Webserver) > Webserver
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop) > Webserver_Stop
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Status) > Webserver_Status
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar) > StereoToMono_Planar
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar) > MonoToStereo_Planar
- helper::DynFuncHolder< decltype(cloudx1_SetMetadata) > SetMetadata
- helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos) > GetMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos) > FreeMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_SetPresetManager) > SetPresetManager
- helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange) > PresetManager_InformChange
- helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames) > GetPossibleChunkSizeInFrames
- helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames) > GetProcessChunkFrames
- helper::DynFuncHolder< decltype(cloudx1_InitProcess3) > InitProcess3
- helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti) > SetMetadataMulti
- helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth) > Webserver_SetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth) > Webserver_GetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString) > Webserver_FreeString
- helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracing) > SetInstanceTracing
- helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracingProcessActivity) > SetInstanceTracingProcessActivity
- helper::DynFuncHolder< decltype(cloudx1_GetAudioConverter) > GetAudioConverter
- helper::DynFuncHolder< decltype(cloudx1_FreeAudioConverter) > FreeAudioConverter
- helper::DynFuncHolder< decltype(cloudx1_AudioConverter_From) > AudioConverter_From
- helper::DynFuncHolder< decltype(cloudx1_AudioConverter_To) > AudioConverter_To

14.9.1 Detailed Description

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

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

14.9.2 Constructor & Destructor Documentation

14.9.2.1 CDynLib()

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

14.9.3 Member Function Documentation

14.9.3.1 IsOk()

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

Returns

true if OK.

false if not OK.

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

14.9.3.2 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 349 of file sound4.x1.cloud_dyn.hpp.

14.9.4 Member Data Documentation

14.9.4.1 AddAudio

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

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

14.9.4.2 AddPadAudio

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

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

14.9.4.3 AudioConverter_From

```
helper::DynFuncHolder< decltype(cloudx1_AudioConverter_From ) > sound4::cloudx1::dyn::CDyn↩
Lib::AudioConverter_From
```

Definition at line 339 of file sound4.x1.cloud_dyn.hpp.

14.9.4.4 AudioConverter_To

```
helper::DynFuncHolder< decltype(cloudx1_AudioConverter_To ) > sound4::cloudx1::dyn::CDynLib↩
::AudioConverter_To
```

Definition at line 340 of file sound4.x1.cloud_dyn.hpp.

14.9.4.5 AudioConvertFrom

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

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

14.9.4.6 AudioConvertTo

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

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

14.9.4.7 AudioMonoFromLiveStereo

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

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

14.9.4.8 AudioMonoToLiveStereo

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

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

14.9.4.9 DeleteClient

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

Definition at line 283 of file sound4.x1.cloud_dyn.hpp.

14.9.4.10 ExitProcess

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

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

14.9.4.11 FreeAudioConverter

```
helper::DynFuncHolder< decltype(cloudx1_FreeAudioConverter ) > sound4::cloudx1::dyn::CDynLib↩
::FreeAudioConverter
```

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

14.9.4.12 FreeJsonAnswer

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

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

14.9.4.13 FreeMetadataInfos

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

Definition at line 302 of file sound4.x1.cloud_dyn.hpp.

14.9.4.14 FreeParameters

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

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

14.9.4.15 FreeParameterValue

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

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

14.9.4.16 GetAudio

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

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

14.9.4.17 GetAudioConverter

helper::DynFuncHolder< decltype(cloudx1_GetAudioConverter) > sound4::cloudx1::dyn::CDynLib↩
::GetAudioConverter

Definition at line 337 of file sound4.x1.cloud_dyn.hpp.

14.9.4.18 GetAudioInputCount

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

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

14.9.4.19 GetAudioOutputCount

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

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

14.9.4.20 GetBufferIn

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

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

14.9.4.21 GetBufferOut

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

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

14.9.4.22 GetBytesFromFormat

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

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

14.9.4.23 GetChannelCount

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

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

14.9.4.24 GetChunkSizeInFrames

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

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

14.9.4.25 GetEstimatedDelay

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

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

14.9.4.26 GetFormatFromName

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

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

14.9.4.27 GetFormatName

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

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

14.9.4.28 GetMaxPacketFrame

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

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

14.9.4.29 GetMetadataInfos

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

Definition at line 301 of file sound4.x1.cloud_dyn.hpp.

14.9.4.30 GetOutputCount

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

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

14.9.4.31 GetParameter

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

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

14.9.4.32 GetPossibleChunkSizeInFrames

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

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

14.9.4.33 GetProcessChunkFrames

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

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

14.9.4.34 GetSampleRate

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

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

14.9.4.35 GetVersion

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

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

14.9.4.36 InitProcess

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

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

14.9.4.37 InitProcess2

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

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

14.9.4.38 InitProcess3

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

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

14.9.4.39 MonoToStereo

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

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

14.9.4.40 MonoToStereo_Planar

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

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

14.9.4.41 NewClient

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

Definition at line 282 of file sound4.x1.cloud_dyn.hpp.

14.9.4.42 NewParameters

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

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

14.9.4.43 PresetManager_InformChange

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

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

14.9.4.44 ProcessAudio

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

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

14.9.4.45 ProcessAudio_Planar

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

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

14.9.4.46 ProcessJson

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

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

14.9.4.47 SaveState

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

Definition at line 286 of file sound4.x1.cloud_dyn.hpp.

14.9.4.48 SetInstanceTracing

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

Definition at line 332 of file sound4.x1.cloud_dyn.hpp.

14.9.4.49 SetInstanceTracingProcessActivity

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

Definition at line 333 of file sound4.x1.cloud_dyn.hpp.

14.9.4.50 SetLoggerCallback

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

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

14.9.4.51 SetLogSeverity

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

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

14.9.4.52 SetMetadata

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

Definition at line 300 of file sound4.x1.cloud_dyn.hpp.

14.9.4.53 SetMetadataMulti

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

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

14.9.4.54 SetParameter

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

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

14.9.4.55 SetPresetManager

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

Definition at line 305 of file sound4.x1.cloud_dyn.hpp.

14.9.4.56 StartUpdateThread

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

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

14.9.4.57 StereoToMono

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

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

14.9.4.58 StereoToMono_Planar

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

Definition at line 296 of file sound4.x1.cloud_dyn.hpp.

14.9.4.59 StopUpdateThread

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

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

14.9.4.60 TerminateProcess

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

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

14.9.4.61 WaitUpdateThreadReady

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

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

14.9.4.62 Webserver

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

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

14.9.4.63 Webserver_FreeString

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

Definition at line 327 of file sound4.x1.cloud_dyn.hpp.

14.9.4.64 Webserver_GetAppHealth

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

Definition at line 326 of file sound4.x1.cloud_dyn.hpp.

14.9.4.65 Webserver_SetAppHealth

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

Definition at line 325 of file sound4.x1.cloud_dyn.hpp.

14.9.4.66 Webserver_Status

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

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

14.9.4.67 Webserver_Stop

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

Definition at line 291 of file sound4.x1.cloud_dyn.hpp.

14.9.4.68 Webserver_tcp

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

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

14.9.4.69 Webserver_tcp2

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

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

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

- sound4.x1.cloud_dyn.hpp

14.10 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 &&other) noexcept
- CDynLoader & operator= (CDynLoader &&other) noexcept
- 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)

14.10.1 Detailed Description

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

14.10.2 Constructor & Destructor Documentation

14.10.2.1 CDynLoader() [1/3]

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

14.10.2.2 CDynLoader() [2/3]

```
sound4::cloudx1::dyn::helper::CDynLoader::CDynLoader (
            CDynLoader && other )  [inline], [noexcept]
```

Definition at line 119 of file sound4.x1.cloud_dyn.hpp.

14.10.2.3 CDynLoader() [3/3]

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

14.10.2.4 ~CDynLoader()

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

Definition at line 223 of file sound4.x1.cloud_dyn.hpp.

14.10.3 Member Function Documentation

14.10.3.1 Close()

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

Definition at line 184 of file sound4.x1.cloud_dyn.hpp.

14.10.3.2 GetSymbol()

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

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

14.10.3.3 GetSymbol_safe()

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

Definition at line 205 of file sound4.x1.cloud_dyn.hpp.

14.10.3.4 GetThisLibraryPath()

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

Definition at line 209 of file sound4.x1.cloud_dyn.hpp.

14.10.3.5 IsOk()

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

Definition at line 130 of file sound4.x1.cloud_dyn.hpp.

14.10.3.6 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 178 of file sound4.x1.cloud_dyn.hpp.

14.10.3.7 operator bool()

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

Definition at line 127 of file sound4.x1.cloud_dyn.hpp.

14.10.3.8 operator=() [1/2]

```
CDynLoader & sound4::cloudx1::dyn::helper::CDynLoader::operator= (
            CDynLoader && other )  [inline], [noexcept]
```

Definition at line 120 of file sound4.x1.cloud_dyn.hpp.

14.10.3.9 operator=() [2/2]

```
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

14.11 sound4::helper::CDynLoader Class Reference

Helper to load dynamic library.

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

Public Member Functions

- CDynLoader ()=default
- CDynLoader (CDynLoader &&other) noexcept
- CDynLoader & operator= (CDynLoader &&other) noexcept
- 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)

14.11.1 Detailed Description

Definition at line 155 of file sound4cl.hpp.

14.11.2 Constructor & Destructor Documentation

14.11.2.1 CDynLoader() [1/3]

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

14.11.2.2 CDynLoader() [2/3]

```
sound4::helper::CDynLoader::CDynLoader (
            CDynLoader && other )  [inline], [noexcept]
```

Definition at line 166 of file sound4cl.hpp.

14.11.2.3 CDynLoader() [3/3]

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

14.11.2.4 ~CDynLoader()

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

Definition at line 270 of file sound4cl.hpp.

14.11.3 Member Function Documentation

14.11.3.1 Close()

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

Definition at line 231 of file sound4cl.hpp.

14.11.3.2 GetSymbol()

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

Definition at line 244 of file sound4cl.hpp.

14.11.3.3 GetSymbol_safe()

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

Definition at line 252 of file sound4cl.hpp.

14.11.3.4 GetThisLibraryPath()

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

Definition at line 256 of file sound4cl.hpp.

14.11.3.5 IsOk()

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

Definition at line 177 of file sound4cl.hpp.

14.11.3.6 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 225 of file sound4cl.hpp.

14.11.3.7 operator bool()

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

Definition at line 174 of file sound4cl.hpp.

14.11.3.8 operator=() [1/2]

```
CDynLoader & sound4::helper::CDynLoader::operator= (
            CDynLoader && other )  [inline], [noexcept]
```

Definition at line 167 of file sound4cl.hpp.

14.11.3.9 operator=() [2/2]

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

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

- sound4cl.hpp

14.12 sound4::CInstance Class Reference

Instance handling class.

`#include <sound4cl.hpp>`

Classes

- class CClient

Public Member Functions

- CInstance (CProcessor &dynlib)
- CInstance (CProcessor &dynlib, void *_instance)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- virtual ~CInstance ()
- bool IsOk () const
- void SetParam (const std::string &name, const std::string &value)
- std::string GetParam (const std::string &name)
- void SetBus (const CBus &bus)
- void SetPresetManager (CPresetLoader *preset_manager)
- bool Create (const std::string &LoginKey, const std::string &RadioName, const std::string &Access_Key_ID, const std::string &Access_Key_Secret, const std::filesystem::path &save_path, int json_port=0, unsigned int frames_per_chunk=0)
- unsigned int GetChunkFrames ()
- void PresetManager_InformChange (const std::filesystem::path &relative_path, PresetChange_Kind change_kind)
- void SetMetadata (const std::string &key, const char *value)
- void SetMetadataMulti (const std::unordered_map< std::string, const char * > &list)
- std::vector< std::tuple< std::string, std::string > > GetMetadataInfos ()
- virtual void OnUpdateThreadStart ()
- virtual void OnUpdateThreadStop ()
- bool StartWebServer (int http_port, int https_port=0)
- void StopWebServer (int timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- int Terminate ()
- void Stop ()
- unsigned int GetEstimatedDelay ()
- float * GetBufferIn ()
- float * GetBufferOut ()
- void ProcessAudio ()
- void ProcessAudio_Planar (float const *const *input, float *const *output)
- template<typename T >
  std::vector< T > ProcessAnyAudio (const std::vector< T > input)
- std::shared_ptr< CClient > NewClient ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

Protected Member Functions

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

14.12.1 Detailed Description

Holds an instance of processing

Definition at line 923 of file sound4cl.hpp.

14.12.2 Constructor & Destructor Documentation

14.12.2.1 CInstance() [1/3]

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

Creates the instance, with default parameters

Parameters

dynlib	the processor dynamic library

Definition at line 930 of file sound4cl.hpp.

14.12.2.2 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 945 of file sound4cl.hpp.

14.12.2.3 CInstance() [3/3]

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

14.12.2.4 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 957 of file sound4cl.hpp.

14.12.3 Member Function Documentation

14.12.3.1 AddAudio()

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

Definition at line 1547 of file sound4cl.hpp.

14.12.3.2 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 1061 of file sound4cl.hpp.

14.12.3.3 GetAudio()

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

Definition at line 1553 of file sound4cl.hpp.

14.12.3.4 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 1364 of file sound4cl.hpp.

14.12.3.5 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 1381 of file sound4cl.hpp.

14.12.3.6 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 1111 of file sound4cl.hpp.

14.12.3.7 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 1346 of file sound4cl.hpp.

14.12.3.8 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 1199 of file sound4cl.hpp.

14.12.3.9 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 986 of file sound4cl.hpp.

14.12.3.10 GetWebServerAppHealth()

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

Get the web server /apphealth status info

Definition at line 1292 of file sound4cl.hpp.

14.12.3.11 IsOk()

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

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

Definition at line 965 of file sound4cl.hpp.

14.12.3.12 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 1497 of file sound4cl.hpp.

14.12.3.13 OnUpdateThreadStart()

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

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

Definition at line 1223 of file sound4cl.hpp.

14.12.3.14 OnUpdateThreadStop()

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

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

Definition at line 1227 of file sound4cl.hpp.

14.12.3.15 operator=()

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

14.12.3.16 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 1125 of file sound4cl.hpp.

14.12.3.17 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 1423 of file sound4cl.hpp.

14.12.3.18 ProcessAudio()

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

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

Definition at line 1389 of file sound4cl.hpp.

14.12.3.19 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 1405 of file sound4cl.hpp.

14.12.3.20 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 1513 of file sound4cl.hpp.

14.12.3.21 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 1003 of file sound4cl.hpp.

14.12.3.22 SetInstanceTracing()

```
void sound4::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

Parameters

activity_guid	the activity guid, used for creation, updates and destruction
tracing_provider	the provider as created with TRACELOGGING_DEFINE_PROVIDER

Definition at line 1527 of file sound4cl.hpp.

14.12.3.23 SetInstanceTracingProcessActivity()

```
void sound4::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

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

Parameters

activity_guid	the activity guid

Definition at line 1538 of file sound4cl.hpp.

14.12.3.24 SetMetadata()

```
void sound4::CInstance::SetMetadata (
            const std::string & key,
            const char * value )  [inline]
```

Adds the specific metadata to the process.

It can come from any thread, and will be used as soon as possible. Setting a null value erase the metadata, which is different from setting an empty string.

Parameters

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

Definition at line 1143 of file sound4cl.hpp.

14.12.3.25 SetMetadataMulti()

```
void sound4::CInstance::SetMetadataMulti (
            const std::unordered_map< std::string, const char * > & list )  [inline]
```

Adds multiple metadata to the process.

This works like SetMetadata but with multiple metadata at once.

See also

CInstance::SetMetadata

Parameters

list	a map of key/value pair.

Definition at line 1163 of file sound4cl.hpp.

14.12.3.26 SetParam()

```
void sound4::CInstance::SetParam (
            const std::string & name,
            const std::string & value )  [inline]
```

Set a parameter value

Must be done before Create See also Environment variables and parameters

Parameters

name	the parameter name
value	the parameter value

Definition at line 975 of file sound4cl.hpp.

14.12.3.27 SetPresetManager()

```
void sound4::CInstance::SetPresetManager (
            CPresetLoader * preset_manager )  [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in Create

To dynamically update when storage change, see PresetManager_InformChange

Must be done before Create

Parameters

preset_manager	The custom preset manager

Definition at line 1020 of file sound4cl.hpp.

14.12.3.28 SetWebServerAppHealth()

```
void sound4::CInstance::SetWebServerAppHealth (
            int httpcode,
```

```
            const std::string & contenttype,
            const std::string & content )  [inline]
```

Set the web server /apphealth status info

Definition at line 1283 of file sound4cl.hpp.

14.12.3.29 StartWebServer()

```
bool sound4::CInstance::StartWebServer (
            int http_port,
            int https_port = 0 )  [inline]
```

Starts a Web server on requested ports

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

Warning

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

Parameters

http_port	The port to listen for HTTP connections (0 to disable)
https_port	The port to listen for HTTPS connections (0 to disable)

Returns

true if ports were opened successfully.

Definition at line 1243 of file sound4cl.hpp.

14.12.3.30 Stop()

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

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

Note

Stop will not stop a wrapped C instance

Definition at line 1323 of file sound4cl.hpp.

14.12.3.31 StopWebServer()

```
void sound4::CInstance::StopWebServer (
            int timeout_ms = 1000 )  [inline]
```

Stop a web server started with StartWebServer

Warning

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

Parameters

timeout_ms	The timeout to wait gracefull shutdown

Definition at line 1271 of file sound4cl.hpp.

14.12.3.32 Terminate()

```
int sound4::CInstance::Terminate ( )  [inline]
```

Terminate the processing.

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

Returns

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

Definition at line 1312 of file sound4cl.hpp.

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

- sound4cl.hpp

14.13 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 ()
- std::vector< AuxiliaryOutputInfos > GetAuxiliaryAudioOutputInfos ()
- void ActivateAuxiliaryAudioID (const std::string &id)
- virtual void OnUpdateThreadStart ()
- virtual void OnUpdateThreadStop ()
- bool StartWebServer (int http_port, int https_port=0)
- void StopWebServer (int timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- int Terminate ()
- void Stop ()
- unsigned int GetEstimatedDelay ()
- std::array< float, InputSampleSize > & GetBufferIn ()
- std::array< float, OutputSampleSize > & GetBufferOut ()
- void ProcessAudio ()
- void ProcessAudio_Planar (float const *const *input, float *const *output)
- template<typename T >
  std::vector< T > ProcessAnyAudio (const std::vector< T > input)
- std::shared_ptr< CClient > NewClient ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

Protected Member Functions

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

14.13.1 Detailed Description

Holds an instance of processing

Definition at line 478 of file sound4.x1.cloud.hpp.

14.13.2 Constructor & Destructor Documentation

14.13.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 483 of file sound4.x1.cloud.hpp.

14.13.2.2 CInstance() [2/2]

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

14.13.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 494 of file sound4.x1.cloud.hpp.

14.13.3 Member Function Documentation

14.13.3.1 ActivateAuxiliaryAudioID()

```
void sound4::cloudx1::CInstance::ActivateAuxiliaryAudioID (
            const std::string & id )  [inline]
```

Definition at line 759 of file sound4.x1.cloud.hpp.

14.13.3.2 AddAudio()

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

Definition at line 1080 of file sound4.x1.cloud.hpp.

14.13.3.3 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 597 of file sound4.x1.cloud.hpp.

14.13.3.4 GetAudio()

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

Definition at line 1086 of file sound4.x1.cloud.hpp.

14.13.3.5 GetAuxiliaryAudioOutputInfos()

```
std::vector< AuxiliaryOutputInfos > sound4::cloudx1::CInstance::GetAuxiliaryAudioOutputInfos (
) [inline]
```

Get information on auxiliary audio outputs.

Returns

a list of AuxiliaryOutputInfos

Definition at line 740 of file sound4.x1.cloud.hpp.

14.13.3.6 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 901 of file sound4.x1.cloud.hpp.

14.13.3.7 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 919 of file sound4.x1.cloud.hpp.

14.13.3.8 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 638 of file sound4.x1.cloud.hpp.

14.13.3.9 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 883 of file sound4.x1.cloud.hpp.

14.13.3.10 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 716 of file sound4.x1.cloud.hpp.

14.13.3.11 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 523 of file sound4.x1.cloud.hpp.

14.13.3.12 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 830 of file sound4.x1.cloud.hpp.

14.13.3.13 IsOk()

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

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

Definition at line 502 of file sound4.x1.cloud.hpp.

14.13.3.14 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 1034 of file sound4.x1.cloud.hpp.

14.13.3.15 OnUpdateThreadStart()

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

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

Definition at line 766 of file sound4.x1.cloud.hpp.

14.13.3.16 OnUpdateThreadStop()

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

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

Definition at line 770 of file sound4.x1.cloud.hpp.

14.13.3.17 operator=()

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

14.13.3.18 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 652 of file sound4.x1.cloud.hpp.

14.13.3.19 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 962 of file sound4.x1.cloud.hpp.

14.13.3.20 ProcessAudio()

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

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

Definition at line 928 of file sound4.x1.cloud.hpp.

14.13.3.21 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 944 of file sound4.x1.cloud.hpp.

14.13.3.22 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 1050 of file sound4.x1.cloud.hpp.

14.13.3.23 SetInstanceTracing()

```
void sound4::cloudx1::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

Parameters

activity_guid	the activity guid, used for creation, updates and destruction
tracing_provider	the provider as created with TRACELOGGING_DEFINE_PROVIDER

Definition at line 1064 of file sound4.x1.cloud.hpp.

14.13.3.24 SetInstanceTracingProcessActivity()

```
void sound4::cloudx1::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

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

Parameters

activity_guid	the activity guid

Definition at line 1073 of file sound4.x1.cloud.hpp.

14.13.3.25 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 668 of file sound4.x1.cloud.hpp.

14.13.3.26 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 686 of file sound4.x1.cloud.hpp.

14.13.3.27 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 512 of file sound4.x1.cloud.hpp.

14.13.3.28 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 557 of file sound4.x1.cloud.hpp.

14.13.3.29 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 822 of file sound4.x1.cloud.hpp.

14.13.3.30 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 786 of file sound4.x1.cloud.hpp.

14.13.3.31 Stop()

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

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

Definition at line 860 of file sound4.x1.cloud.hpp.

14.13.3.32 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 811 of file sound4.x1.cloud.hpp.

14.13.3.33 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 851 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.14 sound4::cloudx1::dyn::CInstance Class Reference

Instance handling class.

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

Classes

- class CClient

Public Member Functions

- CInstance (CDynLib &dynlib)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- virtual ~CInstance ()
- bool IsOk () const
- void SetParam (const std::string &name, const std::string &value)
- std::string GetParam (const std::string &name)
- void SetPresetManager (CPresetLoader *preset_manager)
- bool Create (const std::string &LoginKey, const std::string &RadioName, const std::string &Access_Key_ID, const std::string &Access_Key_Secret, const std::filesystem::path &save_path, int json_port=0, unsigned int frames_per_chunk=64)
- unsigned int GetChunkFrames ()
- void PresetManager_InformChange (const std::filesystem::path &relative_path, PresetChange_Kind change_kind)
- void SetMetadata (const std::string &key, const char *value)
- void SetMetadataMulti (const std::unordered_map< std::string, const char * > &list)
- std::vector< std::tuple< std::string, std::string > > GetMetadataInfos ()
- virtual void OnUpdateThreadStart ()
- virtual void OnUpdateThreadStop ()
- bool StartWebServer (int http_port, int https_port=0)
- void StopWebServer (int timeout_ms=1000)
- void SetWebServerAppHealth (int httpcode, const std::string &contenttype, const std::string &content)
- void GetWebServerAppHealth (int &httpcode, std::string &contenttype, std::string &content)
- int Terminate ()
- void Stop ()
- unsigned int GetEstimatedDelay ()
- std::array< float, InputSampleSize > & GetBufferIn ()
- std::array< float, OutputSampleSize > & GetBufferOut ()
- void ProcessAudio ()
- void ProcessAudio_Planar (float const *const *input, float *const *output)
- template<typename T >
  std::vector< T > ProcessAnyAudio (const std::vector< T > input)
- std::shared_ptr< CClient > NewClient ()
- bool SaveState ()
- void SetInstanceTracing (TraceLoggingHProvider tracing_provider, GUID activity_guid={})
- void SetInstanceTracingProcessActivity (GUID activity_guid)

Protected Member Functions

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

14.14.1 Detailed Description

Holds an instance of processing

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

14.14.2 Constructor & Destructor Documentation

14.14.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 967 of file sound4.x1.cloud_dyn.hpp.

14.14.2.2 CInstance() [2/2]

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

14.14.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 979 of file sound4.x1.cloud_dyn.hpp.

14.14.3 Member Function Documentation

14.14.3.1 AddAudio()

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

Definition at line 1566 of file sound4.x1.cloud_dyn.hpp.

14.14.3.2 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 1084 of file sound4.x1.cloud_dyn.hpp.

14.14.3.3 GetAudio()

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

Definition at line 1572 of file sound4.x1.cloud_dyn.hpp.

14.14.3.4 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 1381 of file sound4.x1.cloud_dyn.hpp.

14.14.3.5 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 1399 of file sound4.x1.cloud_dyn.hpp.

14.14.3.6 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 1131 of file sound4.x1.cloud_dyn.hpp.

14.14.3.7 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 1363 of file sound4.x1.cloud_dyn.hpp.

14.14.3.8 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 1220 of file sound4.x1.cloud_dyn.hpp.

14.14.3.9 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 1008 of file sound4.x1.cloud_dyn.hpp.

14.14.3.10 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 1309 of file sound4.x1.cloud_dyn.hpp.

14.14.3.11 IsOk()

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

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

Definition at line 987 of file sound4.x1.cloud_dyn.hpp.

14.14.3.12 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 1516 of file sound4.x1.cloud_dyn.hpp.

14.14.3.13 OnUpdateThreadStart()

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

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

Definition at line 1244 of file sound4.x1.cloud_dyn.hpp.

14.14.3.14 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 1248 of file sound4.x1.cloud_dyn.hpp.

14.14.3.15 operator=()

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

14.14.3.16 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 1145 of file sound4.x1.cloud_dyn.hpp.

14.14.3.17 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 1442 of file sound4.x1.cloud_dyn.hpp.

14.14.3.18 ProcessAudio()

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

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

Definition at line 1408 of file sound4.x1.cloud_dyn.hpp.

14.14.3.19 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 1424 of file sound4.x1.cloud_dyn.hpp.

14.14.3.20 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 1532 of file sound4.x1.cloud_dyn.hpp.

14.14.3.21 SetInstanceTracing()

```
void sound4::cloudx1::dyn::CInstance::SetInstanceTracing (
            TraceLoggingHProvider tracing_provider,
            GUID activity_guid = {} )  [inline]
```

Set the instance's tracing provider before creating instance

Parameters

activity_guid	the activity guid, used for creation, updates and destruction
tracing_provider	the provider as created with TRACELOGGING_DEFINE_PROVIDER

Definition at line 1546 of file sound4.x1.cloud_dyn.hpp.

14.14.3.22 SetInstanceTracingProcessActivity()

```
void sound4::cloudx1::dyn::CInstance::SetInstanceTracingProcessActivity (
            GUID activity_guid )  [inline]
```

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

Parameters

activity_guid	the activity guid

Definition at line 1557 of file sound4.x1.cloud_dyn.hpp.

14.14.3.23 SetMetadata()

```
void sound4::cloudx1::dyn::CInstance::SetMetadata (
            const std::string & key,
            const char * value )  [inline]
```

Adds the specific metadata to the process.

It can come from any thread, and will be used as soon as possible. Setting a null value erase the metadata, which is different from setting an empty string.

Parameters

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

Definition at line 1163 of file sound4.x1.cloud_dyn.hpp.

14.14.3.24 SetMetadataMulti()

```
void sound4::cloudx1::dyn::CInstance::SetMetadataMulti (
            const std::unordered_map< std::string, const char * > & list )  [inline]
```

Adds multiple metadata to the process.

This works like SetMetadata but with multiple metadata at once.

See also

CInstance::SetMetadata

Parameters

list	a map of key/value pair.

Definition at line 1184 of file sound4.x1.cloud_dyn.hpp.

14.14.3.25 SetParam()

```
void sound4::cloudx1::dyn::CInstance::SetParam (
            const std::string & name,
            const std::string & value )  [inline]
```

Set a parameter value

Must be done before Create See also Environment variables and parameters

Parameters

name	the parameter name
value	the parameter value

Definition at line 997 of file sound4.x1.cloud_dyn.hpp.

14.14.3.26 SetPresetManager()

```
void sound4::cloudx1::dyn::CInstance::SetPresetManager (
            CPresetLoader * preset_manager )  [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in Create

To dynamically update when storage change, see PresetManager_InformChange

Must be done before Create

Parameters

preset_manager	The custom preset manager

Definition at line 1043 of file sound4.x1.cloud_dyn.hpp.

14.14.3.27 SetWebServerAppHealth()

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

Set the web server /apphealth status info

Definition at line 1300 of file sound4.x1.cloud_dyn.hpp.

14.14.3.28 StartWebServer()

```
bool sound4::cloudx1::dyn::CInstance::StartWebServer (
            int http_port,
            int https_port = 0 )  [inline]
```

Starts a Web server on requested ports

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

Warning

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

Parameters

http_port	The port to listen for HTTP connections (0 to disable)
https_port	The port to listen for HTTPS connections (0 to disable)

Returns

true if ports were opened successfully.

Definition at line 1264 of file sound4.x1.cloud_dyn.hpp.

14.14.3.29 Stop()

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

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

Definition at line 1340 of file sound4.x1.cloud_dyn.hpp.

14.14.3.30 StopWebServer()

```
void sound4::cloudx1::dyn::CInstance::StopWebServer (
            int timeout_ms = 1000 )  [inline]
```

Stop a web server started with StartWebServer

Warning

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

Parameters

timeout_ms	The timeout to wait gracefull shutdown

Definition at line 1289 of file sound4.x1.cloud_dyn.hpp.

14.14.3.31 Terminate()

```
int sound4::cloudx1::dyn::CInstance::Terminate ( )  [inline]
```

Terminate the processing.

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

Returns

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

Definition at line 1331 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.15 cloudx1_CAudioConverter Struct Reference

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

14.15.1 Detailed Description

Abstract holder for an audio conveerter

See also

cloudx1_GetAudioConverter

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

- sound4.x1.cloud.h

14.16 cloudx1_CClientInstance Struct Reference

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

14.16.1 Detailed Description

Abstract pointer for a client instance

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

- sound4.x1.cloud.h

14.17 cloudx1_CInstance Struct Reference

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

14.17.1 Detailed Description

Abstract holder for a process instance

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

- sound4.x1.cloud.h

14.18 cloudx1_CParameters Struct Reference

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

14.18.1 Detailed Description

Abstract holder for a process instance parameters

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

- sound4.x1.cloud.h

14.19 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

14.19.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 458 of file sound4.x1.cloud.hpp.

14.19.2 Constructor & Destructor Documentation

14.19.2.1 CPresetLoader()

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

14.19.2.2 ~CPresetLoader()

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

14.19.3 Member Function Documentation

14.19.3.1 Exists()

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

14.19.3.2 GetAll()

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

14.19.3.3 IsReadOnly()

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

14.19.3.4 Read()

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

14.19.3.5 Remove()

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

14.19.3.6 Rename()

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

14.19.3.7 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:

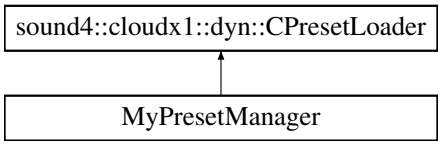
- sound4.x1.cloud.hpp

14.20 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

14.20.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

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

14.20.2 Constructor & Destructor Documentation

14.20.2.1 CPresetLoader()

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

14.20.2.2 ~CPresetLoader()

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

14.20.3 Member Function Documentation

14.20.3.1 Exists()

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

Implemented in MyPresetManager, and MyPresetManager.

14.20.3.2 GetAll()

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

Implemented in MyPresetManager, and MyPresetManager.

14.20.3.3 IsReadOnly()

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

Implemented in MyPresetManager, and MyPresetManager.

14.20.3.4 Read()

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

Implemented in MyPresetManager, and MyPresetManager.

14.20.3.5 Remove()

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

Implemented in MyPresetManager, and MyPresetManager.

14.20.3.6 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.

14.20.3.7 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:

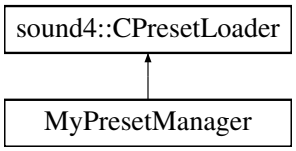
- sound4.x1.cloud_dyn.hpp

14.21 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

14.21.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 900 of file sound4cl.hpp.

14.21.2 Constructor & Destructor Documentation

14.21.2.1 CPresetLoader()

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

14.21.2.2 ~CPresetLoader()

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

14.21.3 Member Function Documentation

14.21.3.1 Exists()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.2 GetAll()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.3 IsReadOnly()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.4 Read()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.5 Remove()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.6 Rename()

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

Implemented in MyPresetManager, and MyPresetManager.

14.21.3.7 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

14.22 sound4::CProcessor Class Reference

Dynamic library interface.

`#include <sound4cl.hpp>`

Public Member Functions

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

  Loads the library.
- bool IsOk () const

  Check if the library was loaded correctly.
- void AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
- void AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
- void StereoToMono (const float *input, float *output, size_t nFrame)
- void MonoToStereo (const float *input, float *output, size_t nFrame)
- void StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void AudioMonoFromLiveStereo (const uint8_t *payload, float *output)
- void AudioMonoToLiveStereo (const float *input, uint8_t *payload)
- std::string GetVersion ()
- unsigned int GetChunkSizeInFrames ()
- std::vector< unsigned int > GetPossibleChunkSizeInFrames ()
- unsigned int GetChannelCount ()
- unsigned int GetAudioInputCount ()
- unsigned int GetAudioOutputCount ()
- unsigned int GetSampleRate ()
- std::string GetFormatName (const SampleFormat fmt)
- SampleFormat GetFormatFromName (const std::string &name)
- unsigned int GetBytesFromFormat (const SampleFormat fmt)
- void SetLogSeverity (LogSeverity severity)
- void SetLoggerCallback (log_cb_t cb)

Protected Member Functions

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

Friends

- class CBus
- class CInstance
- class CAudioConverter

14.22.1 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 314 of file sound4cl.hpp.

14.22.2 Constructor & Destructor Documentation

14.22.2.1 CProcessor()

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

14.22.3 Member Function Documentation

14.22.3.1 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 803 of file sound4cl.hpp.

14.22.3.2 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 813 of file sound4cl.hpp.

14.22.3.3 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 850 of file sound4cl.hpp.

14.22.3.4 GetChannelCount()

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

Get the number of channels needed by this process

Returns

The process channels count

Definition at line 793 of file sound4cl.hpp.

14.22.3.5 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 764 of file sound4cl.hpp.

14.22.3.6 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 841 of file sound4cl.hpp.

14.22.3.7 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 832 of file sound4cl.hpp.

14.22.3.8 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 775 of file sound4cl.hpp.

14.22.3.9 GetPrefixSymbol()

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

Definition at line 421 of file sound4cl.hpp.

14.22.3.10 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 823 of file sound4cl.hpp.

14.22.3.11 GetVersion()

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

Get the process version string

Returns

version string

Definition at line 751 of file sound4cl.hpp.

14.22.3.12 IsOk()

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

Returns

true if OK.

false if not OK.

Definition at line 638 of file sound4cl.hpp.

14.22.3.13 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 436 of file sound4cl.hpp.

14.22.3.14 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 862 of file sound4cl.hpp.

14.22.3.15 SetLogSeverity()

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

Sets the log severity (global)

Definition at line 856 of file sound4cl.hpp.

14.22.4 Friends And Related Symbol Documentation

14.22.4.1 CAudioConverter

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

Definition at line 322 of file sound4cl.hpp.

14.22.4.2 CBus

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

Definition at line 320 of file sound4cl.hpp.

14.22.4.3 CInstance

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

Definition at line 321 of file sound4cl.hpp.

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

- sound4cl.hpp

14.23 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

14.23.1 Detailed Description

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

Parameters

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

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

14.23.2 Constructor & Destructor Documentation

14.23.2.1 DynFuncHolder() [1/2]

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

14.23.2.2 DynFuncHolder() [2/2]

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

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

14.23.3 Member Function Documentation

14.23.3.1 IsOk()

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

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

14.23.3.2 operator bool()

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

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

14.23.3.3 operator T*()

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

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

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

- sound4.x1.cloud_dyn.hpp

14.24 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

14.24.1 Detailed Description

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

Parameters

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

Definition at line 134 of file sound4cl.hpp.

14.24.2 Constructor & Destructor Documentation

14.24.2.1 DynFuncHolder() [1/2]

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

14.24.2.2 DynFuncHolder() [2/2]

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

Definition at line 145 of file sound4cl.hpp.

14.24.3 Member Function Documentation

14.24.3.1 IsOk()

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

Definition at line 147 of file sound4cl.hpp.

14.24.3.2 operator bool()

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

Definition at line 146 of file sound4cl.hpp.

14.24.3.3 operator T*()

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

Definition at line 148 of file sound4cl.hpp.

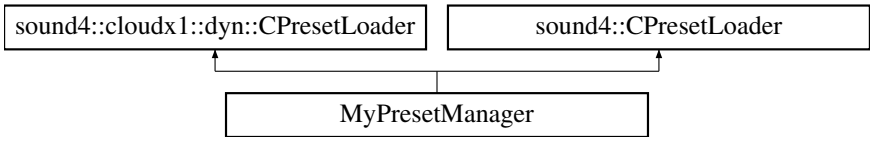
The documentation for this class was generated from the following file:

- sound4cl.hpp

14.25 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

14.25.1 Detailed Description

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

14.25.2 Constructor & Destructor Documentation

14.25.2.1 MyPresetManager() [1/2]

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

14.25.2.2 ~MyPresetManager() [1/2]

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

14.25.2.3 MyPresetManager() [2/2]

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

14.25.2.4 ~MyPresetManager() [2/2]

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

14.25.3 Member Function Documentation

14.25.3.1 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.

14.25.3.2 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.

14.25.3.3 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.

14.25.3.4 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.

14.25.3.5 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.

14.25.3.6 IsReadOnly() [2/2]

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

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

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

14.25.3.7 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.

14.25.3.8 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.

14.25.3.9 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.

14.25.3.10 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.

14.25.3.11 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.

14.25.3.12 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.

14.25.3.13 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.

14.25.3.14 SetupPath() [2/2]

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

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

14.25.3.15 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.

14.25.3.16 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

14.26 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

14.26.1 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 584 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

14.27 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

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

14.27.1 Detailed Description

template<typename T>
struct sound4::cloudx1::helper::SampleFormat< T >

Template Parameters

T	the C type for one audio sample

Definition at line 132 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

14.28 sound4::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

`#include <sound4cl.hpp>`

14.28.1 Detailed Description

template<typename T>
struct sound4::helper::SampleFormat< T >

Template Parameters

T	the C type for one audio sample

Definition at line 113 of file sound4cl.hpp.

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

- sound4cl.hpp

14.29 SOUND4_AuxiliaryOutputInfos Struct Reference

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

Public Attributes

- char * id
  The ID of the output (fixed across versions)
- char * description
  The description of the output.
- unsigned int sample_rate
  The sampling rate of this output.
- unsigned int channel_count
  The number of channels of this output.
- const float * buffer
  The output buffer of the output. See cloudx1_ProcessAudio for format.

14.29.1 Detailed Description

Structure holding information on an auxiliary output

Definition at line 811 of file sound4.x1.cloud.h.

14.29.2 Member Data Documentation

14.29.2.1 buffer

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

Definition at line 816 of file sound4.x1.cloud.h.

14.29.2.2 channel_count

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

Definition at line 815 of file sound4.x1.cloud.h.

14.29.2.3 description

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

Definition at line 813 of file sound4.x1.cloud.h.

14.29.2.4 id

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

Definition at line 812 of file sound4.x1.cloud.h.

14.29.2.5 sample_rate

```
unsigned int SOUND4_AuxiliaryOutputInfos::sample_rate
```

Definition at line 814 of file sound4.x1.cloud.h.

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

- sound4.x1.cloud.h

14.30 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.

14.30.1 Detailed Description

Information on this processor.

See also

SOUND4_GetProcessInfo

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

14.30.2 Member Data Documentation

14.30.2.1 name

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

Definition at line 79 of file sound4.x1.cloud.h.

14.30.2.2 prefix

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

Definition at line 80 of file sound4.x1.cloud.h.

14.30.2.3 shortname

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

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

14.30.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

Definition at line 78 of file sound4.x1.cloud.h.

14.30.2.5 version

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

Definition at line 82 of file sound4.x1.cloud.h.

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

- sound4.x1.cloud.h

Chapter 15

File Documentation

15.1 doc_examples.h File Reference

15.2 doc_examples.h

Go to the documentation of this file.

```
00001
00022 // ------------------------------------------------------------
```

15.3 doc_gencpp.h File Reference

Namespaces

- namespace sound4

15.4 doc_gencpp.h

Go to the documentation of this file.

```
00001 namespace sound4 {
00002
00059 }; // namespace sound4
```

15.5 doc_json.h File Reference

15.6 doc_json.h

Go to the documentation of this file.

```
00001
```

15.7 doc_main.h File Reference

15.8 doc_main.h

Go to the documentation of this file.

```
00001
```

15.9 doc_params.h File Reference

15.10 doc_params.h

Go to the documentation of this file.

```
00001
00002 // ------------------------------------------------------------
```

15.11 sound4.x1.cloud-pipe-cxx.cpp File Reference

C++ sample using C++ interface wrapper.

```
#include <unistd.h>
#include <iostream>
#include <strings.h>
#include <sys/types.h>
#include <filesystem>
#include <io.h>
#include <fcntl.h>
#include "sound4.x1.cloud.hpp"
```

Functions

- std::string GetEnv (const std::string &name)
- void ProcessLoop (sound4::cloudx1::CInstance &instance)
- void Usage (const char *progname)
- void MyLogger (sound4::cloudx1::LogSeverity severity, const std::string &msg)
- int main (int argc, const char **argv)

15.11.1 Function Documentation

15.11.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 41 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.2 main()

```
int main (
            int argc,
            const char ** argv )
```

[Setup and sanity checks]

[Setup and sanity checks]

[Create the processing instance]

[Create the processing instance]

[Communicating with the instance]

[Communicating with the instance]

Definition at line 106 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.3 MyLogger()

```
void MyLogger (
            sound4::cloudx1::LogSeverity severity,
            const std::string & msg )
```

Definition at line 101 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::cloudx1::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 50 of file sound4.x1.cloud-pipe-cxx.cpp.

15.11.1.5 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 81 of file sound4.x1.cloud-pipe-cxx.cpp.

15.12 sound4.x1.cloud-pipe-cxx.cpp

Go to the documentation of this file.

```
00001
00004 #include <unistd.h>
00005 #include <iostream>
00006 #include <strings.h>
00007 #include <sys/types.h>
00008 #include <filesystem>
00009 #if defined(_WIN32) || defined(WIN32)
00010     #include <io.h>
00011     #include <fcntl.h>
00012 #endif
00013
00014 #include "sound4.x1.cloud.hpp"
00015
00016 // ------------------------------------------------------------
00041 std::string GetEnv(const std::string& name)
00042 {
00043     char *data=getenv(name.c_str());
00044     if (!data) return {};
00045     return std::string(data);
00046 }
00047
00048
00049 // Reads from stdin, write to stdout
00050 void ProcessLoop(sound4::cloudx1::CInstance &instance)
00051 {
00052 bool stopthread=false;
00053
00054     #if defined(_WIN32) || defined(WIN32)
00055         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00056         _setmode(_fileno(stdin), _O_BINARY);
00057         _setmode(_fileno(stdout), _O_BINARY);
00058     #endif
00059
00061     while (!stopthread)
00062     {
00063         // Read a chunk from stdin
00064         if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::cloudx1::InputSampleSize,
      stdin) != sound4::cloudx1::InputSampleSize) {
00065             std::cerr « "ProcessLoop: input stopped"« std::endl;
00066             stopthread=true;
00067             break;
00068         }
00069         if (stopthread) break;
00070         // Process it
00071         instance.ProcessAudio();
00072         // Write it to stdout
00073         if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::cloudx1::OutputSampleSize,
      stdout) != sound4::cloudx1::OutputSampleSize) {
00074             std::cerr « "Failed to write audio output" « std::endl;
00075             stopthread=true;
00076         }
00077     }
00079 }
00080
00081 void Usage(const char *progname)
00082 {
00083     std::cerr « "Usage: " « progname « std::endl;;
00084     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00085     std::cerr « "Environment:"« std::endl;;
00086     std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
      std::endl;;
00087     std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00088     std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00089     std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00090     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00091     std::cerr « std::endl;
00092     std::cerr « "Example:"« std::endl;
00093     std::cerr « "ffmpeg -re -i file.wav -ar "«sound4::cloudx1::GetSampleRate()«" -ac
      "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00094     #if defined(_WIN32) || defined(WIN32)
00095         std::cerr « "ffmpeg -f dshow -i audio=\"Microphone\" -ar "«sound4::cloudx1::GetSampleRate()«"
      -ac "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00096     #else
00097         std::cerr « "ffmpeg -f pulse -i default -ar "«sound4::cloudx1::GetSampleRate()«" -ac
      "«sound4::cloudx1::GetChannelCount()«" -f f32le - | "«progname«" | ffplay -f f32le -ar
      "«sound4::cloudx1::GetSampleRate()«" -ac "«sound4::cloudx1::GetChannelCount()«" -i -" « std::endl;;
00098     #endif
00099 }
00100
00101 void MyLogger(sound4::cloudx1::LogSeverity severity,const std::string& msg)
```

```
00102 {
00103     std::cerr « msg « std::endl;
00104 }
00105
00106 int main(int argc, const char **argv)
00107 {
00108 #if (CLOUDX1_HAS_WEBSERVER)
00109     int listen_port = 8080;
00110 #endif // (CLOUDX1_HAS_WEBSERVER)
00111     auto store_path_str=GetEnv("S4STORE_PATH");
00112
00113     // Get credentials from environment
00114     auto LoginKey=GetEnv("S4LOGINKEY");
00115     if (LoginKey.empty()) {
00116         LoginKey=GetEnv("S4USERID");
00117     }
00118     auto RadioName=GetEnv("RADIO_NAME");
00119     auto KeyId=GetEnv("ACCESS_KEY_ID");
00120     auto AccessKey=GetEnv("ACCESS_KEY_SECRET");
00121
00122     const char *progname=argv[0];
00123
00124     if (argc>1) {
00125         if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
00126             Usage(progname);
00127             return 0;
00128         }
00129     }
00130     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00131         std::cerr « "Missing credentials in environment" « std::endl;
00132         Usage(progname);
00133         return 1;
00134     }
00135     std::filesystem::path store_path;
00136     if (!store_path_str.empty()) {
00137         std::error_code ec;
00138         store_path = std::filesystem::path(store_path_str);
00139         if (!std::filesystem::exists(store_path,ec) || !std::filesystem::is_directory(store_path,ec))
      {
00140             std::cerr « "Storage folder '"«(const char*)store_path.u8string().c_str()«"' is not a
      valid directory" « std::endl;
00141             Usage(progname);
00142             return 1;
00143         }
00144     }
00145
00147     sound4::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 '"«(const char*)store_path.u8string().c_str()«"' is not a
    valid directory" « std::endl;
00243             Usage(progname,dynlib);
00244             return 1;
00245         }
00246     }
00247 
00249     sound4::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 " « (const
    char*)store_path.u8string().c_str() « std::endl;
00262             return 1;
00263         }
00264         instance.SetPresetManager(&preset_manager);
00265     }
00267 
00268     std::cerr « "Creating" « std::endl;
00270     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00271         std::cerr « "Failed to create the processing instance" « std::endl;
00272         return 1;
00273     }
00275 
00276 #if (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 '"«(const char*)fs_store_path.u8string().c_str()«"' is not a valid
      directory" « std::endl;
00119         Usage(progname);
00120         return 1;
00121     }
00122     std::string store_path;
00123     #ifdef _WIN32
00124     {
00125         // Convert path to UTF-8
00126         std::wstring wstr(fs_store_path.wstring());
00127         if (wstr.empty()) return {};
00128         int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
00129         std::string str(size, 0);
00130         WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00131         store_path=str;
00132     }
00133     #else // !_WIN32
00134         store_path=fs_store_path;
00135     #endif // !_WIN32
00136
00138     // Optional: Use our own logging, instead of default color output to stderr.
00139     //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 '"«(const char*)store_path.u8string().c_str()«"' is not a
      valid directory" « std::endl;
00264             return 1;
00265         }
00266     }
00267
00269     processor.SetLogSeverity(sound4::verbose5);
00270     processor.SetLoggerCallback([](sound4::LogSeverity severity,const std::string& msg) {
00271             std::cerr « msg « std::endl;
00272         });
00273
00274     sound4::CInstance instance(processor);
00275     instance.SetParam("ADMIN_USER","admin");
00276     instance.SetParam("ADMIN_SECRET","admin");
00278
00280     MyPresetManager preset_manager;
00281     if (!store_path.empty()) {
00282         if (!preset_manager.SetupPath(store_path)) {
00283             std::cerr « "Failed to setup the custom preset manager on path " « (const
      char*)store_path.u8string().c_str() « std::endl;
00284             return 1;
00285         }
00286         instance.SetPresetManager(&preset_manager);
00287     }
00289
00290     std::cerr « "Creating" « std::endl;
00292     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00293         std::cerr « "Failed to create the processing instance" « std::endl;
00294         return 1;
00295     }
00297
00298     std::cerr « "Starting web server" « std::endl;
00300     if (!instance.StartWebServer(listen_port,0)) {
00301         std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
00302         return 1;
00303     }
00305
00306     {
```

```
00308         auto client = instance.NewClient();
00309         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00310         std::cerr « "Request for processor name returned "«answer«std::endl;
00312     }
00313
00314     ProcessLoop(instance, processor);
00315
00316     std::cerr « "Stopping now" « std::endl;
00317     instance.Stop();
00318
00319     std::cerr « "End" « std::endl;
00320     return 0;
00321 }
```

15.19 sound4.x1.cloud.h File Reference

C interface for library.

```
#include <windows.h>
#include <stdint.h>
```

Classes

- struct SOUND4_ProcessInfo
- struct SOUND4_AuxiliaryOutputInfos

Macros

- #define CLOUDX1_DECL __declspec(dllimport)
- #define CLOUDX1_HAS_WEBSERVER 1
- #define CLOUDX1_HAS_CLOUDBUS 0
- #define CLOUDX1_AUDIOFRAME_COUNT 64
- #define CLOUDX1_CHANNEL_COUNT 2
- #define CLOUDX1_AUDIO_INPUT_COUNT 1
- #define CLOUDX1_AUDIO_OUTPUT_COUNT 1
- #define CLOUDX1_SAMPLE_RATE 48000
- #define SOUND4_INVALID_WEBSERVER_ID ((uint64_t)-1)
- #define SOUND4_WEBSERVER_HTTP_OK (1<<0)
- #define SOUND4_WEBSERVER_HTTPS_OK (1<<1)

Typedefs

- typedef void(* cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *)
- typedef struct _tlgProvider_t const * TraceLoggingHProvider
- typedef wchar_t fs_char
- typedef char *(* cloudx1_storage_reader) (const fs_char *filename, void *userdata)
- typedef void(* cloudx1_storage_reader_free) (char *content, void *userdata)
- typedef int(* cloudx1_storage_writer) (const fs_char *filename, const char *content, void *userdata)
- typedef int(* cloudx1_storage_exists) (const fs_char *filename, void *userdata)
- typedef fs_char **(* cloudx1_storage_getall) (void *userdata)
- typedef void(* cloudx1_storage_getall_free) (fs_char **all, void *userdata)
- typedef int(* cloudx1_storage_remove) (const fs_char *filename, void *userdata)
- typedef int(* cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)

Enumerations

- enum cloudx1_LogSeverity {
  none = 0 , fatal = 1 , error = 2 , warning = 3 ,
  info = 4 , verbose = 5 , verbose2 = 6 , verbose3 = 7 ,
  verbose4 = 8 , verbose5 = 9 }
- enum cloudx1_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted }
- enum cloudx1_SampleFormat {
  INVALID_FORMAT =0 , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE =S16_LE , S24_NATIVE =S24_LE , S32_NATIVE =S32_LE ,
  F32_NATIVE =F32_LE }
- enum cloudx1_ConverterTargetMode {
  ConverterTargetMode_AsInput = 0 , ConverterTargetMode_Mono , ConverterTargetMode_Stereo ,
  ConverterTargetMode_StereoToMono ,
  ConverterTargetMode_MonoToStereo }

Functions

- const struct SOUND4_ProcessInfo * SOUND4_GetProcessInfo ()
- const char * cloudx1_GetVersion ()
- unsigned int cloudx1_GetChunkSizeInFrames ()
- unsigned int * cloudx1_GetPossibleChunkSizeInFrames ()
- unsigned int cloudx1_GetChannelCount ()
- unsigned int cloudx1_GetAudioInputCount ()
- unsigned int cloudx1_GetAudioOutputCount ()
- unsigned int cloudx1_GetSampleRate ()
- void cloudx1_SetLoggerCallback (cloudx1_loggerfn logger)
- void cloudx1_SetLogSeverity (enum cloudx1_LogSeverity severity)
- struct cloudx1_CParameters * cloudx1_NewParameters ()
- void cloudx1_FreeParameters (struct cloudx1_CParameters *params)
- void cloudx1_SetParameter (struct cloudx1_CParameters *params, const char *name, const char *value)
- const char * cloudx1_GetParameter (struct cloudx1_CParameters *params, const char *name)
- void cloudx1_FreeParameterValue (const char *value)
- struct cloudx1_CInstance * cloudx1_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct cloudx1_CInstance * cloudx1_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters)
- struct cloudx1_CInstance * cloudx1_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters, unsigned int frames_per_chunk)
- unsigned int cloudx1_GetProcessChunkFrames (struct cloudx1_CInstance *instance)
- int cloudx1_TerminateProcess (struct cloudx1_CInstance *instance)
- void cloudx1_ExitProcess (struct cloudx1_CInstance *instance)
- void cloudx1_SetInstanceTracing (struct cloudx1_CParameters *params, TraceLoggingHProvider tracing_↩ provider, GUID activity_guid)
- void cloudx1_SetInstanceTracingProcessActivity (struct cloudx1_CInstance *instance, GUID activity_guid)
- void cloudx1_SetPresetManager (struct cloudx1_CParameters *params, cloudx1_storage_reader reader, cloudx1_storage_reader_free, cloudx1_storage_writer writer, cloudx1_storage_exists exists, cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free, cloudx1_storage_remove remove, cloudx1_storage_rename rename, int IsReadOnly, void *userdata)
- void cloudx1_PresetManager_InformChange (struct cloudx1_CInstance *instance, const fs_char *relative↩ _path, enum cloudx1_PresetChange_Kind change_kind)

- void cloudx1_SetMetadata (struct cloudx1_CInstance *instance, const char *key, const char *value)
- void cloudx1_SetMetadataMulti (struct cloudx1_CInstance *instance, const char **keyvalue)
- const char ** cloudx1_GetMetadataInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeMetadataInfos (struct cloudx1_CInstance *instance, const char **infos)
- void cloudx1_StartUpdateThread (struct cloudx1_CInstance *instance, unsigned int port)
- void cloudx1_StopUpdateThread (struct cloudx1_CInstance *instance)
- int cloudx1_WaitUpdateThreadReady (struct cloudx1_CInstance *instance, int milliseconds)
- void cloudx1_ProcessAudio (struct cloudx1_CInstance *instance, const float *input, float *output)
- void cloudx1_ProcessAudio_Planar (struct cloudx1_CInstance *instance, float const *const *input, float *const *output)
- float * cloudx1_GetBufferIn (struct cloudx1_CInstance *instance)
- float * cloudx1_GetBufferOut (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetEstimatedDelay (struct cloudx1_CInstance *instance)
- struct SOUND4_AuxiliaryOutputInfos ** cloudx1_GetAuxiliaryAudioOutputInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeAuxiliaryAudioInfos (struct SOUND4_AuxiliaryOutputInfos **infos)
- int cloudx1_ActivateAuxiliaryAudioID (struct cloudx1_CInstance *instance, const char *id)
- unsigned int cloudx1_GetAuxiliaryAudioEstimatedDelay (struct cloudx1_CInstance *instance, const char *id)
- 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_CAudioConverter * cloudx1_GetAudioConverter (enum cloudx1_SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset, enum cloudx1_ConverterTargetMode target_mode)
- void cloudx1_FreeAudioConverter (struct cloudx1_CAudioConverter *converter)
- void cloudx1_AudioConverter_From (struct cloudx1_CAudioConverter *converter, const uint8_t *payload, float *output)
- void cloudx1_AudioConverter_To (struct cloudx1_CAudioConverter *converter, const float *input, uint8_↩ t *payload)
- struct cloudx1_CClientInstance * cloudx1_NewClient (struct cloudx1_CInstance *instance)
- void cloudx1_DeleteClient (struct cloudx1_CClientInstance *client)
- const char * cloudx1_ProcessJson (struct cloudx1_CClientInstance *client, const char *json_str, int *need↩ _save)
- void cloudx1_FreeJsonAnswer (const char *json_str)
- int cloudx1_SaveState (struct cloudx1_CInstance *instance)
- uint64_t cloudx1_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)
- uint64_t cloudx1_Webserver_tcp2 (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port, const struct cloudx1_CParameters *parameters)

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

15.19.1 Macro Definition Documentation

15.19.1.1 CLOUDX1_DECL

```
#define CLOUDX1_DECL __declspec(dllimport)
```

Definition at line 14 of file sound4.x1.cloud.h.

15.19.1.2 CLOUDX1_HAS_CLOUDBUS

```
#define CLOUDX1_HAS_CLOUDBUS 0
```

Definition at line 26 of file sound4.x1.cloud.h.

15.19.1.3 CLOUDX1_HAS_WEBSERVER

```
#define CLOUDX1_HAS_WEBSERVER 1
```

Definition at line 25 of file sound4.x1.cloud.h.

15.19.2 Typedef Documentation

15.19.2.1 TraceLoggingHProvider

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

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

15.20 sound4.x1.cloud.h

Go to the documentation of this file.

```
00001
00004 #pragma once
00005
00006 #ifdef __unix__
00007     #include <sys/types.h>
00008     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00009 #elif defined(_WIN32) || defined(WIN32)
00010     #include <windows.h>
00011     #ifdef BUILDING_CLOUDX1_DLL
00012         #define CLOUDX1_DECL    __declspec(dllexport)
00013     #else
00014         #define CLOUDX1_DECL    __declspec(dllimport)
00015     #endif
00016     #ifdef _MSC_VER
00017         #pragma warning(disable : 4996)
00018     #endif
00019 #elif defined(__APPLE__)
00020     #include <sys/types.h>
00021     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00022 #endif
00023 #include <stdint.h>
00024
00025 #define CLOUDX1_HAS_WEBSERVER 1
00026 #define CLOUDX1_HAS_CLOUDBUS 0
00027
00028 #ifdef __cplusplus
00029 extern "C" {
00030 #endif // __cplusplus
00031
00032 // ------------------------------------------------------------
00046 #define CLOUDX1_AUDIOFRAME_COUNT      64
00052 #define CLOUDX1_CHANNEL_COUNT          2
00058 #define CLOUDX1_AUDIO_INPUT_COUNT      1
00064 #define CLOUDX1_AUDIO_OUTPUT_COUNT     1
00070 #define CLOUDX1_SAMPLE_RATE            48000
00071
00077 struct SOUND4_ProcessInfo {
00078     size_t struct_size;
00079     const char *name;
00080     const char *prefix;
00081     const char *shortname;
00082     const char *version;
00083 };
00084
00092 CLOUDX1_DECL const struct SOUND4_ProcessInfo* SOUND4_GetProcessInfo();
00093
00099 CLOUDX1_DECL const char *cloudx1_GetVersion();
00100
00113 CLOUDX1_DECL unsigned int cloudx1_GetChunkSizeInFrames();
00114
00123 CLOUDX1_DECL unsigned int* cloudx1_GetPossibleChunkSizeInFrames();
00124
00132 CLOUDX1_DECL unsigned int cloudx1_GetChannelCount();
00133
00142 CLOUDX1_DECL unsigned int cloudx1_GetAudioInputCount();
00143
00152 CLOUDX1_DECL unsigned int cloudx1_GetAudioOutputCount();
00153
00161 CLOUDX1_DECL unsigned int cloudx1_GetSampleRate();
00162
00166 enum cloudx1_LogSeverity {
00167     none     = 0,
00168     fatal    = 1,
00169     error    = 2,
00170     warning  = 3,
00171     info     = 4,
00172     verbose  = 5,
00173     verbose2 = 6,
00174     verbose3 = 7,
00175     verbose4 = 8,
00176     verbose5 = 9,
00177 };
00178
00185 typedef void (*cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *);
00186
00197 CLOUDX1_DECL void cloudx1_SetLoggerCallback( cloudx1_loggerfn logger );
00198
00206 CLOUDX1_DECL void cloudx1_SetLogSeverity( enum cloudx1_LogSeverity severity );
00207
00210 // ------------------------------------------------------------
00218 // ------------------------------------------------------------
00222 struct cloudx1_CInstance;
```

```
00223
00227 struct cloudx1_CParameters;
00228
00236 CLOUDX1_DECL struct cloudx1_CParameters *cloudx1_NewParameters();
00237
00241 CLOUDX1_DECL void cloudx1_FreeParameters(struct cloudx1_CParameters *params);
00242
00254 CLOUDX1_DECL void cloudx1_SetParameter(struct cloudx1_CParameters *params, const char *name, const
      char *value);
00255
00261 CLOUDX1_DECL const char *cloudx1_GetParameter(struct cloudx1_CParameters *params, const char *name);
00262
00268 CLOUDX1_DECL void cloudx1_FreeParameterValue(const char *value);
00269
00277 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess(
00278         const char *LoginKey,
00279         const char *RadioName,
00280         const char *Access_Key_ID,
00281         const char *Access_Key_Secret,
00282         const char *save_path
00283     );
00284
00292 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess2(
00293         const char *LoginKey,
00294         const char *RadioName,
00295         const char *Access_Key_ID,
00296         const char *Access_Key_Secret,
00297         const char *save_path,
00298         const struct cloudx1_CParameters *parameters
00299     );
00300
00328 CLOUDX1_DECL struct cloudx1_CInstance *cloudx1_InitProcess3(
00329         const char *LoginKey,
00330         const char *RadioName,
00331         const char *Access_Key_ID,
00332         const char *Access_Key_Secret,
00333         const char *save_path,
00334         const struct cloudx1_CParameters *parameters,
00335         unsigned int frames_per_chunk
00336     );
00337
00349 CLOUDX1_DECL unsigned int cloudx1_GetProcessChunkFrames(struct cloudx1_CInstance *instance);
00350
00360 CLOUDX1_DECL int cloudx1_TerminateProcess(struct cloudx1_CInstance *instance);
00361
00367 CLOUDX1_DECL void cloudx1_ExitProcess(struct cloudx1_CInstance *instance);
00368
00371 #if CLOUDX1_HAS_CLOUDBUS
00372 // ---------------------------------------------------------------
00382 struct cloudx1_CBus;
00383
00389 CLOUDX1_DECL struct cloudx1_CBus* cloudx1_NewBus();
00390
00396 CLOUDX1_DECL void cloudx1_FreeBus(struct cloudx1_CBus *bus);
00397
00407 CLOUDX1_DECL void cloudx1_SetInstanceBus(struct cloudx1_CParameters *params, struct cloudx1_CBus
      *bus);
00408
00409
00411 #endif // CLOUDX1_HAS_CLOUDBUS
00412
00413 // ---------------------------------------------------------------
00414 //   Tracing (Windows only)
00415 // ---------------------------------------------------------------
00416 #ifdef _WIN32
00417 #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00418     struct _tlgProvider_t;
00419     typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00420 #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00421 // ---------------------------------------------------------------
00436 CLOUDX1_DECL void cloudx1_SetInstanceTracing(struct cloudx1_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00437
00445 CLOUDX1_DECL void cloudx1_SetInstanceTracingProcessActivity(struct cloudx1_CInstance *instance, GUID
      activity_guid);
00446
00448 #endif // _WIN32
00449
00450 // ---------------------------------------------------------------
00473 // ---------------------------------------------------------------
00474
00475 // OS dependent filesystem path char type
00476 #ifdef _WIN32
00477     typedef wchar_t fs_char;
00478 #else // !_WIN32
00479     typedef char fs_char;
00480 #endif // !_WIN32
```

```
00481
00489 typedef char * (*cloudx1_storage_reader) (const fs_char *filename, void* userdata);
00490
00497 typedef void (*cloudx1_storage_reader_free) (char *content, void* userdata);
00498
00507 typedef int (*cloudx1_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00508
00516 typedef int (*cloudx1_storage_exists) (const fs_char *filename, void* userdata);
00517
00525 typedef fs_char** (*cloudx1_storage_getall) (void* userdata);
00526
00533 typedef void (*cloudx1_storage_getall_free) (fs_char**all, void* userdata);
00534
00542 typedef int (*cloudx1_storage_remove) (const fs_char *filename, void* userdata);
00543
00552 typedef int (*cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00553
00571 CLOUDX1_DECL void cloudx1_SetPresetManager(struct cloudx1_CParameters *params
00572     , cloudx1_storage_reader reader, cloudx1_storage_reader_free
00573     , cloudx1_storage_writer writer
00574     , cloudx1_storage_exists exists
00575     , cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free
00576     , cloudx1_storage_remove remove
00577     , cloudx1_storage_rename rename
00578     , int IsReadOnly
00579     , void* userdata
00580     );
00581
00585 enum cloudx1_PresetChange_Kind {
00586     change_kind_created = 1,
00587     change_kind_modified   ,
00588     change_kind_deleted    ,
00589 };
00590
00599 CLOUDX1_DECL void cloudx1_PresetManager_InformChange(struct cloudx1_CInstance *instance, const fs_char
      *relative_path, enum cloudx1_PresetChange_Kind change_kind);
00600
00603 // ----------------------------------------------------------------
00611 // ----------------------------------------------------------------
00612
00629 CLOUDX1_DECL void cloudx1_SetMetadata(struct cloudx1_CInstance *instance, const char* key, const char*
      value);
00630
00641 CLOUDX1_DECL void cloudx1_SetMetadataMulti(struct cloudx1_CInstance *instance, const char** keyvalue);
00642
00650 CLOUDX1_DECL const char** cloudx1_GetMetadataInfos(struct cloudx1_CInstance *instance);
00651
00658 CLOUDX1_DECL void cloudx1_FreeMetadataInfos(struct cloudx1_CInstance *instance, const char** infos);
00659
00662 // ----------------------------------------------------------------
00669 // ----------------------------------------------------------------
00670
00683 CLOUDX1_DECL void cloudx1_StartUpdateThread(struct cloudx1_CInstance *instance, unsigned int port);
00684
00690 CLOUDX1_DECL void cloudx1_StopUpdateThread(struct cloudx1_CInstance *instance);
00691
00699 CLOUDX1_DECL int cloudx1_WaitUpdateThreadReady(struct cloudx1_CInstance *instance, int milliseconds);
00700
00703 // ----------------------------------------------------------------
00710 // ----------------------------------------------------------------
00732 CLOUDX1_DECL void cloudx1_ProcessAudio(struct cloudx1_CInstance *instance, const float *input, float
      *output );
00733
00752 CLOUDX1_DECL void cloudx1_ProcessAudio_Planar(struct cloudx1_CInstance *instance
00753     , float const * const *input
00754     , float * const *output);
00755
00768 CLOUDX1_DECL float *cloudx1_GetBufferIn(struct cloudx1_CInstance *instance);
00769
00782 CLOUDX1_DECL float *cloudx1_GetBufferOut(struct cloudx1_CInstance *instance);
00783
00793 CLOUDX1_DECL unsigned int cloudx1_GetEstimatedDelay(struct cloudx1_CInstance *instance);
00794
00797 // ----------------------------------------------------------------
00807 // ----------------------------------------------------------------
00811 struct SOUND4_AuxiliaryOutputInfos {
00812     char *id;
00813     char *description;
00814     unsigned int sample_rate;
00815     unsigned int channel_count;
00816     const float *buffer;
00817 };
00818
00826 CLOUDX1_DECL struct SOUND4_AuxiliaryOutputInfos** cloudx1_GetAuxiliaryAudioOutputInfos(struct
      cloudx1_CInstance *instance);
00827
00833 CLOUDX1_DECL void cloudx1_FreeAuxiliaryAudioInfos(struct SOUND4_AuxiliaryOutputInfos** infos);
```

```
00834
00844 CLOUDX1_DECL int cloudx1_ActivateAuxiliaryAudioID(struct cloudx1_CInstance *instance, const char* id);
00845
00857 CLOUDX1_DECL unsigned int cloudx1_GetAuxiliaryAudioEstimatedDelay(struct cloudx1_CInstance *instance,
      const char* id);
00858
00861 // ------------------------------------------------------------
00870 // ------------------------------------------------------------
00871     #ifdef _MSC_VER
00872         #define __BYTE_ORDER__  REG_DWORD
00873         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00874         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00875     #endif
00876
00880 enum cloudx1_SampleFormat {
00881     INVALID_FORMAT=0,
00882     S16_LE,
00883     S16_BE,
00884     S24_LE,
00885     S24_BE,
00886     S32_LE,
00887     S32_BE,
00888     F32_LE,
00889     F32_BE,
00890
00891     // Native definitions, only links to the physical ones
00892     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00893         S16_NATIVE=S16_LE,
00894         S24_NATIVE=S24_LE,
00895         S32_NATIVE=S32_LE,
00896         F32_NATIVE=F32_LE,
00897     #else
00898         S16_NATIVE=S16_BE,
00899         S24_NATIVE=S24_BE,
00900         S32_NATIVE=S32_BE,
00901         F32_NATIVE=F32_BE,
00902     #endif
00903 };
00904
00911 CLOUDX1_DECL const char *cloudx1_GetFormatName(const enum cloudx1_SampleFormat fmt);
00912
00919 CLOUDX1_DECL enum cloudx1_SampleFormat cloudx1_GetFormatFromName(const char *name);
00920
00927 CLOUDX1_DECL unsigned int cloudx1_GetBytesFromFormat(const enum cloudx1_SampleFormat fmt);
00928
00929
00936 CLOUDX1_DECL unsigned int cloudx1_GetMaxPacketFrame(struct cloudx1_CInstance *instance);
00937
00948 CLOUDX1_DECL unsigned int cloudx1_AddAudio(struct cloudx1_CInstance *instance, const uint8_t *payload,
      unsigned int nFrame, enum cloudx1_SampleFormat fmt );
00949
00956 CLOUDX1_DECL unsigned int cloudx1_AddPadAudio(struct cloudx1_CInstance *instance);
00957
00964 CLOUDX1_DECL unsigned int cloudx1_GetOutputCount(struct cloudx1_CInstance *instance);
00965
00975 CLOUDX1_DECL unsigned int cloudx1_GetAudio(struct cloudx1_CInstance *instance, uint8_t *payload,
      unsigned int max_nFrame, enum cloudx1_SampleFormat fmt );
00976
00980 // ------------------------------------------------------------
00987 // ------------------------------------------------------------
00998 CLOUDX1_DECL void cloudx1_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
00999
01010 CLOUDX1_DECL void cloudx1_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      cloudx1_SampleFormat fmt);
01011
01021 CLOUDX1_DECL void cloudx1_StereoToMono(const float *input, float *output, size_t nFrame);
01022
01032 CLOUDX1_DECL void cloudx1_MonoToStereo(const float *input, float *output, size_t nFrame);
01033
01044 CLOUDX1_DECL void cloudx1_StereoToMono_Planar(const float *inputL, const float *inputR, float *output,
      size_t nFrame);
01045
01056 CLOUDX1_DECL void cloudx1_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame);
01057
01064 CLOUDX1_DECL void cloudx1_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
01065
01072 CLOUDX1_DECL void cloudx1_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
01073
01079 struct cloudx1_CAudioConverter;
01080
01084 enum cloudx1_ConverterTargetMode {
01085     ConverterTargetMode_AsInput = 0,
01086     ConverterTargetMode_Mono,
01087     ConverterTargetMode_Stereo,
01088     ConverterTargetMode_StereoToMono,
```

```
01089     ConverterTargetMode_MonoToStereo,
01090 };
01091
01113 CLOUDX1_DECL struct cloudx1_CAudioConverter* cloudx1_GetAudioConverter(enum cloudx1_SampleFormat fmt,
      size_t nFrame
01114     , size_t payl_ch, size_t payl_ch_offset
01115     , enum cloudx1_ConverterTargetMode target_mode);
01116
01120 CLOUDX1_DECL void cloudx1_FreeAudioConverter(struct cloudx1_CAudioConverter *converter);
01121
01131 CLOUDX1_DECL void cloudx1_AudioConverter_From(struct cloudx1_CAudioConverter *converter, const uint8_t
      *payload, float *output);
01132
01142 CLOUDX1_DECL void cloudx1_AudioConverter_To(struct cloudx1_CAudioConverter *converter, const float
      *input, uint8_t *payload);
01143
01146 // ------------------------------------------------------------
01153 // ------------------------------------------------------------
01157 struct cloudx1_CClientInstance;
01158
01167 CLOUDX1_DECL struct cloudx1_CClientInstance *cloudx1_NewClient(struct cloudx1_CInstance *instance);
01168
01174 CLOUDX1_DECL void cloudx1_DeleteClient(struct cloudx1_CClientInstance *client);
01175
01189 CLOUDX1_DECL const char *cloudx1_ProcessJson(struct cloudx1_CClientInstance *client, const char
      *json_str, int *need_save);
01190
01196 CLOUDX1_DECL void cloudx1_FreeJsonAnswer(const char *json_str);
01197
01209 CLOUDX1_DECL int cloudx1_SaveState(struct cloudx1_CInstance *instance);
01210
01213 #if CLOUDX1_HAS_WEBSERVER
01214 // ------------------------------------------------------------
01232 // ------------------------------------------------------------
01240 CLOUDX1_DECL uint64_t cloudx1_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
      const char *socket_ip, unsigned int socket_port);
01241
01242 // ------------------------------------------------------------
01262 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);
01263
01283 CLOUDX1_DECL uint64_t cloudx1_Webserver(unsigned int listenport, unsigned int listenport_secure,
      struct cloudx1_CInstance *instance);
01284
01294 CLOUDX1_DECL int cloudx1_Webserver_Stop(uint64_t id, int timeout_ms);
01295
01304 CLOUDX1_DECL int cloudx1_Webserver_Status(uint64_t id);
01305
01316 CLOUDX1_DECL void cloudx1_Webserver_SetAppHealth(struct cloudx1_CInstance *instance, int httpcode,
      const char *contenttype, const char *content);
01317
01326 CLOUDX1_DECL void cloudx1_Webserver_GetAppHealth(struct cloudx1_CInstance *instance, int* httpcode,
      char** contenttype, char** content);
01327
01333 CLOUDX1_DECL void cloudx1_Webserver_FreeString(char *str);
01334
01339 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
01340
01344 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
01348 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
01349
01351 #endif // CLOUDX1_HAS_WEBSERVER
01352
01353 #ifdef __cplusplus
01354 } // extern "C"
01355 #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::AuxiliaryOutputInfos
- 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
- class sound4::cloudx1::CAudioConverter

  Audio converter helper.

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.CL"
- 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 1198 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 1203 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 1213 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 1184 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 1189 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 1220 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 1225 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 1193 of file sound4.x1.cloud.hpp.

15.22 sound4.x1.cloud.hpp

Go to the documentation of this file.

```
00001
00004 #pragma once
00005
00006 #include <string>
00007 #include <exception>
00008 #include <thread>
00009 #include <functional>
00010 #include <memory>
00011 #include <array>
00012 #include <assert.h>
00013 #include <stdexcept>
00014 #include <filesystem>
00015 #include <string.h>
00016
00017 #include "sound4.x1.cloud.h"
00018
00019
00020 // Filesystem path char type
00021 #ifdef _WIN32
00022     #include <stringapiset.h>
00023     #include <wchar.h>
00025     #define fs_strdup wcsdup
00026     #ifdef _MSC_VER
00027         #pragma warning(disable : 4996)
00028     #endif
00029 #else // !_WIN32
00031     #define fs_strdup strdup
00032 #endif // !_WIN32
00033
00034 // bridge C callbacks to CPresetLoader
00035 extern "C" {
00036     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00037     static void cloudx1_custom_reader_free(char *content, void* userdata);
00038     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00039     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00040     static fs_char** cloudx1_custom_getall(void* userdata);
00041     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00042     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00043     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00044 };
00045
00049 namespace sound4 {
00050     struct AuxiliaryOutputInfos {
00051         std::string id;
00052         std::string description;
00053         unsigned int sample_rate;
00054         unsigned int channel_count;
00055         const float *buffer;
00056     };
00060 namespace cloudx1 {
00061     static constexpr const char *process_name = "SOUND4 x1.CL";
00062     static constexpr const char *process_shortname = "x1";
00063     using SampleFormat = cloudx1_SampleFormat;
00068     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00073     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00078     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00083     const size_t InputSampleSize=ChunkFrames*InputChannels;
00088     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00093     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
00094
00098     namespace helper {
00099         #ifdef _WIN32
00105         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00106             if (wstr.empty()) return {};
00107             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
      NULL);
00108             std::string str(size, 0);
00109             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00110             return str;
00111         }
00117         static inline std::wstring UTF8ToWString(const std::string& str) {
00118             if (str.empty()) return std::wstring();
00119             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00120             std::wstring wstr(size, 0);
00121             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00122             return wstr;
00123         }
00124         #endif // _WIN32
00125
00131             template<typename T>
00132         struct SampleFormat {
00133         };
```

```
00134             template<>
00135         struct SampleFormat<int16_t> {
00136             const cloudx1_SampleFormat format = S16_NATIVE;
00137         };
00138             template<>
00139         struct SampleFormat<int32_t> {
00140             const cloudx1_SampleFormat format = S32_NATIVE;
00141         };
00142             template<>
00143         struct SampleFormat<float> {
00144             const cloudx1_SampleFormat format = F32_NATIVE;
00145         };
00146 
00147         // ------------------------------------------------------------
00158         static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00159             cloudx1_AudioConvertFrom(payload, output, nSpl, fmt);
00160         }
00161 
00172         static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl,
      cloudx1_SampleFormat fmt) {
00173             cloudx1_AudioConvertTo(input, payload, nSpl, fmt);
00174         }
00175 
00185         static inline void StereoToMono(const float *input, float *output, size_t nFrame) {
00186             cloudx1_StereoToMono(input, output, nFrame);
00187         }
00188 
00198         static inline void MonoToStereo(const float *input, float *output, size_t nFrame) {
00199             cloudx1_MonoToStereo(input, output, nFrame);
00200         }
00201 
00212         static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float
      *output, size_t nFrame) {
00213             cloudx1_StereoToMono_Planar(inputL, inputR, output, nFrame);
00214         }
00215 
00226         static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
      size_t nFrame) {
00227             cloudx1_MonoToStereo_Planar(input, outputL, outputR, nFrame);
00228         }
00229 
00236         static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
00237             cloudx1_AudioMonoFromLiveStereo(payload, output);
00238         }
00239 
00246         static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
00247             cloudx1_AudioMonoToLiveStereo(input, payload);
00248         }
00249 
00250     }; // namespace helper
00251 
00257     static inline std::string GetVersion()
00258         {
00259             return std::string(cloudx1_GetVersion());
00260         }
00261 
00272     static inline unsigned int GetChunkSizeInFrames()
00273         {
00274             return cloudx1_GetChunkSizeInFrames();
00275         }
00276 
00285     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00286         {
00287             std::vector<unsigned int> list;
00288             for (unsigned int* src=cloudx1_GetPossibleChunkSizeInFrames();*src;src++) {
00289                 list.push_back(*src);
00290             }
00291             return list;
00292         }
00293 
00299     static inline unsigned int GetChannelCount()
00300         {
00301             return cloudx1_GetChannelCount();
00302         }
00303 
00311     static inline unsigned int GetAudioInputCount()
00312         {
00313             return cloudx1_GetAudioInputCount();
00314         }
00315 
00323     static inline unsigned int GetAudioOutputCount()
00324         {
00325             return cloudx1_GetAudioOutputCount();
00326         }
00327 
00335     static inline unsigned int GetSampleRate()
```

```
00336         {
00337             return cloudx1_GetSampleRate();
00338         }
00339
00345     static inline void SanityCheck(bool a_bCheckFrames = true)
00346     {
00347         // Do some sanity checks
00348         if (cloudx1_GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00349             throw std::runtime_error("Bad library sampling rate");
00350         }
00351         if (a_bCheckFrames && cloudx1_GetChunkSizeInFrames()!=ChunkFrames) {
00352             throw std::runtime_error("Bad library frame size");
00353         }
00354         if (cloudx1_GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00355             throw std::runtime_error("Bad library channel count");
00356         }
00357         if (cloudx1_GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00358             throw std::runtime_error("Bad library input count");
00359         }
00360         if (cloudx1_GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00361             throw std::runtime_error("Bad library output count");
00362         }
00363     }
00364
00371     static inline std::string GetFormatName(const SampleFormat fmt)
00372         {
00373             return std::string(cloudx1_GetFormatName(fmt));
00374         }
00375
00382     static inline SampleFormat GetFormatFromName(const std::string& name)
00383         {
00384             return cloudx1_GetFormatFromName(name.c_str());
00385         }
00386
00393     static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
00394         {
00395             return cloudx1_GetBytesFromFormat(fmt);
00396         }
00397
00401     using LogSeverity = cloudx1_LogSeverity;
00405     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00406
00410     static log_cb_t _log_cb;
00411     extern "C" {
00412         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00413             _log_cb(severity, std::string(c_msg));
00414         }
00415     }
00419     static inline void SetLogSeverity(LogSeverity severity)
00420         {
00421             cloudx1_SetLogSeverity(severity);
00422         }
00426     static inline void SetLoggerCallback(log_cb_t cb)
00427         {
00428             _log_cb=cb;
00429             cloudx1_SetLoggerCallback(&_log_cb_c);
00430         }
00431
00432 #if CLOUDX1_HAS_CLOUDBUS
00438     class CBus {
00439         public:
00440             CBus()
00441                 : m_bus(cloudx1_NewBus()) {}
00442             ~CBus()
00443                 {
00444                     cloudx1_FreeBus(m_bus);
00445                     m_bus = nullptr;
00446                 }
00447             cloudx1_CBus *Get() const { return m_bus; }
00448         private:
00449             cloudx1_CBus *m_bus;
00450     };
00451 #endif // CLOUDX1_HAS_CLOUDBUS
00452
00458     class CPresetLoader {
00459     public:
00460         CPresetLoader() = default;
00461         virtual ~CPresetLoader() = default;
00462
00463         virtual bool IsReadOnly() = 0;
00464         virtual bool Exists(const std::filesystem::path &name) = 0;
00465         virtual bool Remove(const std::filesystem::path &name) = 0;
00466         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00467         virtual std::vector<std::filesystem::path> GetAll() = 0;
00468         virtual std::string Read(const std::filesystem::path &filename) = 0;
00469         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00470     };
```

```
00471     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00472
00478     class CInstance {
00479     public:
00483         CInstance()
00484             {
00485                 params = cloudx1_NewParameters();
00486                 if (!params) throw std::bad_alloc();
00487             }
00488         // non-copyable
00489         CInstance( const CInstance& ) = delete;
00490         CInstance& operator=( const CInstance& ) = delete;
00494         virtual ~CInstance()
00495             {
00496                 Stop();
00497                 if (params) cloudx1_FreeParameters(params);
00498             }
00502        bool IsOk() const { return instance!=nullptr; }
00512         void SetParam(const std::string& name, const std::string& value)
00513             {
00514                 assert(params);
00515                 cloudx1_SetParameter(params, name.c_str(), value.c_str());
00516             }
00523         std::string GetParam(const std::string& name)
00524             {
00525                 assert(params);
00526                 auto c_value=cloudx1_GetParameter(params, name.c_str());
00527                 std::string ret(c_value);
00528                 cloudx1_FreeParameterValue(c_value);
00529                 return ret;
00530             }
00531
00532 #if CLOUDX1_HAS_CLOUDBUS
00541             void SetBus(const CBus& bus) {
00542                 assert(params);
00543                 cloudx1_SetInstanceBus(params,bus.Get());
00544             }
00545 #endif // CLOUDX1_HAS_CLOUDBUS
00546
00557         void SetPresetManager(CPresetLoader *preset_manager)
00558             {
00559                 if (!preset_manager) return;
00560                 assert(params);
00561                 cloudx1_SetPresetManager(params,
00562                     cloudx1_custom_reader,
00563                     cloudx1_custom_reader_free,
00564                     cloudx1_custom_writer,
00565                     cloudx1_custom_exists,
00566                     cloudx1_custom_getall,
00567                     cloudx1_custom_getall_free,
00568                     cloudx1_custom_remove,
00569                     cloudx1_custom_rename,
00570                     preset_manager->IsReadOnly(),
00571                     preset_manager
00572                 );
00573             }
00574
00597         bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
      Access_Key_ID, const std::string& Access_Key_Secret, const std::filesystem::path& save_path, int
      json_port = 0, unsigned int frames_per_chunk=64)
00598             {
00599                 assert(!instance);
00600                 std::string l_save_path;
00601                 if (!save_path.empty()) {
00602                     #if (__cplusplus<202002L)
00603                     l_save_path = save_path.u8string();
00604                     #else // C++20
00605                     l_save_path = (const char*)save_path.u8string().c_str();
00606                     #endif // C++20
00607                 }
00608                 instance = 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);
00609                 if (!instance) return false;
00610                 if (!init_metadata.empty()) {
00611                     for (auto&& [key,value]: init_metadata) {
00612                         cloudx1_SetMetadata(instance, key.c_str(), value.c_str());
00613                     }
00614                     init_metadata.clear();
00615                 }
00616                 update_thread = std::thread([this, json_port](){
00617                     OnUpdateThreadStart();
00618                     cloudx1_StartUpdateThread(instance, json_port);
00619                     OnUpdateThreadStop();
00620                 });
00621                 int timeout=5000;
00622                 #ifdef DEBUG
00623                     timeout=60*1000; // For debugger pauses
```

```
00624                 #endif
00625                 if (cloudx1_WaitUpdateThreadReady(instance, timeout)<0) {
00626                     return false;
00627                 }
00628                 return true;
00629             }
00630
00638         unsigned int GetChunkFrames() {
00639             if (instance) {
00640                 return cloudx1_GetProcessChunkFrames(instance);
00641             }
00642             return 0;
00643         }
00644
00652         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
00653             {
00654                 cloudx1_PresetManager_InformChange(instance, relative_path.c_str(),
      (cloudx1_PresetChange_Kind) change_kind);
00655             }
00656
00668         void SetMetadata(const std::string& key, const char* value)
00669             {
00670                 if (instance) {
00671                     cloudx1_SetMetadata(instance, key.c_str(), value);
00672                 } else {
00673                     init_metadata.push_back( {key, value} );
00674                 }
00675             }
00676
00686         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
00687             {
00688                 if (instance) {
00689                     typedef const char* pchar;
00690                     pchar* keyvalue=new pchar[2*list.size()+1];
00691                     size_t n=0;
00692                     for (auto&& [key,value]: list) {
00693                         keyvalue[2*n+0]=key.c_str();
00694                         keyvalue[2*n+1]=value;
00695                         n++;
00696                     }
00697                     keyvalue[2*n+0]=nullptr;
00698
00699                     cloudx1_SetMetadataMulti(instance, keyvalue);
00700
00701                     delete[] keyvalue;
00702                 } else {
00703                     for (auto&& [key,value]: list) {
00704                         if (value) {
00705                             init_metadata.push_back( {key, value} );
00706                         }
00707                     }
00708                 }
00709             }
00710
00716         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
00717             {
00718                 std::vector< std::tuple<std::string,std::string> > values;
00719                 const char** c_values = cloudx1_GetMetadataInfos(instance);
00720                 if (c_values) {
00721                     for (const char** c_value=c_values; *c_value; ) {
00722                         std::string key(*c_value);
00723                         c_value++;
00724                         if (*c_value) {
00725                             std::string descr(*c_value);
00726                             c_value++;
00727                             values.push_back( {key,descr} );
00728                         }
00729                     }
00730                     cloudx1_FreeMetadataInfos(instance, c_values);
00731                 }
00732                 return values;
00733             }
00734
00740         std::vector<AuxiliaryOutputInfos> GetAuxiliaryAudioOutputInfos()
00741             {
00742                 std::vector<AuxiliaryOutputInfos> values;
00743                 SOUND4_AuxiliaryOutputInfos** c_values =
      cloudx1_GetAuxiliaryAudioOutputInfos(instance);
00744                 if (c_values) {
00745                     for (SOUND4_AuxiliaryOutputInfos** c_value=c_values; *c_value; c_value++) {
00746                         auto c_info=*c_value;
00747                         values.emplace_back( AuxiliaryOutputInfos{
00748                             .id            = c_info->id          ,
00749                             .description   = c_info->description  ,
00750                             .sample_rate   = c_info->sample_rate  ,
00751                             .channel_count = c_info->channel_count,
```

```
00752                             .buffer        = c_info->buffer       ,
00753                         });
00754                     }
00755                     cloudx1_FreeAuxiliaryAudioInfos(c_values);
00756                 }
00757                 return values;
00758             }
00759         void ActivateAuxiliaryAudioID(const std::string& id)
00760             {
00761                 cloudx1_ActivateAuxiliaryAudioID(instance, id.c_str());
00762             }
00766         virtual void OnUpdateThreadStart() {}
00770         virtual void OnUpdateThreadStop() {}
00771
00772 #if CLOUDX1_HAS_WEBSERVER
00786         bool StartWebServer(int http_port, int https_port=0)
00787             {
00788                 assert(instance);
00789                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00790                 webserver = cloudx1_Webserver(http_port,https_port,instance);
00791                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00792                     return false;
00793                 }
00794                 int web_status=cloudx1_Webserver_Status(webserver);
00795                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00796                     return false;
00797                 }
00798                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00799                     return false;
00800                 }
00801                 return true;
00802             }
00803
00811             void StopWebServer(int timeout_ms = 1000)
00812             {
00813                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00814                     cloudx1_Webserver_Stop(webserver, timeout_ms);
00815                     webserver = SOUND4_INVALID_WEBSERVER_ID;
00816                 }
00817             }
00818
00822         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
00823             {
00824                 assert(instance);
00825                 cloudx1_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
00826             }
00830         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
00831             {
00832                 assert(instance);
00833                 char* c_contenttype=nullptr;
00834                 char* c_content=nullptr;
00835                 cloudx1_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
00836                 contenttype=c_contenttype;
00837                 content=c_content;
00838                 cloudx1_Webserver_FreeString(c_contenttype);
00839                 cloudx1_Webserver_FreeString(c_content);
00840         }
00841 #endif // CLOUDX1_HAS_WEBSERVER
00842
00851         int Terminate()
00852             {
00853                 if (!instance) return 0;
00854                 return cloudx1_TerminateProcess(instance);
00855             }
00856
00860         void Stop()
00861             {
00862                 if (!instance) return;
00863 #if CLOUDX1_HAS_WEBSERVER
00864                 StopWebServer();
00865 #endif // CLOUDX1_HAS_WEBSERVER
00866                 if (update_thread.joinable()) {
00867                     cloudx1_StopUpdateThread(instance); // this will exit the update thread
00868                     update_thread.join();
00869                 }
00870                 cloudx1_ExitProcess(instance);
00871                 instance=nullptr;
00872             }
00873
00883         unsigned int GetEstimatedDelay()
00884             {
00885                 assert(instance);
00886                 return cloudx1_GetEstimatedDelay(instance);
00887             }
00888
```

```
00901         std::array<float,InputSampleSize>& GetBufferIn()
00902             {
00903                 assert(instance);
00904                 float *buf=cloudx1_GetBufferIn(instance);
00905                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
00906
00907             }
00919         std::array<float,OutputSampleSize>& GetBufferOut()
00920             {
00921                 assert(instance);
00922                 float *buf= cloudx1_GetBufferOut(instance);
00923                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00924             }
00928         void ProcessAudio()
00929             {
00930                 assert(instance);
00931                 cloudx1_ProcessAudio(instance, cloudx1_GetBufferIn(instance),
      cloudx1_GetBufferOut(instance));
00932             }
00944         void ProcessAudio_Planar(float const * const *input, float * const *output)
00945             {
00946                 assert(instance);
00947                 cloudx1_ProcessAudio_Planar(instance, input, output);
00948             }
00949
00961             template<typename T>
00962         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00963             {
00964                 assert(instance);
00965                 std::vector<T> output;
00966                 unsigned int out_offset=0;
00967                 unsigned int in_offset=0;
00968                 unsigned int todo = input.size();
00969                 while (todo>0) {
00970                     unsigned int left = AddAudio(&input[in_offset], todo);
00971                     unsigned int out_avail = cloudx1_GetOutputCount(instance);
00972                     output.resize(out_offset + out_avail);
00973                     GetAudio(&output[out_offset], out_avail);
00974                     out_offset+=out_avail;
00975                     in_offset += todo-left;
00976                     todo=left;
00977                 }
00978                 return output;
00979             }
00980
00986         class CClient {
00987             public:
00988                 CClient(cloudx1_CInstance* instance)
00989                     {
00990                         assert(instance);
00991                         client=cloudx1_NewClient(instance);
00992                         if (!client) throw std::bad_alloc();
00993                     }
00994                 // non-copyable
00995                 CClient( const CInstance& ) = delete;
00996                 CClient& operator=( const CInstance& ) = delete;
00997                 ~CClient()
00998                     {
00999                         if (client) {
01000                             cloudx1_DeleteClient(client);
01001                             client=nullptr;
01002                         }
01003                     }
01011                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01012                     {
01013                         assert(client);
01014                         int need_save=0;
01015                         const char *canswer = cloudx1_ProcessJson(client, request.c_str(),
      &need_save);
01016                         if (!canswer) return {};
01017                         std::string answer(canswer);
01018                         cloudx1_FreeJsonAnswer (canswer);
01019                         if (NeedSave) {
01020                             *NeedSave=(need_save!=0);
01021                         }
01022                         return answer;
01023                     }
01024             private:
01025                 cloudx1_CClientInstance *client = nullptr;
01026         };
01034         std::shared_ptr<CClient> NewClient()
01035             {
01036                 assert(instance);
01037                 return std::make_shared<CClient>(instance);
01038             }
01039
01050         bool SaveState() {
```

```
01051             assert(instance);
01052             if (cloudx1_SaveState(instance)!=0) {
01053                 return false;
01054             }
01055             return true;
01056         }
01057 #ifdef _WIN32
01064         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01065             cloudx1_SetInstanceTracing(params, tracing_provider, activity_guid);
01066         }
01073         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01074             assert(instance);
01075             cloudx1_SetInstanceTracingProcessActivity(instance, activity_guid);
01076         }
01077 #endif // _WIN32
01078     protected:
01079             template<typename T>
01080         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01081             {
01082                 assert(instance);
01083                 return cloudx1_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01084             }
01085             template<typename T>
01086         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01087             {
01088                 assert(instance);
01089                 return cloudx1_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01090             }
01091
01092     private:
01093         cloudx1_CParameters* params = nullptr;
01094         cloudx1_CInstance* instance = nullptr;
01095         std::thread update_thread;
01096         std::vector< std::pair<std::string,std::string> > init_metadata;
01097 #if CLOUDX1_HAS_WEBSERVER
01098         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01099 #endif // CLOUDX1_HAS_WEBSERVER
01100     };
01101
01108     class CAudioConverter {
01109         public:
01113             CAudioConverter() = default;
01114
01118             virtual ~CAudioConverter()
01119                 {
01120                     Delete();
01121                 }
01122
01138             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
      cloudx1_ConverterTargetMode target_mode)
01139                 {
01140                     Delete();
01141                     m_cvt = cloudx1_GetAudioConverter((cloudx1_SampleFormat)fmt, nFrame, payl_ch,
      payl_ch_offset, target_mode);
01142                     return IsOK();
01143                 }
01144             void Delete()
01145                 {
01146                     if (m_cvt) {
01147                         cloudx1_FreeAudioConverter(m_cvt);
01148                     }
01149                     m_cvt = nullptr;
01150                 }
01151             bool IsOK() const { return (m_cvt!=nullptr); }
01152
01161             void ConvertFrom(const uint8_t *payload, float *output)
01162                 { cloudx1_AudioConverter_From(m_cvt, payload, output); }
01163
01172             void ConvertTo(const float *input, uint8_t *payload)
01173                 { cloudx1_AudioConverter_To(m_cvt, input, payload); }
01174
01175         private:
01176             cloudx1_CAudioConverter *m_cvt = nullptr;
01177     };
01178
01179 }; // namespace cloudx1
01180 }; // namespace sound4
01181
01182 // bridge C callbacks to CPresetLoader
01183 extern "C" {
01184     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01185         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01186         std::string content=preset_loader->Read(filename);
01187         return strdup(content.c_str());
```

```
01188     }
01189     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01190         if (!content) return;
01191         free(content);
01192     }
01193     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01194         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01195         auto res=preset_loader->Write(filename,content);
01196         return res?0:-1;
01197     }
01198     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01199         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01200         auto res=preset_loader->Exists(filename);
01201         return res?0:-1;
01202     }
01203     static fs_char** cloudx1_custom_getall(void* userdata) {
01204         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01205         auto res=preset_loader->GetAll();
01206         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01207         for (size_t n=0;n<res.size();n++) {
01208             all[n]=fs_strdup(res[n].c_str());
01209         }
01210         all[res.size()]=nullptr;
01211         return all;
01212     }
01213     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01214         if (!all) return;
01215         for (fs_char** one=all;*one;one++) {
01216             free(*one);
01217         }
01218         free(all);
01219     }
01220     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01221         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01222         auto res=preset_loader->Remove(filename);
01223         return res?0:-1;
01224     }
01225     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01226         sound4::cloudx1::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
01227         auto res=preset_loader->Rename(from,to);
01228         return res?0:-1;
01229     }
01230
01231 };
```

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
- class sound4::cloudx1::dyn::CAudioConverter

  Audio converter helper.

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.CL"
- 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 1692 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 1697 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 1707 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 1678 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 1683 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 1714 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 1719 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 1687 of file sound4.x1.cloud_dyn.hpp.

15.24 sound4.x1.cloud_dyn.hpp

Go to the documentation of this file.

```
00001
00007 #pragma once
00008
00009 #include <string>
00010 #include <exception>
00011 #include <thread>
00012 #include <functional>
00013 #include <memory>
00014 #include <array>
00015 #include <assert.h>
00016 #include <stdexcept>
00017 #include <filesystem>
00018 #include <string.h>
00019
00020 #if defined(__unix__) || defined(__APPLE__)
00021     #define UNIXLIKE 1
00022 #else
00023     #define UNIXLIKE 0
00024 #endif
00025
00026 #if UNIXLIKE
00027     #include <dlfcn.h>
00028 #elif defined(_WIN32)
00029     #include <libloaderapi.h>
00030 #else
00031     #error "Unsupported OS"
00032 #endif
00033
00034 // Needed to have function declarations
00035 #include "sound4.x1.cloud.h"
00036
00037
00038 #ifdef _WIN32
00039     #include <stringapiset.h>
00040     #include <wchar.h>
00042     #define fs_strdup wcsdup
00043     #ifdef _MSC_VER
00044         #pragma warning(disable : 4996)
00045     #endif
00046 #else // !_WIN32
00048     #define fs_strdup strdup
00049 #endif // !_WIN32
00050
00051 // bridge C callbacks to CPresetLoader
00052 extern "C" {
00053     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00054     static void cloudx1_custom_reader_free(char *content, void* userdata);
00055     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00056     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
```

```
00057     static fs_char** cloudx1_custom_getall(void* userdata);
00058     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00059     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00060     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00061 };
00062
00066 namespace sound4 {
00070 namespace cloudx1 {
00074 namespace dyn {
00075
00079     namespace helper {
00080
00086             template <typename T>
00087         class DynFuncHolder {
00088             // Storage definition, depending on OS
00089             #if UNIXLIKE
00090                 using DynFunc_t = void *;
00091             #else
00092                 using DynFunc_t = FARPROC;
00093             #endif
00094         private:
00095             DynFunc_t m_ptr = NULL;
00096         public:
00097             DynFuncHolder() = default;
00098             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00099             operator bool() const { return (m_ptr!=NULL); }
00100             bool IsOk() const { return m_ptr!=NULL; }
00101             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00102         };
00103
00108         class CDynLoader {
00109             // Storage definition, depending on OS
00110             #if UNIXLIKE
00111                 using DynLib_t = void *;
00112             #else
00113                 using DynLib_t = HMODULE;
00114             #endif
00115             private:
00116                 DynLib_t m_lib {};
00117             public:
00118                 CDynLoader() = default;
00119                 CDynLoader(CDynLoader&& other) noexcept : m_lib(other.m_lib) { other.m_lib = NULL; }
00120                 CDynLoader& operator=(CDynLoader&& other) noexcept {
00121                     if (this != &other) { Close(); m_lib = other.m_lib; other.m_lib = NULL; }
00122                     return *this;
00123                 }
00124                 // No copy allowed
00125                 CDynLoader(const CDynLoader&) = delete;
00126                 CDynLoader& operator=(CDynLoader const&) = delete;
00127                 operator bool() const {
00128                     return (m_lib!=NULL);
00129                 }
00130                 bool IsOk() const {
00131                     return (m_lib!=NULL);
00132                 }
00133                 #if UNIXLIKE
00134                     void Close() {
00135                         if (IsOk()) {
00136                             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00137                             dlclose(m_lib);
00138                             #endif // __SANITIZE_ADDRESS__
00139                             m_lib=NULL;
00140                         }
00141                     }
00142                     bool Load(const std::filesystem::path& dynlib)
00143                     {
00144                         Close();
00145                         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00146                         return IsOk();
00147                     }
00148                         template <typename T>
00149                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00150                         auto ptr=dlsym(m_lib, name.c_str());
00151                         if (!ptr) {
00152                             throw std::runtime_error("Missing function in library");
00153                         }
00154                         return DynFuncHolder<T>(ptr);
00155                     }
00156                         template <typename T>
00157                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00158                         auto ptr=dlsym(m_lib, name.c_str());
00159                         return DynFuncHolder<T>(ptr);
00160                     }
00161                     static std::filesystem::path GetThisLibraryPath(void) {
00162                         Dl_info info;   // local: static would be a data race across concurrent
    callers
00163                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
```

```
00164                         auto rp=realpath(info.dli_fname, NULL);
00165                         if (!rp) {
00166                             #if defined(__unix__)
00167                                 rp=realpath("/proc/self/exe", NULL);
00168                             #elif defined(__APPLE__)
00169                                 // How to solve this for Apple ?
00170                             #endif
00171                         }
00172                         if (!rp) return {};
00173                         std::filesystem::path p(rp);
00174                         free(rp);
00175                         return p;
00176                     }
00177                 #else
00178                     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
00179                     {
00180                         Close();
00181                         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00182                         return IsOk();
00183                     }
00184                     void Close() {
00185                         if (IsOk()) {
00186                             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
    FreeLibraryAndExitThread but...
00187                             // So define SOUND4_CALL_FREELIBRARY before including this file if you
    really want to unload the library dynamically
00188                             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00189                                 FreeLibraryAndExitThread(m_lib, 0);
00190                             #elif defined(SOUND4_CALL_FREELIBRARY)
00191                                 FreeLibrary(m_lib);
00192                             #endif // SOUND4_CALL_FREELIBRARY
00193                             m_lib=NULL;
00194                         }
00195                     }
00196                         template <typename T>
00197                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00198                         auto ptr=GetProcAddress(m_lib, name.c_str());
00199                         if (!ptr) {
00200                             throw std::runtime_error("Missing function in library");
00201                         }
00202                         return DynFuncHolder<T>(ptr);
00203                     }
00204                         template <typename T>
00205                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00206                         auto ptr=GetProcAddress(m_lib, name.c_str());
00207                         return DynFuncHolder<T>(ptr);
00208                     }
00209                     static std::filesystem::path GetThisLibraryPath(void) {
00210                         static wchar_t path[MAX_PATH]={};
00211                         HMODULE hm = NULL;
00212                         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00213                                 GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00214                                 (LPCWSTR) &GetThisLibraryPath, &hm))
00215                         {
00216                             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00217                                 return std::filesystem::path(path);
00218                             }
00219                         }
00220                         return {};
00221                     }
00222                 #endif
00223                 virtual ~CDynLoader() {
00224                     Close();
00225                 }
00226         }; // class CDynLoader
00227     }; // namespace helper
00228
00235     class CDynLib {
00236     private:
00237         helper::CDynLoader m_lib;
00238         bool m_bOK = false;
00239     public:
00240         CDynLib() = default;
00241
00242         // All those wrapped functions have the same signature as there C cloudx1_XXX
00243         helper::DynFuncHolder< decltype(cloudx1_GetVersion             ) > GetVersion             ;
00244         helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames   ) > GetChunkSizeInFrames   ;
00245         helper::DynFuncHolder< decltype(cloudx1_GetChannelCount        ) > GetChannelCount        ;
00246         helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount     ) > GetAudioInputCount     ;
00247         helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount    ) > GetAudioOutputCount    ;
00248         helper::DynFuncHolder< decltype(cloudx1_GetSampleRate          ) > GetSampleRate          ;
00249         helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback      ) > SetLoggerCallback      ;
00250         helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity         ) > SetLogSeverity         ;
00251         helper::DynFuncHolder< decltype(cloudx1_NewParameters          ) > NewParameters          ;
00252         helper::DynFuncHolder< decltype(cloudx1_FreeParameters         ) > FreeParameters         ;
00253         helper::DynFuncHolder< decltype(cloudx1_SetParameter           ) > SetParameter           ;
```

```
00254         helper::DynFuncHolder< decltype(cloudx1_GetParameter           ) > GetParameter           ;
00255         helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue     ) > FreeParameterValue     ;
00256         helper::DynFuncHolder< decltype(cloudx1_InitProcess            ) > InitProcess            ;
00257         helper::DynFuncHolder< decltype(cloudx1_InitProcess2           ) > InitProcess2           ;
00258         helper::DynFuncHolder< decltype(cloudx1_TerminateProcess       ) > TerminateProcess       ;
00259         helper::DynFuncHolder< decltype(cloudx1_ExitProcess            ) > ExitProcess            ;
00260         helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread      ) > StartUpdateThread      ;
00261         helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread       ) > StopUpdateThread       ;
00262         helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady  ) > WaitUpdateThreadReady  ;
00263         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio           ) > ProcessAudio           ;
00264         helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar    ) > ProcessAudio_Planar    ;
00265         helper::DynFuncHolder< decltype(cloudx1_GetBufferIn            ) > GetBufferIn            ;
00266         helper::DynFuncHolder< decltype(cloudx1_GetBufferOut           ) > GetBufferOut           ;
00267         helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay      ) > GetEstimatedDelay      ;
00268         helper::DynFuncHolder< decltype(cloudx1_GetFormatName          ) > GetFormatName          ;
00269         helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName      ) > GetFormatFromName      ;
00270         helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat     ) > GetBytesFromFormat     ;
00271         helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame      ) > GetMaxPacketFrame      ;
00272         helper::DynFuncHolder< decltype(cloudx1_AddAudio               ) > AddAudio               ;
00273         helper::DynFuncHolder< decltype(cloudx1_AddPadAudio            ) > AddPadAudio            ;
00274         helper::DynFuncHolder< decltype(cloudx1_GetOutputCount         ) > GetOutputCount         ;
00275         helper::DynFuncHolder< decltype(cloudx1_GetAudio               ) > GetAudio               ;
00276         helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom       ) > AudioConvertFrom       ;
00277         helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo         ) > AudioConvertTo         ;
00278         helper::DynFuncHolder< decltype(cloudx1_StereoToMono           ) > StereoToMono           ;
00279         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo           ) > MonoToStereo           ;
00280         helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo;
00281         helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo  ) > AudioMonoToLiveStereo  ;
00282         helper::DynFuncHolder< decltype(cloudx1_NewClient              ) > NewClient              ;
00283         helper::DynFuncHolder< decltype(cloudx1_DeleteClient           ) > DeleteClient           ;
00284         helper::DynFuncHolder< decltype(cloudx1_ProcessJson            ) > ProcessJson            ;
00285         helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer         ) > FreeJsonAnswer         ;
00286         helper::DynFuncHolder< decltype(cloudx1_SaveState              ) > SaveState              ;
00287 #if CLOUDX1_HAS_WEBSERVER
00288         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp          ) > Webserver_tcp          ;
00289         helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2         ) > Webserver_tcp2         ;
00290         helper::DynFuncHolder< decltype(cloudx1_Webserver              ) > Webserver              ;
00291         helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop         ) > Webserver_Stop         ;
00292         helper::DynFuncHolder< decltype(cloudx1_Webserver_Status       ) > Webserver_Status       ;
00293 #endif // CLOUDX1_HAS_WEBSERVER
00294
00295         // added 2023-03-22
00296         helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar    ) > StereoToMono_Planar    ;
00297         helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar    ) > MonoToStereo_Planar    ;
00298
00299         // added 2023-05-22
00300         helper::DynFuncHolder< decltype(cloudx1_SetMetadata            ) > SetMetadata            ;
00301         helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos       ) > GetMetadataInfos       ;
00302         helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos      ) > FreeMetadataInfos      ;
00303
00304         // added 2023-06-20
00305         helper::DynFuncHolder< decltype(cloudx1_SetPresetManager               ) > SetPresetManager
    ;
00306         helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange     ) >
    PresetManager_InformChange     ;
00307
00308         // added 2023-09-21
00309         helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames   ) >
    GetPossibleChunkSizeInFrames   ;
00310         helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames          ) >
    GetProcessChunkFrames   ;
00311         helper::DynFuncHolder< decltype(cloudx1_InitProcess3                   ) > InitProcess3
    ;
00312
00313 #if CLOUDX1_HAS_CLOUDBUS
00314         // added 2023-06-26
00315         helper::DynFuncHolder< decltype(cloudx1_NewBus                 ) > NewBus      ;
00316         helper::DynFuncHolder< decltype(cloudx1_FreeBus                ) > FreeBus     ;
00317         helper::DynFuncHolder< decltype(cloudx1_SetInstanceBus         ) > SetInstanceBus      ;
00318 #endif // CLOUDX1_HAS_CLOUDBUS
00319
00320         // added 2024-02-19
00321         helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti       ) > SetMetadataMulti        ;
00322
00323 #if CLOUDX1_HAS_WEBSERVER
00324         // added 2024-05-23
00325         helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth    ) > Webserver_SetAppHealth
    ;
00326         helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth    ) > Webserver_GetAppHealth
    ;
00327         helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString      ) > Webserver_FreeString
    ;
00328 #endif // CLOUDX1_HAS_WEBSERVER
00329
00330 #ifdef _WIN32
00331         // added 2025-05-28
00332         helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracing             ) >
```

```
    SetInstanceTracing                ;
00333         helper::DynFuncHolder< decltype(cloudx1_SetInstanceTracingProcessActivity ) >
    SetInstanceTracingProcessActivity ;
00334 #endif // _WIN32
00335
00336         // added 2025-05-28
00337         helper::DynFuncHolder< decltype(cloudx1_GetAudioConverter      ) > GetAudioConverter      ;
00338         helper::DynFuncHolder< decltype(cloudx1_FreeAudioConverter     ) > FreeAudioConverter     ;
00339         helper::DynFuncHolder< decltype(cloudx1_AudioConverter_From    ) > AudioConverter_From    ;
00340         helper::DynFuncHolder< decltype(cloudx1_AudioConverter_To      ) > AudioConverter_To      ;
00341
00349         bool Load(const std::filesystem::path& path = {}) {
00350             std::string filename;
00351             #if defined(__APPLE__)
00352                 filename="libsound4.x1.cloud.dylib";
00353             #elif defined(__unix__)
00354                 filename="libsound4.x1.cloud.so";
00355             #else
00356                 filename="sound4.x1.cloud.dll";
00357             #endif
00358             m_lib.Close();
00359             // If a path is given, use it directly and do not try other path
00360             if (!path.empty() && !m_lib.Load(path / filename)) {
00361                 return false;
00362             } else if (!m_lib.IsOk()) {
00363                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00364                 #ifndef _WIN32
00365                 // Linux: if in a bin directory, try ../lib first
00366                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00367                     auto libdir = thisdir.parent_path() / "lib";
00368                     m_lib.Load(libdir / filename);
00369                 }
00370                 #endif // !_WIN32
00371                 if (!m_lib.IsOk()) {
00372                     // Search in the same directory this code is in
00373                     m_lib.Load(thisdir / filename);
00374                 }
00375                 if (!m_lib.IsOk()) {
00376                     // Try current path
00377                     std::error_code ec;
00378                     auto p = std::filesystem::current_path(ec);
00379                     if (!ec) {
00380                         m_lib.Load(p / filename);
00381                     }
00382                 }
00383                 if (!m_lib.IsOk()) {
00384                     return false;
00385                 }
00386             }
00387             // Load all C functions
00388             try {
00389                 GetVersion              = m_lib.GetSymbol< decltype(cloudx1_GetVersion             )
    >("cloudx1_GetVersion"             );
00390                 GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(cloudx1_GetChunkSizeInFrames   )
    >("cloudx1_GetChunkSizeInFrames"   );
00391                 GetChannelCount         = m_lib.GetSymbol< decltype(cloudx1_GetChannelCount        )
    >("cloudx1_GetChannelCount"        );
00392                 GetAudioInputCount      = m_lib.GetSymbol< decltype(cloudx1_GetAudioInputCount     )
    >("cloudx1_GetAudioInputCount"     );
00393                 GetAudioOutputCount     = m_lib.GetSymbol< decltype(cloudx1_GetAudioOutputCount    )
    >("cloudx1_GetAudioOutputCount"    );
00394                 GetSampleRate           = m_lib.GetSymbol< decltype(cloudx1_GetSampleRate          )
    >("cloudx1_GetSampleRate"          );
00395                 SetLoggerCallback       = m_lib.GetSymbol< decltype(cloudx1_SetLoggerCallback      )
    >("cloudx1_SetLoggerCallback"      );
00396                 SetLogSeverity          = m_lib.GetSymbol< decltype(cloudx1_SetLogSeverity         )
    >("cloudx1_SetLogSeverity"         );
00397                 NewParameters           = m_lib.GetSymbol< decltype(cloudx1_NewParameters          )
    >("cloudx1_NewParameters"          );
00398                 FreeParameters          = m_lib.GetSymbol< decltype(cloudx1_FreeParameters         )
    >("cloudx1_FreeParameters"         );
00399                 SetParameter            = m_lib.GetSymbol< decltype(cloudx1_SetParameter           )
    >("cloudx1_SetParameter"           );
00400                 GetParameter            = m_lib.GetSymbol< decltype(cloudx1_GetParameter           )
    >("cloudx1_GetParameter"           );
00401                 FreeParameterValue      = m_lib.GetSymbol< decltype(cloudx1_FreeParameterValue     )
    >("cloudx1_FreeParameterValue"     );
00402                 InitProcess             = m_lib.GetSymbol< decltype(cloudx1_InitProcess            )
    >("cloudx1_InitProcess"            );
00403                 InitProcess2            = m_lib.GetSymbol< decltype(cloudx1_InitProcess2           )
    >("cloudx1_InitProcess2"           );
00404                 TerminateProcess        = m_lib.GetSymbol< decltype(cloudx1_TerminateProcess       )
    >("cloudx1_TerminateProcess"       );
00405                 ExitProcess             = m_lib.GetSymbol< decltype(cloudx1_ExitProcess            )
    >("cloudx1_ExitProcess"            );
00406                 StartUpdateThread       = m_lib.GetSymbol< decltype(cloudx1_StartUpdateThread      )
    >("cloudx1_StartUpdateThread"      );
```

```
00407                 StopUpdateThread        = m_lib.GetSymbol< decltype(cloudx1_StopUpdateThread       )
      >("cloudx1_StopUpdateThread"       );
00408                 WaitUpdateThreadReady   = m_lib.GetSymbol< decltype(cloudx1_WaitUpdateThreadReady  )
      >("cloudx1_WaitUpdateThreadReady"  );
00409                 ProcessAudio            = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio           )
      >("cloudx1_ProcessAudio"           );
00410                 ProcessAudio_Planar     = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio_Planar    )
      >("cloudx1_ProcessAudio_Planar"    );
00411                 GetBufferIn             = m_lib.GetSymbol< decltype(cloudx1_GetBufferIn            )
      >("cloudx1_GetBufferIn"            );
00412                 GetBufferOut            = m_lib.GetSymbol< decltype(cloudx1_GetBufferOut           )
      >("cloudx1_GetBufferOut"           );
00413                 GetEstimatedDelay       = m_lib.GetSymbol< decltype(cloudx1_GetEstimatedDelay      )
      >("cloudx1_GetEstimatedDelay"      );
00414                 GetFormatName           = m_lib.GetSymbol< decltype(cloudx1_GetFormatName          )
      >("cloudx1_GetFormatName"          );
00415                 GetFormatFromName       = m_lib.GetSymbol< decltype(cloudx1_GetFormatFromName      )
      >("cloudx1_GetFormatFromName"      );
00416                 GetBytesFromFormat      = m_lib.GetSymbol< decltype(cloudx1_GetBytesFromFormat     )
      >("cloudx1_GetBytesFromFormat"     );
00417                 GetMaxPacketFrame       = m_lib.GetSymbol< decltype(cloudx1_GetMaxPacketFrame      )
      >("cloudx1_GetMaxPacketFrame"      );
00418                 AddAudio                = m_lib.GetSymbol< decltype(cloudx1_AddAudio               )
      >("cloudx1_AddAudio"               );
00419                 AddPadAudio             = m_lib.GetSymbol< decltype(cloudx1_AddPadAudio            )
      >("cloudx1_AddPadAudio"            );
00420                 GetOutputCount          = m_lib.GetSymbol< decltype(cloudx1_GetOutputCount         )
      >("cloudx1_GetOutputCount"         );
00421                 GetAudio                = m_lib.GetSymbol< decltype(cloudx1_GetAudio               )
      >("cloudx1_GetAudio"               );
00422                 AudioConvertFrom        = m_lib.GetSymbol< decltype(cloudx1_AudioConvertFrom       )
      >("cloudx1_AudioConvertFrom"       );
00423                 AudioConvertTo          = m_lib.GetSymbol< decltype(cloudx1_AudioConvertTo         )
      >("cloudx1_AudioConvertTo"         );
00424                 StereoToMono            = m_lib.GetSymbol< decltype(cloudx1_StereoToMono           )
      >("cloudx1_StereoToMono"           );
00425                 MonoToStereo            = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo           )
      >("cloudx1_MonoToStereo"           );
00426                 AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(cloudx1_AudioMonoFromLiveStereo)
      >("cloudx1_AudioMonoFromLiveStereo");
00427                 AudioMonoToLiveStereo   = m_lib.GetSymbol< decltype(cloudx1_AudioMonoToLiveStereo  )
      >("cloudx1_AudioMonoToLiveStereo"  );
00428                 NewClient               = m_lib.GetSymbol< decltype(cloudx1_NewClient              )
      >("cloudx1_NewClient"              );
00429                 DeleteClient            = m_lib.GetSymbol< decltype(cloudx1_DeleteClient           )
      >("cloudx1_DeleteClient"           );
00430                 ProcessJson             = m_lib.GetSymbol< decltype(cloudx1_ProcessJson            )
      >("cloudx1_ProcessJson"            );
00431                 FreeJsonAnswer          = m_lib.GetSymbol< decltype(cloudx1_FreeJsonAnswer         )
      >("cloudx1_FreeJsonAnswer"         );
00432                 SaveState               = m_lib.GetSymbol< decltype(cloudx1_SaveState              )
      >("cloudx1_SaveState"              );
00433 #if CLOUDX1_HAS_WEBSERVER
00434                 Webserver_tcp           = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp          )
      >("cloudx1_Webserver_tcp"          );
00435                 Webserver_tcp2          = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp2         )
      >("cloudx1_Webserver_tcp2"         );
00436                 Webserver               = m_lib.GetSymbol< decltype(cloudx1_Webserver              )
      >("cloudx1_Webserver"              );
00437                 Webserver_Stop          = m_lib.GetSymbol< decltype(cloudx1_Webserver_Stop         )
      >("cloudx1_Webserver_Stop"         );
00438                 Webserver_Status        = m_lib.GetSymbol< decltype(cloudx1_Webserver_Status       )
      >("cloudx1_Webserver_Status"       );
00439 #endif // CLOUDX1_HAS_WEBSERVER
00440
00441                 // added 2023-03-22
00442                 StereoToMono_Planar     = m_lib.GetSymbol< decltype(cloudx1_StereoToMono_Planar    )
      >("cloudx1_StereoToMono_Planar"    );
00443                 MonoToStereo_Planar     = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo_Planar    )
      >("cloudx1_MonoToStereo_Planar"    );
00444
00445             } catch (std::runtime_error& ) {
00446                 return false;
00447             }
00448             // C functions allowed to be missing
00449             try {
00450                 // added 2023-05-22
00451                 SetMetadata             = m_lib.GetSymbol< decltype(cloudx1_SetMetadata            )
      >("cloudx1_SetMetadata"            );
00452                 GetMetadataInfos        = m_lib.GetSymbol< decltype(cloudx1_GetMetadataInfos       )
      >("cloudx1_GetMetadataInfos"       );
00453                 FreeMetadataInfos       = m_lib.GetSymbol< decltype(cloudx1_FreeMetadataInfos      )
      >("cloudx1_FreeMetadataInfos"      );
00454                 // added 2023-06-20
00455                 SetPresetManager            = m_lib.GetSymbol< decltype(cloudx1_SetPresetManager
      ) >("cloudx1_SetPresetManager"           );
00456                 PresetManager_InformChange  = m_lib.GetSymbol<
```

```
    decltype(cloudx1_PresetManager_InformChange     ) >("cloudx1_PresetManager_InformChange"    );
00457
00458                 // added 2023-09-21
00459                 GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
    decltype(cloudx1_GetPossibleChunkSizeInFrames  ) >("cloudx1_GetPossibleChunkSizeInFrames"  );
00460                 GetProcessChunkFrames        = m_lib.GetSymbol< decltype(cloudx1_GetProcessChunkFrames
    ) >("cloudx1_GetProcessChunkFrames"         );
00461                 InitProcess3                 = m_lib.GetSymbol< decltype(cloudx1_InitProcess3
    ) >("cloudx1_InitProcess3"                  );
00462
00463 #if CLOUDX1_HAS_CLOUDBUS
00464                 // added 2023-06-26
00465                 NewBus                   = m_lib.GetSymbol< decltype(cloudx1_NewBus                 )
    >("cloudx1_NewBus"                );
00466                 FreeBus                  = m_lib.GetSymbol< decltype(cloudx1_FreeBus                )
    >("cloudx1_FreeBus"               );
00467                 SetInstanceBus           = m_lib.GetSymbol< decltype(cloudx1_SetInstanceBus         )
    >("cloudx1_SetInstanceBus"        );
00468 #endif // CLOUDX1_HAS_CLOUDBUS
00469
00470                 // added 2024-02-19
00471                 SetMetadataMulti         = m_lib.GetSymbol< decltype(cloudx1_SetMetadataMulti       )
    >("cloudx1_SetMetadataMulti"      );
00472
00473 #if CLOUDX1_HAS_WEBSERVER
00474                 // added 2024-05-23
00475                 Webserver_SetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_SetAppHealth
    ) >("Webserver_SetAppHealth"     );
00476                 Webserver_GetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_GetAppHealth
    ) >("Webserver_GetAppHealth"     );
00477                 Webserver_FreeString        = m_lib.GetSymbol< decltype(cloudx1_Webserver_FreeString
    ) >("Webserver_FreeString"       );
00478                 #endif // CLOUDX1_HAS_WEBSERVER
00479
00480 #ifdef _WIN32
00481                 // added 2025-05-28
00482                 SetInstanceTracing               = m_lib.GetSymbol<
    decltype(cloudx1_SetInstanceTracing               ) >("SetInstanceTracing"               );
00483                 SetInstanceTracingProcessActivity = m_lib.GetSymbol<
    decltype(cloudx1_SetInstanceTracingProcessActivity) >("SetInstanceTracingProcessActivity");
00484 #endif // _WIN32
00485
00486                 // added 2025-05-28
00487                 GetAudioConverter                = m_lib.GetSymbol<
    decltype(cloudx1_GetAudioConverter                ) >("GetAudioConverter"                );
00488                 FreeAudioConverter               = m_lib.GetSymbol<
    decltype(cloudx1_FreeAudioConverter               ) >("FreeAudioConverter"               );
00489                 AudioConverter_From              = m_lib.GetSymbol<
    decltype(cloudx1_AudioConverter_From              ) >("AudioConverter_From"              );
00490                 AudioConverter_To                = m_lib.GetSymbol<
    decltype(cloudx1_AudioConverter_To                ) >("AudioConverter_To"                );
00491
00492             }  catch (std::runtime_error& ) {
00493                 // Ignored, handler will take care
00494             }
00495
00496             m_bOK=true;
00497             return true;
00498         }
00505         bool IsOk() const {
00506             return m_bOK;
00507         }
00508
00509     };
00510
00511
00512
00513     static constexpr const char *process_name = "SOUND4 x1.CL";
00514     static constexpr const char *process_shortname = "x1";
00515     using SampleFormat = cloudx1_SampleFormat;
00520     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00525     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00530     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00535     const size_t InputSampleSize=ChunkFrames*InputChannels;
00540     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00545     const size_t SampleRate=CLOUDX1_SAMPLE_RATE;
00546
00550     namespace helper {
00551         #ifdef _WIN32
00557         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00558             if (wstr.empty()) return {};
00559             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
    NULL);
00560             std::string str(size, 0);
00561             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00562             return str;
00563         }
```

```
00569         static inline std::wstring UTF8ToWString(const std::string& str) {
00570             if (str.empty()) return std::wstring();
00571             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00572             std::wstring wstr(size, 0);
00573             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00574             return wstr;
00575         }
00576         #endif // _WIN32
00577
00583             template<typename T>
00584         struct SampleFormat {
00585         };
00586             template<>
00587         struct SampleFormat<int16_t> {
00588             const cloudx1_SampleFormat format = S16_NATIVE;
00589         };
00590             template<>
00591         struct SampleFormat<int32_t> {
00592             const cloudx1_SampleFormat format = S32_NATIVE;
00593         };
00594             template<>
00595         struct SampleFormat<float> {
00596             const cloudx1_SampleFormat format = F32_NATIVE;
00597         };
00598
00599         // ------------------------------------------------------------
00611         static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00612             dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00613         }
00614
00626         static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
      size_t nSpl, cloudx1_SampleFormat fmt) {
00627             dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00628         }
00629
00640         static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00641             dynlib.StereoToMono(input, output, nFrame);
00642         }
00643
00654         static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
      nFrame) {
00655             dynlib.MonoToStereo(input, output, nFrame);
00656         }
00657
00669         static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
      *inputR, float *output, size_t nFrame) {
00670             dynlib.StereoToMono_Planar(inputL, inputR, output, nFrame);
00671         }
00672
00684         static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
      float *outputR, size_t nFrame) {
00685             dynlib.MonoToStereo_Planar(input, outputL, outputR, nFrame);
00686         }
00687
00695         static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
      *output) {
00696             dynlib.AudioMonoFromLiveStereo(payload, output);
00697         }
00698
00706         static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
      *payload) {
00707             dynlib.AudioMonoToLiveStereo(input, payload);
00708         }
00709
00710     }; // namespace helper
00711
00718     static inline std::string GetVersion(CDynLib& dynlib)
00719         {
00720             return std::string(dynlib.GetVersion());
00721         }
00722
00734     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00735         {
00736             return dynlib.GetChunkSizeInFrames();
00737         }
00738
00748     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00749         {
00750             std::vector<unsigned int> list;
00751             if (dynlib.GetPossibleChunkSizeInFrames) {
00752                 for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00753                     list.push_back(*src);
00754                 }
00755             } else {
00756                 list.push_back(64);
```

```
00757             }
00758             return list;
00759         }
00760
00767     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00768         {
00769             return dynlib.GetChannelCount();
00770         }
00771
00780     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00781         {
00782             return dynlib.GetAudioInputCount();
00783         }
00784
00793     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00794         {
00795             return dynlib.GetAudioOutputCount();
00796         }
00797
00806     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00807         {
00808             return dynlib.GetSampleRate();
00809         }
00810
00818     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00819     {
00820         // Do some sanity checks
00821         if (dynlib.GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
00822             throw std::runtime_error("Bad library sampling rate");
00823         }
00824         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
00825             throw std::runtime_error("Bad library frame size");
00826         }
00827         if (dynlib.GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
00828             throw std::runtime_error("Bad library channel count");
00829         }
00830         if (dynlib.GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
00831             throw std::runtime_error("Bad library input count");
00832         }
00833         if (dynlib.GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
00834             throw std::runtime_error("Bad library output count");
00835         }
00836     }
00837
00845     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00846         {
00847             return std::string(dynlib.GetFormatName(fmt));
00848         }
00849
00857     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00858         {
00859             return dynlib.GetFormatFromName(name.c_str());
00860         }
00861
00869     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00870         {
00871             return dynlib.GetBytesFromFormat(fmt);
00872         }
00873
00877     using LogSeverity = cloudx1_LogSeverity;
00881     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00882
00886     static log_cb_t _log_cb;
00887     extern "C" {
00888         static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
00889             _log_cb(severity, std::string(c_msg));
00890         }
00891     }
00895     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00896         {
00897             dynlib.SetLogSeverity(severity);
00898         }
00902     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00903         {
00904             _log_cb=cb;
00905             dynlib.SetLoggerCallback(&_log_cb_c);
00906         }
00907
00908 #if CLOUDX1_HAS_CLOUDBUS
00914     class CBus {
00915         public:
00916             CBus(CDynLib& dynlib)
00917                 : m_dynlib(dynlib)
00918                 {
00919                     if (m_dynlib.NewBus) {
00920                         m_bus=m_dynlib.NewBus();
00921                     }
```

```
00922                 }
00923             ~CBus()
00924                 {
00925                     if (m_bus && m_dynlib.FreeBus) {
00926                         m_dynlib.FreeBus(m_bus);
00927                     }
00928                     m_bus = nullptr;
00929                 }
00930             cloudx1_CBus *Get() const { return m_bus; }
00931         private:
00932             CDynLib& m_dynlib;
00933             cloudx1_CBus *m_bus;
00934     };
00935 #endif // CLOUDX1_HAS_CLOUDBUS
00936
00942     class CPresetLoader {
00943     public:
00944         CPresetLoader() = default;
00945         virtual ~CPresetLoader() = default;
00946
00947         virtual bool IsReadOnly() = 0;
00948         virtual bool Exists(const std::filesystem::path &name) = 0;
00949         virtual bool Remove(const std::filesystem::path &name) = 0;
00950         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00951         virtual std::vector<std::filesystem::path> GetAll() = 0;
00952         virtual std::string Read(const std::filesystem::path &filename) = 0;
00953         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00954     };
00955     using PresetChange_Kind = cloudx1_PresetChange_Kind;
00956
00962     class CInstance {
00963     public:
00967         CInstance(CDynLib& dynlib)
00968         : m_dynlib(dynlib)
00969             {
00970                 params = m_dynlib.NewParameters();
00971                 if (!params) throw std::bad_alloc();
00972             }
00973         // non-copyable
00974         CInstance( const CInstance& ) = delete;
00975         CInstance& operator=( const CInstance& ) = delete;
00979         virtual ~CInstance()
00980             {
00981                 Stop();
00982                 if (params) m_dynlib.FreeParameters(params);
00983             }
00987        bool IsOk() const { return instance!=nullptr; }
00997         void SetParam(const std::string& name, const std::string& value)
00998             {
00999                 assert(params);
01000                 m_dynlib.SetParameter(params, name.c_str(), value.c_str());
01001             }
01008         std::string GetParam(const std::string& name)
01009             {
01010                 assert(params);
01011                 auto c_value=m_dynlib.GetParameter(params, name.c_str());
01012                 std::string ret(c_value);
01013                 m_dynlib.FreeParameterValue(c_value);
01014                 return ret;
01015             }
01016 #if CLOUDX1_HAS_CLOUDBUS
01025             void SetBus(const CBus& bus) {
01026                 assert(params);
01027                 if (m_dynlib.SetInstanceBus) {
01028                     m_dynlib.SetInstanceBus(params,bus.Get());
01029                 }
01030             }
01031 #endif // CLOUDX1_HAS_CLOUDBUS
01032
01043         void SetPresetManager(CPresetLoader *preset_manager)
01044             {
01045                 if (!preset_manager) return;
01046                 if (!m_dynlib.SetPresetManager) return;
01047                 assert(params);
01048                 m_dynlib.SetPresetManager(params,
01049                     cloudx1_custom_reader,
01050                     cloudx1_custom_reader_free,
01051                     cloudx1_custom_writer,
01052                     cloudx1_custom_exists,
01053                     cloudx1_custom_getall,
01054                     cloudx1_custom_getall_free,
01055                     cloudx1_custom_remove,
01056                     cloudx1_custom_rename,
01057                     preset_manager->IsReadOnly(),
01058                     preset_manager
01059                 );
01060             }
```

```
01061
01084         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)
01085             {
01086                 assert(!instance);
01087                 std::string l_save_path;
01088                 if (!save_path.empty()) {
01089                     #if (__cplusplus<202002L)
01090                     l_save_path = save_path.u8string();
01091                     #else // C++20
01092                     l_save_path = (const char*)save_path.u8string().c_str();
01093                     #endif // C++20
01094                 }
01095                 if (m_dynlib.InitProcess3) {
01096                     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);
01097                 } else if (frames_per_chunk!=64) {
01098                     return false;
01099                 } else {
01100                     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);
01101                 }
01102                 if (!instance) return false;
01103                 if (!init_metadata.empty()) {
01104                     for (auto&& [key,value]: init_metadata) {
01105                         m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
01106                     }
01107                     init_metadata.clear();
01108                 }
01109                 update_thread = std::thread([this, json_port](){
01110                     OnUpdateThreadStart();
01111                     m_dynlib.StartUpdateThread(instance, json_port);
01112                     OnUpdateThreadStop();
01113                 });
01114                 int timeout=5000;
01115                 #ifdef DEBUG
01116                     timeout=60*1000; // For debugger pauses
01117                 #endif
01118                 if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
01119                     return false;
01120                 }
01121                 return true;
01122             }
01123
01131         unsigned int GetChunkFrames() {
01132             if (instance && m_dynlib.GetProcessChunkFrames) {
01133                 return m_dynlib.GetProcessChunkFrames(instance);
01134             }
01135             return 64;
01136         }
01137
01145         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
      change_kind)
01146             {
01147                 if (m_dynlib.PresetManager_InformChange) {
01148                     m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
      (cloudx1_PresetChange_Kind) change_kind);
01149                 }
01150             }
01151
01163         void SetMetadata(const std::string& key, const char* value)
01164             {
01165                 if (m_dynlib.SetMetadata) {
01166                     if (instance) {
01167                         m_dynlib.SetMetadata(instance, key.c_str(), value);
01168                     } else {
01169                         init_metadata.push_back( {key, value} );
01170                     }
01171                 }
01172             }
01173
01174
01184         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01185             {
01186                 if (instance) {
01187                     if (m_dynlib.SetMetadataMulti) {
01188                         typedef const char* pchar;
01189                         pchar* keyvalue=new pchar[2*list.size()+1];
01190                         size_t n=0;
01191                         for (auto&& [key,value]: list) {
01192                             keyvalue[2*n+0]=key.c_str();
01193                             keyvalue[2*n+1]=value;
01194                             n++;
01195                         }
01196                         keyvalue[2*n+0]=nullptr;
01197
```

```
01198                         cloudx1_SetMetadataMulti(instance, keyvalue);
01199
01200                         delete[] keyvalue;
01201                     } else {
01202                         for (auto&& [key,value]: list) {
01203                             SetMetadata(key, value);
01204                         }
01205                     }
01206                 } else {
01207                     for (auto&& [key,value]: list) {
01208                         if (value) {
01209                             init_metadata.push_back( {key, value} );
01210                         }
01211                     }
01212                 }
01213             }
01214
01220         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01221             {
01222                 std::vector< std::tuple<std::string,std::string> > values;
01223                 if (m_dynlib.GetMetadataInfos) {
01224                     const char** c_values = m_dynlib.GetMetadataInfos(instance);
01225                     if (c_values) {
01226                         for (const char** c_value=c_values; *c_value; ) {
01227                             std::string key(*c_value);
01228                             c_value++;
01229                             if (*c_value) {
01230                                 std::string descr(*c_value);
01231                                 c_value++;
01232                                 values.push_back( {key,descr} );
01233                             }
01234                         }
01235                         m_dynlib.FreeMetadataInfos(instance, c_values);
01236                     }
01237                 }
01238                 return values;
01239             }
01240
01244         virtual void OnUpdateThreadStart() {}
01248         virtual void OnUpdateThreadStop() {}
01249
01250 #if CLOUDX1_HAS_WEBSERVER
01264         bool StartWebServer(int http_port, int https_port=0)
01265             {
01266                 assert(instance);
01267                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01268                 webserver = m_dynlib.Webserver(http_port,https_port,instance);
01269                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01270                     return false;
01271                 }
01272                 int web_status=m_dynlib.Webserver_Status(webserver);
01273                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01274                     return false;
01275                 }
01276                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01277                     return false;
01278                 }
01279                 return true;
01280             }
01281
01289         void StopWebServer(int timeout_ms = 1000)
01290             {
01291                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01292                     m_dynlib.Webserver_Stop(webserver, timeout_ms);
01293                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01294                 }
01295             }
01296
01300         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
01301             {
01302                 assert(instance);
01303                 if (!m_dynlib.Webserver_SetAppHealth) return;
01304                 m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
    content.c_str());
01305             }
01309         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01310             {
01311                 assert(instance);
01312                 if (!m_dynlib.Webserver_GetAppHealth) return;
01313                 char* c_contenttype=nullptr;
01314                 char* c_content=nullptr;
01315                 m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01316                 contenttype=c_contenttype;
01317                 content=c_content;
01318                 m_dynlib.Webserver_FreeString(c_contenttype);
01319                 m_dynlib.Webserver_FreeString(c_content);
```

```
01320             }
01321 #endif // CLOUDX1_HAS_WEBSERVER
01322
01331         int Terminate()
01332             {
01333                 if (!instance) return 0;
01334                 return m_dynlib.TerminateProcess(instance);
01335             }
01336
01340         void Stop()
01341             {
01342                 if (!instance) return;
01343 #if CLOUDX1_HAS_WEBSERVER
01344                 StopWebServer();
01345 #endif // CLOUDX1_HAS_WEBSERVER
01346                 if (update_thread.joinable()) {
01347                     m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01348                     update_thread.join();
01349                 }
01350                 m_dynlib.ExitProcess(instance);
01351                 instance=nullptr;
01352             }
01353
01363         unsigned int GetEstimatedDelay()
01364             {
01365                 assert(instance);
01366                 return m_dynlib.GetEstimatedDelay(instance);
01367             }
01368
01381         std::array<float,InputSampleSize>& GetBufferIn()
01382             {
01383                 assert(instance);
01384                 float *buf=m_dynlib.GetBufferIn(instance);
01385                 return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
01386
01387             }
01399         std::array<float,OutputSampleSize>& GetBufferOut()
01400             {
01401                 assert(instance);
01402                 float *buf= m_dynlib.GetBufferOut(instance);
01403                 return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
01404             }
01408         void ProcessAudio()
01409             {
01410                 assert(instance);
01411                 m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
      m_dynlib.GetBufferOut(instance));
01412             }
01424         void ProcessAudio_Planar(float const * const *input, float * const *output)
01425             {
01426                 assert(instance);
01427                 m_dynlib.ProcessAudio_Planar(instance, input, output);
01428             }
01429
01441             template<typename T>
01442         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01443             {
01444                 assert(instance);
01445                 std::vector<T> output;
01446                 unsigned int out_offset=0;
01447                 unsigned int in_offset=0;
01448                 unsigned int todo = input.size();
01449                 while (todo>0) {
01450                     unsigned int left = AddAudio(&input[in_offset], todo);
01451                     unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01452                     output.resize(out_offset + out_avail);
01453                     GetAudio(&output[out_offset], out_avail);
01454                     out_offset+=out_avail;
01455                     in_offset += todo-left;
01456                     todo=left;
01457                 }
01458                 return output;
01459             }
01460
01466         class CClient {
01467             public:
01468                 CClient(cloudx1_CInstance* instance, CDynLib& dynlib)
01469                 : m_dynlib(dynlib)
01470                     {
01471                         assert(instance);
01472                         client=m_dynlib.NewClient(instance);
01473                         if (!client) throw std::bad_alloc();
01474                     }
01475                 // non-copyable
01476                 CClient( const CInstance& ) = delete;
01477                 CClient& operator=( const CInstance& ) = delete;
01478                 ~CClient()
```

```
01479                     {
01480                         if (client) {
01481                             m_dynlib.DeleteClient(client);
01482                             client=nullptr;
01483                         }
01484                     }
01492                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01493                     {
01494                         assert(client);
01495                         int need_save=0;
01496                         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
      &need_save);
01497                         if (!canswer) return {};
01498                         std::string answer(canswer);
01499                         m_dynlib.FreeJsonAnswer (canswer);
01500                         if (NeedSave) {
01501                             *NeedSave=(need_save!=0);
01502                         }
01503                         return answer;
01504                     }
01505             private:
01506                 CDynLib& m_dynlib;
01507                 cloudx1_CClientInstance *client = nullptr;
01508         };
01516         std::shared_ptr<CClient> NewClient()
01517             {
01518                 assert(instance);
01519                 return std::make_shared<CClient>(instance, m_dynlib);
01520             }
01521
01532         bool SaveState() {
01533             assert(instance);
01534             if (m_dynlib.SaveState(instance)!=0) {
01535                 return false;
01536             }
01537             return true;
01538         }
01539 #ifdef _WIN32
01546         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01547             if (m_dynlib.SetInstanceTracing) {
01548                 m_dynlib.SetInstanceTracing(params, tracing_provider, activity_guid);
01549             }
01550         }
01557         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01558             assert(instance);
01559             if (m_dynlib.SetInstanceTracingProcessActivity) {
01560                 m_dynlib.SetInstanceTracingProcessActivity(instance, activity_guid);
01561             }
01562         }
01563 #endif // _WIN32
01564     protected:
01565             template<typename T>
01566         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01567             {
01568                 assert(instance);
01569                 return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01570             }
01571             template<typename T>
01572         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01573             {
01574                 assert(instance);
01575                 return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01576             }
01577
01578     private:
01579         CDynLib& m_dynlib;
01580         cloudx1_CParameters* params = nullptr;
01581         cloudx1_CInstance* instance = nullptr;
01582         std::thread update_thread;
01583         std::vector< std::pair<std::string,std::string> > init_metadata;
01584 #if CLOUDX1_HAS_WEBSERVER
01585         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01586 #endif // CLOUDX1_HAS_WEBSERVER
01587     };
01588
01595     class CAudioConverter {
01596         public:
01602             CAudioConverter(CDynLib& dynlib)
01603                 : m_dynlib(dynlib)
01604                 { }
01605
01609             virtual ~CAudioConverter()
01610                 {
01611                     Delete();
01612                 }
```

```
01613
01629             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
      cloudx1_ConverterTargetMode target_mode)
01630                 {
01631                     Delete();
01632                     if (m_dynlib.GetAudioConverter)
01633                         m_cvt = m_dynlib.GetAudioConverter((cloudx1_SampleFormat)fmt, nFrame, payl_ch,
      payl_ch_offset, target_mode);
01634                     return IsOK();
01635                 }
01636             void Delete()
01637                 {
01638                     if (m_cvt && m_dynlib.FreeAudioConverter) {
01639                         m_dynlib.FreeAudioConverter(m_cvt);
01640                     }
01641                     m_cvt = nullptr;
01642                 }
01643             bool IsOK() const { return (m_cvt!=nullptr); }
01644
01653             void ConvertFrom(const uint8_t *payload, float *output)
01654                 { m_dynlib.AudioConverter_From(m_cvt, payload, output); }
01655
01664             void ConvertTo(const float *input, uint8_t *payload)
01665                 { m_dynlib.AudioConverter_To(m_cvt, input, payload); }
01666
01667         private:
01668             CDynLib& m_dynlib;
01669             cloudx1_CAudioConverter *m_cvt = nullptr;
01670     };
01671
01672 }; // namespace dyn
01673 }; // namespace cloudx1
01674 }; // namespace sound4
01675
01676 // bridge C callbacks to CPresetLoader
01677 extern "C" {
01678     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
01679         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01680         std::string content=preset_loader->Read(filename);
01681         return strdup(content.c_str());
01682     }
01683     static void cloudx1_custom_reader_free(char *content, void* userdata) {
01684         if (!content) return;
01685         free(content);
01686     }
01687     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01688         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01689         auto res=preset_loader->Write(filename,content);
01690         return res?0:-1;
01691     }
01692     static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
01693         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01694         auto res=preset_loader->Exists(filename);
01695         return res?0:-1;
01696     }
01697     static fs_char** cloudx1_custom_getall(void* userdata) {
01698         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01699         auto res=preset_loader->GetAll();
01700         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01701         for (size_t n=0;n<res.size();n++) {
01702             all[n]=fs_strdup(res[n].c_str());
01703         }
01704         all[res.size()]=nullptr;
01705         return all;
01706     }
01707     static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
01708         if (!all) return;
01709         for (fs_char** one=all;*one;one++) {
01710             free(*one);
01711         }
01712         free(all);
01713     }
01714     static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
01715         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01716         auto res=preset_loader->Remove(filename);
01717         return res?0:-1;
01718     }
01719     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01720         sound4::cloudx1::dyn::CPresetLoader
      *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
01721         auto res=preset_loader->Rename(from,to);
01722         return res?0:-1;
```

```
01723     }
01724
01725 };
```

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
- class sound4::CAudioConverter

  Audio converter helper.

Namespaces

- namespace sound4
- namespace sound4::helper

Macros

- #define UNIXLIKE

Typedefs

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

Enumerations

- enum sound4::SampleFormat {
  sound4::INVALID_FORMAT , sound4::S16_LE , sound4::S16_BE , sound4::S24_LE ,
  sound4::S24_BE , sound4::S32_LE , sound4::S32_BE , sound4::F32_LE ,
  sound4::F32_BE , sound4::S16_NATIVE , sound4::S24_NATIVE , sound4::S32_NATIVE ,
  sound4::F32_NATIVE }
- enum sound4::LogSeverity {
  sound4::none , sound4::fatal , sound4::error , sound4::warning ,
  sound4::info , sound4::verbose , sound4::verbose2 , sound4::verbose3 ,
  sound4::verbose4 , sound4::verbose5 }
- enum sound4::PresetChange_Kind { sound4::change_kind_created , sound4::change_kind_modified ,
  sound4::change_kind_deleted }
- enum sound4::ConverterTargetMode {
  sound4::ConverterTargetMode_AsInput = sound4cl_ConverterTargetMode_AsInput , sound4::ConverterTargetMode_Mono
  = sound4cl_ConverterTargetMode_Mono , sound4::ConverterTargetMode_Stereo = sound4cl_Converter↩
  TargetMode_Stereo , sound4::ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_↩
  StereoToMono ,
  sound4::ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo }

Functions

- static std::string sound4::helper::WStringToUTF8 (const std::wstring &wstr)
- static std::wstring sound4::helper::UTF8ToWString (const std::string &str)
- static void sound4::_log_cb_c (sound4cl_LogSeverity severity, const char *c_msg)

Variables

- static log_cb_t sound4::_log_cb

15.25.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file sound4cl.hpp.

15.25.2 Macro Definition Documentation

15.25.2.1 UNIXLIKE

```
#define UNIXLIKE
```

Definition at line 24 of file sound4cl.hpp.

15.26 sound4cl.hpp

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include <string>
00011 #include <exception>
00012 #include <thread>
00013 #include <functional>
00014 #include <memory>
00015 #include <array>
00016 #include <assert.h>
00017 #include <stdexcept>
00018 #include <filesystem>
00019 #include <string.h>
00020
00021 #if defined(__unix__) || defined(__APPLE__)
00022     #define UNIXLIKE 1
00023 #else
00024     #define UNIXLIKE 0
00025 #endif
00026
00027 #if UNIXLIKE
00028     #include <dlfcn.h>
00029 #elif defined(_WIN32)
00030     #include <libloaderapi.h>
00031     #ifdef _MSC_VER
00032         #pragma warning(disable : 4996)
00033     #endif
00034 #else
00035     #error "Unsupported OS"
00036 #endif
00037
00038 #include "sound4cl_cdef.h"
00039
00040     // bridge C callbacks to CPresetLoader
00041     extern "C" {
00042         static char *sound4cl_custom_reader(const fs_char *filename, void* userdata);
00043         static void sound4cl_custom_reader_free(char *content, void* userdata);
00044         static int sound4cl_custom_writer(const fs_char *filename, const char *content, void*
    userdata);
00045         static int sound4cl_custom_exists(const fs_char *filename, void* userdata);
00046         static fs_char** sound4cl_custom_getall(void* userdata);
00047         static void sound4cl_custom_getall_free(fs_char** all, void* userdata);
00048         static int sound4cl_custom_remove(const fs_char *filename, void* userdata);
00049         static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00050     };
00051
00055 namespace sound4 {
00056
00060     enum SampleFormat {
00061         INVALID_FORMAT=sound4cl_SampleFormat::INVALID_FORMAT,
00062         S16_LE        =sound4cl_SampleFormat::S16_LE,
00063         S16_BE        =sound4cl_SampleFormat::S16_BE,
00064         S24_LE        =sound4cl_SampleFormat::S24_LE,
00065         S24_BE        =sound4cl_SampleFormat::S24_BE,
00066         S32_LE        =sound4cl_SampleFormat::S32_LE,
00067         S32_BE        =sound4cl_SampleFormat::S32_BE,
00068         F32_LE        =sound4cl_SampleFormat::F32_LE,
00069         F32_BE        =sound4cl_SampleFormat::F32_BE,
00070         S16_NATIVE    =sound4cl_SampleFormat::S16_NATIVE,
00071         S24_NATIVE    =sound4cl_SampleFormat::S24_NATIVE,
00072         S32_NATIVE    =sound4cl_SampleFormat::S32_NATIVE,
00073         F32_NATIVE    =sound4cl_SampleFormat::F32_NATIVE,
00074     };
00075
00079     namespace helper {
00080         #ifdef _WIN32
00086         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00087             if (wstr.empty()) return {};
00088             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
    NULL);
00089             std::string str(size, 0);
00090             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00091             return str;
00092         }
00098         static inline std::wstring UTF8ToWString(const std::string& str) {
00099             if (str.empty()) return std::wstring();
00100             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00101             std::wstring wstr(size, 0);
00102             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00103             return wstr;
00104         }
00105         #endif // _WIN32
```

```
00106
00112          template<typename T>
00113      struct SampleFormat {
00114      };
00115          template<>
00116      struct SampleFormat<int16_t> {
00117          const sound4::SampleFormat format = S16_NATIVE;
00118      };
00119          template<>
00120      struct SampleFormat<int32_t> {
00121          const sound4::SampleFormat format = S32_NATIVE;
00122      };
00123          template<>
00124      struct SampleFormat<float> {
00125          const sound4::SampleFormat format = F32_NATIVE;
00126      };
00127
00133          template <typename T>
00134      class DynFuncHolder {
00135          // Storage definition, depending on OS
00136          #if UNIXLIKE
00137              using DynFunc_t = void *;
00138          #else
00139              using DynFunc_t = FARPROC;
00140          #endif
00141      private:
00142          DynFunc_t m_ptr = NULL;
00143      public:
00144          DynFuncHolder() = default;
00145          DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00146          operator bool() const { return (m_ptr!=NULL); }
00147          bool IsOk() const { return m_ptr!=NULL; }
00148          operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00149      };
00150
00155      class CDynLoader {
00156          // Storage definition, depending on OS
00157          #if UNIXLIKE
00158              using DynLib_t = void *;
00159          #else
00160              using DynLib_t = HMODULE;
00161          #endif
00162          private:
00163              DynLib_t m_lib {};
00164          public:
00165              CDynLoader() = default;
00166              CDynLoader(CDynLoader&& other) noexcept : m_lib(other.m_lib) { other.m_lib = NULL; }
00167              CDynLoader& operator=(CDynLoader&& other) noexcept {
00168                  if (this != &other) { Close(); m_lib = other.m_lib; other.m_lib = NULL; }
00169                  return *this;
00170              }
00171              // No copy allowed
00172              CDynLoader(const CDynLoader&) = delete;
00173              CDynLoader& operator=(CDynLoader const&) = delete;
00174              operator bool() const {
00175                  return (m_lib!=NULL);
00176              }
00177              bool IsOk() const {
00178                  return (m_lib!=NULL);
00179              }
00180              #if UNIXLIKE
00181                  void Close() {
00182                      if (IsOk()) {
00183                          #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00184                          dlclose(m_lib);
00185                          #endif // __SANITIZE_ADDRESS__
00186                          m_lib=NULL;
00187                      }
00188                  }
00189                  bool Load(const std::filesystem::path& dynlib)
00190                  {
00191                      Close();
00192                      m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00193                      return IsOk();
00194                  }
00195                      template <typename T>
00196                  DynFuncHolder<T> GetSymbol(const std::string& name) {
00197                      auto ptr=dlsym(m_lib, name.c_str());
00198                      if (!ptr) {
00199                          throw std::runtime_error("Missing function in library");
00200                      }
00201                      return DynFuncHolder<T>(ptr);
00202                  }
00203                      template <typename T>
00204                  DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00205                      auto ptr=dlsym(m_lib, name.c_str());
00206                      return DynFuncHolder<T>(ptr);
```

```
00207                     }
00208                     static std::filesystem::path GetThisLibraryPath(void) {
00209                         Dl_info info;   // local: static would be a data race across concurrent
      callers
00210                         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
00211                         auto rp=realpath(info.dli_fname, NULL);
00212                         if (!rp) {
00213                             #if defined(__unix__)
00214                                 rp=realpath("/proc/self/exe", NULL);
00215                             #elif defined(__APPLE__)
00216                                 // How to solve this for Apple ?
00217                             #endif
00218                         }
00219                         if (!rp) return {};
00220                         std::filesystem::path p(rp);
00221                         free(rp);
00222                         return p;
00223                     }
00224                 #else
00225                     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
00226                     {
00227                         Close();
00228                         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00229                         return IsOk();
00230                     }
00231                     void Close() {
00232                         if (IsOk()) {
00233                             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
      FreeLibraryAndExitThread but...
00234                             // So define SOUND4_CALL_FREELIBRARY before including this file if you
      really want to unload the library dynamically
00235                             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
00236                                 FreeLibraryAndExitThread(m_lib, 0);
00237                             #elif defined(SOUND4_CALL_FREELIBRARY)
00238                                 FreeLibrary(m_lib);
00239                             #endif // SOUND4_CALL_FREELIBRARY
00240                             m_lib=NULL;
00241                         }
00242                     }
00243                         template <typename T>
00244                     DynFuncHolder<T> GetSymbol(const std::string& name) {
00245                         auto ptr=GetProcAddress(m_lib, name.c_str());
00246                         if (!ptr) {
00247                             throw std::runtime_error("Missing function in library");
00248                         }
00249                         return DynFuncHolder<T>(ptr);
00250                     }
00251                         template <typename T>
00252                     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00253                         auto ptr=GetProcAddress(m_lib, name.c_str());
00254                         return DynFuncHolder<T>(ptr);
00255                     }
00256                     static std::filesystem::path GetThisLibraryPath(void) {
00257                         static wchar_t path[MAX_PATH]={};
00258                         HMODULE hm = NULL;
00259                         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
00260                                 GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00261                                 (LPCWSTR) &GetThisLibraryPath, &hm))
00262                         {
00263                             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00264                                 return std::filesystem::path(path);
00265                             }
00266                         }
00267                         return {};
00268                     }
00269                 #endif
00270                 virtual ~CDynLoader() {
00271                     Close();
00272                 }
00273         }; // class CDynLoader
00274     }; // namespace helper
00275
00276
00280     enum LogSeverity {
00281        none     = sound4cl_none,
00282        fatal    = sound4cl_fatal,
00283        error    = sound4cl_error,
00284        warning  = sound4cl_warning,
00285        info     = sound4cl_info,
00286        verbose  = sound4cl_verbose,
00287        verbose2 = sound4cl_verbose2,
00288        verbose3 = sound4cl_verbose3,
00289        verbose4 = sound4cl_verbose4,
00290        verbose5 = sound4cl_verbose5,
00291     };
00292
```

```
00296     using log_cb_t = std::function<void(LogSeverity,const std::string&)>;
00297
00301     static log_cb_t _log_cb;
00302     extern "C" {
00303         static inline void _log_cb_c(sound4cl_LogSeverity severity, const char *c_msg) {
00304             _log_cb((LogSeverity)severity, std::string(c_msg));
00305         }
00306     }
00307
00314     class CProcessor {
00315     private:
00316         helper::CDynLoader m_lib;
00317         bool m_bOK = false;
00318         std::string m_prefix; // The prefix of functions, depends on the library (cloud_, impact_...)
00319     protected:
00320         friend class CBus;
00321         friend class CInstance;
00322         friend class CAudioConverter;
00323
00324         // All those wrapped functions have the same signature as there C sound4cl_XXX
00325         helper::DynFuncHolder< decltype(sound4cl_GetVersion             ) > fnGetVersion             ;
00326         helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames   ) > fnGetChunkSizeInFrames   ;
00327         helper::DynFuncHolder< decltype(sound4cl_GetChannelCount        ) > fnGetChannelCount        ;
00328         helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount     ) > fnGetAudioInputCount     ;
00329         helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount    ) > fnGetAudioOutputCount    ;
00330         helper::DynFuncHolder< decltype(sound4cl_GetSampleRate          ) > fnGetSampleRate          ;
00331         helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback      ) > fnSetLoggerCallback      ;
00332         helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity         ) > fnSetLogSeverity         ;
00333         helper::DynFuncHolder< decltype(sound4cl_NewParameters          ) > fnNewParameters          ;
00334         helper::DynFuncHolder< decltype(sound4cl_FreeParameters         ) > fnFreeParameters         ;
00335         helper::DynFuncHolder< decltype(sound4cl_SetParameter           ) > fnSetParameter           ;
00336         helper::DynFuncHolder< decltype(sound4cl_GetParameter           ) > fnGetParameter           ;
00337         helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue     ) > fnFreeParameterValue     ;
00338         helper::DynFuncHolder< decltype(sound4cl_InitProcess            ) > fnInitProcess            ;
00339         helper::DynFuncHolder< decltype(sound4cl_InitProcess2           ) > fnInitProcess2           ;
00340         helper::DynFuncHolder< decltype(sound4cl_TerminateProcess       ) > fnTerminateProcess       ;
00341         helper::DynFuncHolder< decltype(sound4cl_ExitProcess            ) > fnExitProcess            ;
00342         helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread      ) > fnStartUpdateThread      ;
00343         helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread       ) > fnStopUpdateThread       ;
00344         helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady  ) > fnWaitUpdateThreadReady  ;
00345         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio           ) > fnProcessAudio           ;
00346         helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar    ) > fnProcessAudio_Planar    ;
00347         helper::DynFuncHolder< decltype(sound4cl_GetBufferIn            ) > fnGetBufferIn            ;
00348         helper::DynFuncHolder< decltype(sound4cl_GetBufferOut           ) > fnGetBufferOut           ;
00349         helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay      ) > fnGetEstimatedDelay      ;
00350         helper::DynFuncHolder< decltype(sound4cl_GetFormatName          ) > fnGetFormatName          ;
00351         helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName      ) > fnGetFormatFromName      ;
00352         helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat     ) > fnGetBytesFromFormat     ;
00353         helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame      ) > fnGetMaxPacketFrame      ;
00354         helper::DynFuncHolder< decltype(sound4cl_AddAudio               ) > fnAddAudio               ;
00355         helper::DynFuncHolder< decltype(sound4cl_AddPadAudio            ) > fnAddPadAudio            ;
00356         helper::DynFuncHolder< decltype(sound4cl_GetOutputCount         ) > fnGetOutputCount         ;
00357         helper::DynFuncHolder< decltype(sound4cl_GetAudio               ) > fnGetAudio               ;
00358         helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom       ) > fnAudioConvertFrom       ;
00359         helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo         ) > fnAudioConvertTo         ;
00360         helper::DynFuncHolder< decltype(sound4cl_StereoToMono           ) > fnStereoToMono           ;
00361         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo           ) > fnMonoToStereo           ;
00362         helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
00363         helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo  ) > fnAudioMonoToLiveStereo  ;
00364         helper::DynFuncHolder< decltype(sound4cl_NewClient              ) > fnNewClient              ;
00365         helper::DynFuncHolder< decltype(sound4cl_DeleteClient           ) > fnDeleteClient           ;
00366         helper::DynFuncHolder< decltype(sound4cl_ProcessJson            ) > fnProcessJson            ;
00367         helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer         ) > fnFreeJsonAnswer         ;
00368         helper::DynFuncHolder< decltype(sound4cl_SaveState              ) > fnSaveState              ;
00369
00370         // only for processes supporting webserver
00371         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp          ) > fnWebserver_tcp          ;
00372         helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2         ) > fnWebserver_tcp2         ;
00373         helper::DynFuncHolder< decltype(sound4cl_Webserver              ) > fnWebserver              ;
00374         helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop         ) > fnWebserver_Stop         ;
00375         helper::DynFuncHolder< decltype(sound4cl_Webserver_Status       ) > fnWebserver_Status       ;
00376
00377         // added 2023-03-22
00378         helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar    ) > fnStereoToMono_Planar    ;
00379         helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar    ) > fnMonoToStereo_Planar    ;
00380
00381         // added 2023-05-22
00382         helper::DynFuncHolder< decltype(sound4cl_SetMetadata            ) > fnSetMetadata            ;
00383         helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos       ) > fnGetMetadataInfos       ;
00384         helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos      ) > fnFreeMetadataInfos      ;
00385
00386         // added 2023-06-20
00387         helper::DynFuncHolder< decltype(sound4cl_SetPresetManager               ) >
      fnSetPresetManager      ;
00388         helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange      ) >
      fnPresetManager_InformChange      ;
00389
```

```
00390         // added 2023-09-21
00391         helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) >
      fnGetPossibleChunkSizeInFrames   ;
00392         helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) >
      fnGetProcessChunkFrames   ;
00393         helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3
      ;
00394
00395         // added 2023-06-26, only for processes supporting bus
00396         helper::DynFuncHolder< decltype(sound4cl_NewBus                  ) > fnNewBus      ;
00397         helper::DynFuncHolder< decltype(sound4cl_FreeBus                 ) > fnFreeBus      ;
00398         helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus          ) > fnSetInstanceBus      ;
00399
00400         // added 2024-02-19
00401         helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti        ) > fnSetMetadataMulti
      ;
00402
00403         // added 2024-05-23
00404         helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) >
      fnWebserver_SetAppHealth      ;
00405         helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) >
      fnWebserver_GetAppHealth      ;
00406         helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString
      ;
00407
00408 #ifdef _WIN32
00409         // added 2025-05-28
00410         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracing                ) >
      fnSetInstanceTracing                ;
00411         helper::DynFuncHolder< decltype(sound4cl_SetInstanceTracingProcessActivity ) >
      fnSetInstanceTracingProcessActivity ;
00412 #endif // _WIN32
00413
00414         // added 2026-02-17
00415         helper::DynFuncHolder< decltype(sound4cl_GetAudioConverter      ) > fnGetAudioConverter      ;
00416         helper::DynFuncHolder< decltype(sound4cl_FreeAudioConverter     ) > fnFreeAudioConverter     ;
00417         helper::DynFuncHolder< decltype(sound4cl_AudioConverter_From    ) > fnAudioConverter_From    ;
00418         helper::DynFuncHolder< decltype(sound4cl_AudioConverter_To      ) > fnAudioConverter_To      ;
00419
00420             template <typename T>
00421         helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
00422             std::string prefix_name = m_prefix+name;
00423             return m_lib.GetSymbol<T>(prefix_name);
00424         }
00425
00426     public:
00427         CProcessor() = default;
00428
00436         bool Load(const std::filesystem::path& filepath) {
00437             m_lib.Close();
00438             // If a path is given, use it directly and do not try other path
00439             if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
00440                 return false;
00441             } else if (!m_lib.IsOk()) {
00442                 auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
00443                 auto filename = filepath.filename();
00444                 #ifndef _WIN32
00445                 // Linux: if in a bin directory, try ../lib first
00446                 if (!m_lib.IsOk() && thisdir.filename()=="bin") {
00447                     auto libdir = thisdir.parent_path() / "lib";
00448                     m_lib.Load(libdir / filename);
00449                 }
00450                 #endif // !_WIN32
00451                 if (!m_lib.IsOk()) {
00452                     // Search in the same directory this code is in
00453                     m_lib.Load(thisdir / filename);
00454                 }
00455                 if (!m_lib.IsOk()) {
00456                     // Try current path
00457                     std::error_code ec;
00458                     auto p = std::filesystem::current_path(ec);
00459                     if (!ec) {
00460                         m_lib.Load(p / filename);
00461                     }
00462                 }
00463                 if (!m_lib.IsOk()) {
00464                     return false;
00465                 }
00466             }
00467             // Get information from the library itself
00468             try {
00469                 auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             )
      >("SOUND4_GetProcessInfo"   );
00470                 auto info = GetInfo();
00471                 m_prefix = info->prefix;
00472             } catch (std::runtime_error& ) {
00473                 // Compatibility: search in known libraries without this function
```

```
00474                 #ifdef _WIN32
00475                     std::string filename = helper::WStringToUTF8(filepath.stem());
00476                 #else // !_WIN32
00477                     std::string filename = filepath.stem();
00478                     // remove lib prefix
00479                     filename=filename.substr(3);
00480                 #endif // ! _WIN32
00481                 if (filename=="sound4.x1.cloud") {
00482                     m_prefix="cloudx1_";
00483                 } else if (filename=="sound4.impact.cl") {
00484                     m_prefix="impact_";
00485                 } else if (filename=="sound4.bigvoice.cl") {
00486                     m_prefix="bigvoice_";
00487                 } else {
00488                     return false;
00489                 }
00490             }
00491 
00492             // Load all C functions
00493             try {
00494                 fnGetVersion              = GetPrefixSymbol< decltype(sound4cl_GetVersion
      ) >("GetVersion"              );
00495                 fnGetChunkSizeInFrames    = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
      ) >("GetChunkSizeInFrames"    );
00496                 fnGetChannelCount         = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
      ) >("GetChannelCount"         );
00497                 fnGetAudioInputCount      = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
      ) >("GetAudioInputCount"      );
00498                 fnGetAudioOutputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
      ) >("GetAudioOutputCount"     );
00499                 fnGetSampleRate           = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
      ) >("GetSampleRate"           );
00500                 fnSetLoggerCallback       = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
      ) >("SetLoggerCallback"       );
00501                 fnSetLogSeverity          = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
      ) >("SetLogSeverity"          );
00502                 fnNewParameters           = GetPrefixSymbol< decltype(sound4cl_NewParameters
      ) >("NewParameters"           );
00503                 fnFreeParameters          = GetPrefixSymbol< decltype(sound4cl_FreeParameters
      ) >("FreeParameters"          );
00504                 fnSetParameter            = GetPrefixSymbol< decltype(sound4cl_SetParameter
      ) >("SetParameter"            );
00505                 fnGetParameter            = GetPrefixSymbol< decltype(sound4cl_GetParameter
      ) >("GetParameter"            );
00506                 fnFreeParameterValue      = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
      ) >("FreeParameterValue"      );
00507                 fnInitProcess             = GetPrefixSymbol< decltype(sound4cl_InitProcess
      ) >("InitProcess"             );
00508                 fnInitProcess2            = GetPrefixSymbol< decltype(sound4cl_InitProcess2
      ) >("InitProcess2"            );
00509                 fnTerminateProcess        = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
      ) >("TerminateProcess"        );
00510                 fnExitProcess             = GetPrefixSymbol< decltype(sound4cl_ExitProcess
      ) >("ExitProcess"             );
00511                 fnStartUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
      ) >("StartUpdateThread"       );
00512                 fnStopUpdateThread        = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
      ) >("StopUpdateThread"        );
00513                 fnWaitUpdateThreadReady   = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
      ) >("WaitUpdateThreadReady"   );
00514                 fnProcessAudio            = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
      ) >("ProcessAudio"            );
00515                 fnProcessAudio_Planar     = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
      ) >("ProcessAudio_Planar"     );
00516                 fnGetBufferIn             = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
      ) >("GetBufferIn"             );
00517                 fnGetBufferOut            = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
      ) >("GetBufferOut"            );
00518                 fnGetEstimatedDelay       = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
      ) >("GetEstimatedDelay"       );
00519                 fnGetFormatName           = GetPrefixSymbol< decltype(sound4cl_GetFormatName
      ) >("GetFormatName"           );
00520                 fnGetFormatFromName       = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
      ) >("GetFormatFromName"       );
00521                 fnGetBytesFromFormat      = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
      ) >("GetBytesFromFormat"      );
00522                 fnGetMaxPacketFrame       = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
      ) >("GetMaxPacketFrame"       );
00523                 fnAddAudio                = GetPrefixSymbol< decltype(sound4cl_AddAudio
      ) >("AddAudio"                );
00524                 fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
      ) >("AddPadAudio"             );
00525                 fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
      ) >("GetOutputCount"          );
00526                 fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
      ) >("GetAudio"                );
00527                 fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
```

```
    ) >("AudioConvertFrom"       );
00528                fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
    ) >("AudioConvertTo"         );
00529                fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
    ) >("StereoToMono"           );
00530                fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo
    ) >("MonoToStereo"           );
00531                fnAudioMonoFromLiveStereo = GetPrefixSymbol<
    decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
00532                fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo
    ) >("AudioMonoToLiveStereo"  );
00533                fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient
    ) >("NewClient"              );
00534                fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient
    ) >("DeleteClient"           );
00535                fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson
    ) >("ProcessJson"            );
00536                fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer
    ) >("FreeJsonAnswer"         );
00537                fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState
    ) >("SaveState"              );
00538            } catch (std::runtime_error& ) {
00539                return false;
00540            }
00541            // Webserver : not for all projects
00542            try {
00543                fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp
    ) >("Webserver_tcp"          );
00544                fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2
    ) >("Webserver_tcp2"         );
00545                fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver
    ) >("Webserver"              );
00546                fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop
    ) >("Webserver_Stop"         );
00547                fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status
    ) >("Webserver_Status"       );
00548            } catch (std::runtime_error& ) {
00549                // Ignored, handler will take care
00550            }
00551            // Stereo/Mono conversion : must be present
00552            try {
00553                // added 2023-03-22
00554                fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar
    ) >("StereoToMono_Planar"    );
00555                fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar
    ) >("MonoToStereo_Planar"    );
00556
00557            } catch (std::runtime_error& ) {
00558                return false;
00559            }
00560            // C functions allowed to be missing
00561            try {
00562                // added 2023-05-22
00563                fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata
    ) >("SetMetadata"            );
00564                fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos
    ) >("GetMetadataInfos"       );
00565                fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos
    ) >("FreeMetadataInfos"      );
00566                // added 2023-06-20
00567                fnSetPresetManager            = GetPrefixSymbol< decltype(sound4cl_SetPresetManager
    ) >("SetPresetManager"           );
00568                fnPresetManager_InformChange  = GetPrefixSymbol<
    decltype(sound4cl_PresetManager_InformChange     ) >("PresetManager_InformChange"    );
00569
00570                // added 2023-09-21
00571                fnGetPossibleChunkSizeInFrames = GetPrefixSymbol<
    decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
00572                fnGetProcessChunkFrames        = GetPrefixSymbol<
    decltype(sound4cl_GetProcessChunkFrames         ) >("GetProcessChunkFrames"         );
00573                fnInitProcess3                 = GetPrefixSymbol< decltype(sound4cl_InitProcess3
    ) >("InitProcess3"               );
00574            }  catch (std::runtime_error& ) {
00575                // Ignored, handler will take care
00576            }
00577
00578            // bus: not for all projects
00579            try {
00580                // added 2023-06-26
00581                fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus
    ) >("NewBus"                 );
00582                fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus
    ) >("FreeBus"                );
00583                fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus
    ) >("SetInstanceBus"         );
00584            }  catch (std::runtime_error& ) {
00585                // Ignored, handler will take care
```

```
00586             }
00587
00588             // Other optional
00589             try {
00590                 // added 2024-02-19
00591                 fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti
      ) >("SetMetadataMulti"       );
00592
00593             }  catch (std::runtime_error& ) {
00594                 // Ignored, handler will take care
00595             }
00596
00597             // Other optional, web server only
00598             try {
00599                 // added 2024-05-23
00600                 fnWebserver_SetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
00601                 fnWebserver_GetAppHealth    = GetPrefixSymbol<
      decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
00602                 fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString
      ) >("Webserver_FreeString"       );
00603             }  catch (std::runtime_error& ) {
00604                 // Ignored, handler will take care
00605             }
00606
00607 #ifdef _WIN32
00608             // Other optional, windows tracing
00609             try {
00610                 // added 2025-05-28
00611                 fnSetInstanceTracing               = GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracing                    ) >("SetInstanceTracing"               );
00612                 fnSetInstanceTracingProcessActivity= GetPrefixSymbol<
      decltype(sound4cl_SetInstanceTracingProcessActivity    ) >("SetInstanceTracingProcessActivity");
00613             }  catch (std::runtime_error& ) {
00614                 // Ignored, handler will take care
00615             }
00616 #endif // _WIN32
00617
00618             // Other optional
00619             try {
00620                 // added 2026-02-17
00621                 fnGetAudioConverter       = GetPrefixSymbol< decltype(sound4cl_GetAudioConverter
      ) >("GetAudioConverter"      );
00622                 fnFreeAudioConverter      = GetPrefixSymbol< decltype(sound4cl_FreeAudioConverter
      ) >("FreeAudioConverter"     );
00623                 fnAudioConverter_From     = GetPrefixSymbol< decltype(sound4cl_AudioConverter_From
      ) >("AudioConverter_From"    );
00624                 fnAudioConverter_To       = GetPrefixSymbol< decltype(sound4cl_AudioConverter_To
      ) >("AudioConverter_To"      );
00625             }  catch (std::runtime_error& ) {
00626                 // Ignored, handler will take care
00627             }
00628
00629             m_bOK=true;
00630             return true;
00631         }
00638         bool IsOk() const {
00639             return m_bOK;
00640         }
00641
00642         // ------------------------------------------------------------
00649         // ------------------------------------------------------------
00660         void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, SampleFormat fmt)
00661             { fnAudioConvertFrom(payload, output, nSpl, (sound4cl_SampleFormat)fmt); }
00662
00673         void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, SampleFormat fmt)
00674             { fnAudioConvertTo(input, payload, nSpl, (sound4cl_SampleFormat)fmt); }
00675
00685         void StereoToMono(const float *input, float *output, size_t nFrame)
00686             { fnStereoToMono(input, output, nFrame); }
00687
00697         void MonoToStereo(const float *input, float *output, size_t nFrame)
00698             { fnMonoToStereo(input, output, nFrame); }
00699
00710         void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame)
00711             { fnStereoToMono_Planar(inputL, inputR, output, nFrame); }
00712
00723         void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame)
00724             { fnMonoToStereo_Planar(input, outputL, outputR, nFrame); }
00725
00732         void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
00733             { fnAudioMonoFromLiveStereo(payload, output); }
00734
00741         void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
00742             { fnAudioMonoToLiveStereo(input, payload); }
00743
```

```
00751         std::string GetVersion()
00752             { return std::string(fnGetVersion()); }
00753
00764         unsigned int GetChunkSizeInFrames()
00765             { return fnGetChunkSizeInFrames(); }
00766
00775         std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00776             {
00777                 std::vector<unsigned int> list;
00778                 if (fnGetPossibleChunkSizeInFrames) {
00779                     for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
00780                         list.push_back(*src);
00781                     }
00782                 } else {
00783                     list.push_back(fnGetChunkSizeInFrames());
00784                 }
00785                 return list;
00786             }
00787
00793         unsigned int GetChannelCount()
00794             { return fnGetChannelCount(); }
00795
00803         unsigned int GetAudioInputCount()
00804             { return fnGetAudioInputCount(); }
00805
00813         unsigned int GetAudioOutputCount()
00814             { return fnGetAudioOutputCount(); }
00815
00823         unsigned int GetSampleRate()
00824             { return fnGetSampleRate(); }
00825
00832         std::string GetFormatName( const SampleFormat fmt)
00833             { return std::string(fnGetFormatName((sound4cl_SampleFormat)fmt)); }
00834
00841         SampleFormat GetFormatFromName( const std::string& name)
00842             { return SampleFormat(fnGetFormatFromName(name.c_str())); }
00843
00850         unsigned int GetBytesFromFormat( const SampleFormat fmt)
00851             { return fnGetBytesFromFormat((sound4cl_SampleFormat)fmt); }
00852
00856         void SetLogSeverity(LogSeverity severity)
00857             { fnSetLogSeverity((sound4cl_LogSeverity)severity); }
00858
00862         void SetLoggerCallback(log_cb_t cb)
00863             { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
00864     };
00865
00871     class CBus {
00872         public:
00873             CBus(CProcessor& dynlib)
00874                 : m_dynlib(dynlib)
00875                 {
00876                     if (m_dynlib.fnNewBus) {
00877                         m_bus=m_dynlib.fnNewBus();
00878                     }
00879                 }
00880             ~CBus()
00881                 {
00882                     if (m_bus && m_dynlib.fnFreeBus) {
00883                         m_dynlib.fnFreeBus(m_bus);
00884                     }
00885                     m_bus = nullptr;
00886                 }
00887         protected:
00888             friend class CInstance;
00889             sound4cl_CBus *Get() const { return m_bus; }
00890         private:
00891             CProcessor& m_dynlib;
00892             sound4cl_CBus *m_bus;
00893     };
00894
00900     class CPresetLoader {
00901     public:
00902         CPresetLoader() = default;
00903         virtual ~CPresetLoader() = default;
00904
00905         virtual bool IsReadOnly() = 0;
00906         virtual bool Exists(const std::filesystem::path &name) = 0;
00907         virtual bool Remove(const std::filesystem::path &name) = 0;
00908         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00909         virtual std::vector<std::filesystem::path> GetAll() = 0;
00910         virtual std::string Read(const std::filesystem::path &filename) = 0;
00911         virtual bool Write(const std::filesystem::path &filename, const std::string &content) =0;
00912     };
00913     enum PresetChange_Kind {
00914         change_kind_created  = sound4cl_PresetChange_Kind::change_kind_created,
00915         change_kind_modified = sound4cl_PresetChange_Kind::change_kind_modified,
```

```
00916         change_kind_deleted  = sound4cl_PresetChange_Kind::change_kind_deleted,
00917     };
00923     class CInstance {
00924     public:
00930         CInstance(CProcessor& dynlib)
00931         : m_dynlib(dynlib)
00932             {
00933                 params = m_dynlib.fnNewParameters();
00934                 if (!params) throw std::bad_alloc();
00935             }
00936
00945         CInstance(CProcessor& dynlib, void* _instance)
00946         : m_dynlib(dynlib)
00947         , instance(reinterpret_cast<sound4cl_CInstance*>(_instance))
00948         , m_bOwned(false)
00949             {
00950             }
00951         // non-copyable
00952         CInstance( const CInstance& ) = delete;
00953         CInstance& operator=( const CInstance& ) = delete;
00957         virtual ~CInstance()
00958             {
00959                 Stop();
00960                 if (params) m_dynlib.fnFreeParameters(params);
00961             }
00965        bool IsOk() const { return instance!=nullptr; }
00975         void SetParam(const std::string& name, const std::string& value)
00976             {
00977                 assert(params);
00978                 m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
00979             }
00986         std::string GetParam(const std::string& name)
00987             {
00988                 assert(params);
00989                 auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
00990                 std::string ret(c_value);
00991                 m_dynlib.fnFreeParameterValue(c_value);
00992                 return ret;
00993             }
00994
01003         void SetBus(const CBus& bus) {
01004             assert(params);
01005             if (m_dynlib.fnSetInstanceBus) {
01006                 m_dynlib.fnSetInstanceBus(params,bus.Get());
01007             }
01008         }
01009
01020         void SetPresetManager(CPresetLoader *preset_manager)
01021             {
01022                 if (!preset_manager) return;
01023                 if (!m_dynlib.fnSetPresetManager) return;
01024                 assert(params);
01025                 m_dynlib.fnSetPresetManager(params,
01026                     sound4cl_custom_reader,
01027                     sound4cl_custom_reader_free,
01028                     sound4cl_custom_writer,
01029                     sound4cl_custom_exists,
01030                     sound4cl_custom_getall,
01031                     sound4cl_custom_getall_free,
01032                     sound4cl_custom_remove,
01033                     sound4cl_custom_rename,
01034                     preset_manager->IsReadOnly(),
01035                     preset_manager
01036                 );
01037             }
01038
01061         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)
01062             {
01063                 assert(!instance);
01064                 std::string l_save_path;
01065                 if (!save_path.empty()) {
01066                     #if (__cplusplus<202002L)
01067                     l_save_path = save_path.u8string();
01068                     #else // C++20
01069                     l_save_path = (const char*)save_path.u8string().c_str();
01070                     #endif // C++20
01071                 }
01072                 if (frames_per_chunk==0) {
01073                     frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
01074                 }
01075                 if (m_dynlib.fnInitProcess3) {
01076                     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);
01077                 } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
01078                     return false;
```

```
01079                 } else {
01080                     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);
01081                 }
01082                 if (!instance) return false;
01083                 if (!init_metadata.empty()) {
01084                     for (auto&& [key,value]: init_metadata) {
01085                         m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
01086                     }
01087                     init_metadata.clear();
01088                 }
01089                 update_thread = std::thread([this, json_port](){
01090                     OnUpdateThreadStart();
01091                     m_dynlib.fnStartUpdateThread(instance, json_port);
01092                     OnUpdateThreadStop();
01093                 });
01094                 int timeout=5000;
01095                 #ifdef DEBUG
01096                     timeout=60*1000; // For debugger pauses
01097                 #endif
01098                 if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
01099                     return false;
01100                 }
01101                 return true;
01102             }
01103
01111         unsigned int GetChunkFrames() {
01112             if (instance && m_dynlib.fnGetProcessChunkFrames) {
01113                 return m_dynlib.fnGetProcessChunkFrames(instance);
01114             }
01115             return m_dynlib.fnGetChunkSizeInFrames();
01116         }
01117
01125         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
    change_kind)
01126             {
01127                 if (m_dynlib.fnPresetManager_InformChange) {
01128                     m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(),
    (sound4cl_PresetChange_Kind) change_kind);
01129                 }
01130             }
01131
01143         void SetMetadata(const std::string& key, const char* value)
01144             {
01145                 if (m_dynlib.fnSetMetadata) {
01146                     if (instance) {
01147                         m_dynlib.fnSetMetadata(instance, key.c_str(), value);
01148                     } else {
01149                         init_metadata.push_back( {key, value} );
01150                     }
01151                 }
01152             }
01153
01163         void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
01164             {
01165                 if (instance) {
01166                     if (m_dynlib.fnSetMetadataMulti) {
01167                         typedef const char* pchar;
01168                         pchar* keyvalue=new pchar[2*list.size()+1];
01169                         size_t n=0;
01170                         for (auto&& [key,value]: list) {
01171                             keyvalue[2*n+0]=key.c_str();
01172                             keyvalue[2*n+1]=value;
01173                             n++;
01174                         }
01175                         keyvalue[2*n+0]=nullptr;
01176
01177                         sound4cl_SetMetadataMulti(instance, keyvalue);
01178
01179                         delete[] keyvalue;
01180                     } else {
01181                         for (auto&& [key,value]: list) {
01182                             SetMetadata(key, value);
01183                         }
01184                     }
01185                 } else {
01186                     for (auto&& [key,value]: list) {
01187                         if (value) {
01188                             init_metadata.push_back( {key, value} );
01189                         }
01190                     }
01191                 }
01192             }
01193
01199         std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
01200             {
01201                 std::vector< std::tuple<std::string,std::string> > values;
```

```
01202                 if (m_dynlib.fnGetMetadataInfos) {
01203                     const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
01204                     if (c_values) {
01205                         for (const char** c_value=c_values; *c_value; ) {
01206                             std::string key(*c_value);
01207                             c_value++;
01208                             if (*c_value) {
01209                                 std::string descr(*c_value);
01210                                 c_value++;
01211                                 values.push_back( {key,descr} );
01212                             }
01213                         }
01214                         m_dynlib.fnFreeMetadataInfos(instance, c_values);
01215                     }
01216                 }
01217                 return values;
01218             }
01219
01223         virtual void OnUpdateThreadStart() {}
01227         virtual void OnUpdateThreadStop() {}
01228
01229
01243         bool StartWebServer(int http_port, int https_port=0)
01244             {
01245                 assert(instance);
01246                 assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01247                 if (!m_dynlib.fnWebserver) {
01248                     return false;
01249                 }
01250                 webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
01251                 if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01252                     return false;
01253                 }
01254                 int web_status=m_dynlib.fnWebserver_Status(webserver);
01255                 if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01256                     return false;
01257                 }
01258                 if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01259                     return false;
01260                 }
01261                 return true;
01262             }
01263
01271             void StopWebServer(int timeout_ms = 1000)
01272             {
01273                 if (!m_dynlib.fnWebserver_Stop) return;
01274                 if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01275                     m_dynlib.fnWebserver_Stop(webserver, timeout_ms);
01276                     webserver = SOUND4_INVALID_WEBSERVER_ID;
01277                 }
01278             }
01279
01283         void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
      content)
01284             {
01285                 assert(instance);
01286                 if (!m_dynlib.fnWebserver_SetAppHealth) return;
01287                 m_dynlib.fnWebserver_SetAppHealth(instance, httpcode, contenttype.c_str(),
      content.c_str());
01288             }
01292         void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
01293             {
01294                 assert(instance);
01295                 if (!m_dynlib.fnWebserver_GetAppHealth) return;
01296                 char* c_contenttype=nullptr;
01297                 char* c_content=nullptr;
01298                 m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
01299                 contenttype=c_contenttype;
01300                 content=c_content;
01301                 m_dynlib.fnWebserver_FreeString(c_contenttype);
01302                 m_dynlib.fnWebserver_FreeString(c_content);
01303             }
01312         int Terminate()
01313             {
01314                 if (!instance) return 0;
01315                 return m_dynlib.fnTerminateProcess(instance);
01316             }
01317
01323         void Stop()
01324             {
01325                 if (!instance) return;
01326                 if (m_bOwned) {
01327                     StopWebServer();
01328                     if (update_thread.joinable()) {
01329                         m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
01330                         update_thread.join();
01331                     }
```

```
01332                     m_dynlib.fnExitProcess(instance);
01333                 }
01334                 instance=nullptr;
01335             }
01336 
01346         unsigned int GetEstimatedDelay()
01347             {
01348                 assert(instance);
01349                 return m_dynlib.fnGetEstimatedDelay(instance);
01350             }
01351 
01364         float* GetBufferIn()
01365             {
01366                 assert(instance);
01367                 return m_dynlib.fnGetBufferIn(instance);
01368 
01369             }
01381         float* GetBufferOut()
01382             {
01383                 assert(instance);
01384                 return m_dynlib.fnGetBufferOut(instance);
01385             }
01389         void ProcessAudio()
01390             {
01391                 assert(instance);
01392                 m_dynlib.fnProcessAudio(instance, GetBufferIn(), GetBufferOut());
01393             }
01405         void ProcessAudio_Planar(float const * const *input, float * const *output)
01406             {
01407                 assert(instance);
01408                 m_dynlib.fnProcessAudio_Planar(instance, input, output);
01409             }
01410 
01422             template<typename T>
01423         std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01424             {
01425                 assert(instance);
01426                 std::vector<T> output;
01427                 unsigned int out_offset=0;
01428                 unsigned int in_offset=0;
01429                 unsigned int todo = input.size();
01430                 while (todo>0) {
01431                     unsigned int left = AddAudio(&input[in_offset], todo);
01432                     unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
01433                     output.resize(out_offset + out_avail);
01434                     GetAudio(&output[out_offset], out_avail);
01435                     out_offset+=out_avail;
01436                     in_offset += todo-left;
01437                     todo=left;
01438                 }
01439                 return output;
01440             }
01441 
01447         class CClient {
01448             public:
01449                 CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
01450                 : m_dynlib(dynlib)
01451                     {
01452                         assert(instance);
01453                         client=m_dynlib.fnNewClient(instance);
01454                         if (!client) throw std::bad_alloc();
01455                     }
01456                 // non-copyable
01457                 CClient( const CInstance& ) = delete;
01458                 CClient& operator=( const CInstance& ) = delete;
01459                 ~CClient()
01460                     {
01461                         if (client) {
01462                             m_dynlib.fnDeleteClient(client);
01463                             client=nullptr;
01464                         }
01465                     }
01473                 std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01474                     {
01475                         assert(client);
01476                         int need_save=0;
01477                         const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(),
    &need_save);
01478                         if (!canswer) return {};
01479                         std::string answer(canswer);
01480                         m_dynlib.fnFreeJsonAnswer (canswer);
01481                         if (NeedSave) {
01482                             *NeedSave=(need_save!=0);
01483                         }
01484                         return answer;
01485                     }
01486             private:
```

```
01487                 CProcessor& m_dynlib;
01488                 sound4cl_CClientInstance *client = nullptr;
01489         };
01497         std::shared_ptr<CClient> NewClient()
01498             {
01499                 assert(instance);
01500                 return std::make_shared<CClient>(instance, m_dynlib);
01501             }
01502
01513         bool SaveState() {
01514             assert(instance);
01515             if (!m_dynlib.fnSaveState || m_dynlib.fnSaveState(instance)!=0) {
01516                 return false;
01517             }
01518             return true;
01519         }
01520 #ifdef _WIN32
01527         void SetInstanceTracing(TraceLoggingHProvider tracing_provider, GUID activity_guid = {}) {
01528             if (m_dynlib.fnSetInstanceTracing) {
01529                 m_dynlib.fnSetInstanceTracing(params, tracing_provider, activity_guid);
01530             }
01531         }
01538         void SetInstanceTracingProcessActivity(GUID activity_guid) {
01539             assert(instance);
01540             if (m_dynlib.fnSetInstanceTracingProcessActivity) {
01541                 m_dynlib.fnSetInstanceTracingProcessActivity(instance, activity_guid);
01542             }
01543         }
01544 #endif // _WIN32
01545     protected:
01546             template<typename T>
01547         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01548             {
01549                 assert(instance);
01550                 return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
      helper::SampleFormat<T>::format);
01551             }
01552             template<typename T>
01553         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01554             {
01555                 assert(instance);
01556                 return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
      helper::SampleFormat<T>::format);
01557             }
01558
01559     private:
01560         CProcessor& m_dynlib;
01561         sound4cl_CParameters* params = nullptr;
01562         sound4cl_CInstance* instance = nullptr;
01563         std::thread update_thread;
01564         std::vector< std::pair<std::string,std::string> > init_metadata;
01565         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01566         bool m_bOwned = true;
01567     };
01568
01569
01570     // ------------------------------------------------------------
01576     enum ConverterTargetMode {
01577         ConverterTargetMode_AsInput      = sound4cl_ConverterTargetMode_AsInput,
01578         ConverterTargetMode_Mono         = sound4cl_ConverterTargetMode_Mono,
01579         ConverterTargetMode_Stereo       = sound4cl_ConverterTargetMode_Stereo,
01580         ConverterTargetMode_StereoToMono = sound4cl_ConverterTargetMode_StereoToMono,
01581         ConverterTargetMode_MonoToStereo = sound4cl_ConverterTargetMode_MonoToStereo,
01582     };
01583
01591     class CAudioConverter {
01592         public:
01598             CAudioConverter(CProcessor& dynlib)
01599                 : m_dynlib(dynlib)
01600                 { }
01601
01605             virtual ~CAudioConverter()
01606                 {
01607                     Delete();
01608                 }
01609
01625             bool Configure(SampleFormat fmt, size_t nFrame, size_t payl_ch, size_t payl_ch_offset,
      ConverterTargetMode target_mode)
01626                 {
01627                     Delete();
01628                     if (m_dynlib.fnGetAudioConverter)
01629                         m_cvt = m_dynlib.fnGetAudioConverter((sound4cl_SampleFormat)fmt, nFrame,
      payl_ch, payl_ch_offset, (sound4cl_ConverterTargetMode)target_mode);
01630                     return IsOK();
01631                 }
01632             void Delete()
01633                 {
```

```
01634                     if (m_cvt && m_dynlib.fnFreeAudioConverter) {
01635                         m_dynlib.fnFreeAudioConverter(m_cvt);
01636                     }
01637                     m_cvt = nullptr;
01638                 }
01639             bool IsOK() const { return (m_cvt!=nullptr); }
01640
01649             void ConvertFrom(const uint8_t *payload, float *output)
01650                 { m_dynlib.fnAudioConverter_From(m_cvt, payload, output); }
01651
01660             void ConvertTo(const float *input, uint8_t *payload)
01661                 { m_dynlib.fnAudioConverter_To(m_cvt, input, payload); }
01662
01663         private:
01664             CProcessor& m_dynlib;
01665             sound4cl_CAudioConverter *m_cvt = nullptr;
01666     };
01667
01668 }; // namespace sound4
01669
01670
01671 // bridge C callbacks to CPresetLoader
01672 extern "C" {
01673     static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
01674         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01675         std::string content=preset_loader->Read(filename);
01676         return strdup(content.c_str());
01677     }
01678     static void sound4cl_custom_reader_free(char *content, void* userdata) {
01679         if (!content) return;
01680         free(content);
01681     }
01682     static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01683         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01684         auto res=preset_loader->Write(filename,content);
01685         return res?0:-1;
01686     }
01687     static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
01688         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01689         auto res=preset_loader->Exists(filename);
01690         return res?0:-1;
01691     }
01692     static fs_char** sound4cl_custom_getall(void* userdata) {
01693         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01694         auto res=preset_loader->GetAll();
01695         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01696         for (size_t n=0;n<res.size();n++) {
01697             all[n]=fs_strdup(res[n].c_str());
01698         }
01699         all[res.size()]=nullptr;
01700         return all;
01701     }
01702     static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
01703         if (!all) return;
01704         for (fs_char** one=all;*one;one++) {
01705             free(*one);
01706         }
01707         free(all);
01708     }
01709     static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
01710         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01711         auto res=preset_loader->Remove(filename);
01712         return res?0:-1;
01713     }
01714     static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01715         sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
01716         auto res=preset_loader->Rename(from,to);
01717         return res?0:-1;
01718     }
01719
01720 };
```

15.27 sound4cl_cdef.h File Reference

C function signature definition for all sound4.cl libraries.

```
#include <stdint.h>
#include <stringapiset.h>
#include <wchar.h>
```

Macros

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

Typedefs

- typedef wchar_t fs_char
- typedef struct _tlgProvider_t const * TraceLoggingHProvider

15.27.1 Detailed Description

Used by sound4cl.hpp for dynamic library loading.

Definition in file sound4cl_cdef.h.

15.27.2 Macro Definition Documentation

15.27.2.1 SOUND4_INVALID_WEBSERVER_ID

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

Definition at line 67 of file sound4cl_cdef.h.

15.27.2.2 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 68 of file sound4cl_cdef.h.

15.27.2.3 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 69 of file sound4cl_cdef.h.

15.27.3 Typedef Documentation

15.27.3.1 fs_char

```
typedef wchar_t fs_char
```

Definition at line 19 of file sound4cl_cdef.h.

15.27.3.2 TraceLoggingHProvider

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

Definition at line 227 of file sound4cl_cdef.h.

15.28 sound4cl_cdef.h

Go to the documentation of this file.

```
00001
00007 #pragma once
00008
00009 #include <stdint.h>
00010 #ifdef __unix__
00011     #include <sys/types.h>
00012 #endif
00013
00014 #ifdef _WIN32
00015     #include <stringapiset.h>
00016     #include <wchar.h>
00018     #define fs_strdup wcsdup
00019     typedef wchar_t fs_char;
00020     #ifdef _MSC_VER
00021         #define __BYTE_ORDER__  REG_DWORD
00022         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00023         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00024     #endif
00025     #ifdef _MSC_VER
00026         #pragma warning(disable : 4996)
00027     #endif
00028 #else // !_WIN32
00030     #define fs_strdup strdup
00031     typedef char fs_char;
00032 #endif // !_WIN32
00033
00034 #ifdef __cplusplus
00035 extern "C" {
00036 #endif // __cplusplus
00037
00038 struct sound4cl_SOUND4_ProcessInfo {
00039     size_t struct_size;
00040     const char *name;
00041     const char *prefix;
00042     const char *shortname;
00043     const char *version;
00044 };
00045 enum sound4cl_LogSeverity {
00046     sound4cl_none     = 0,
00047     sound4cl_fatal    = 1,
00048     sound4cl_error    = 2,
00049     sound4cl_warning  = 3,
00050     sound4cl_info     = 4,
00051     sound4cl_verbose  = 5,
00052     sound4cl_verbose2 = 6,
00053     sound4cl_verbose3 = 7,
00054     sound4cl_verbose4 = 8,
00055     sound4cl_verbose5 = 9,
00056 };
00057 struct sound4cl_CInstance;
00058 struct sound4cl_CParameters;
00059 struct sound4cl_CBus;
00060 struct sound4cl_CClientInstance;
00061 struct sound4cl_CAudioConverter;
00062 enum sound4cl_PresetChange_Kind {
00063     change_kind_created = 1,
00064     change_kind_modified   ,
00065     change_kind_deleted    ,
00066 };
00067 #define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
00068 #define SOUND4_WEBSERVER_HTTP_OK               (1<<0)
00069 #define SOUND4_WEBSERVER_HTTPS_OK              (1<<1)
00070
00074 enum sound4cl_SampleFormat {
00075     INVALID_FORMAT=0,
00076     S16_LE,
00077     S16_BE,
00078     S24_LE,
00079     S24_BE,
```

```
00080     S32_LE,
00081     S32_BE,
00082     F32_LE,
00083     F32_BE,
00084
00085     // Native definitions, only links to the physical ones
00086     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00087         S16_NATIVE=S16_LE,
00088         S24_NATIVE=S24_LE,
00089         S32_NATIVE=S32_LE,
00090         F32_NATIVE=F32_LE,
00091     #else
00092         S16_NATIVE=S16_BE,
00093         S24_NATIVE=S24_BE,
00094         S32_NATIVE=S32_BE,
00095         F32_NATIVE=F32_BE,
00096     #endif
00097 };
00098
00102 enum sound4cl_ConverterTargetMode {
00103     sound4cl_ConverterTargetMode_AsInput = 0,
00104     sound4cl_ConverterTargetMode_Mono,
00105     sound4cl_ConverterTargetMode_Stereo,
00106     sound4cl_ConverterTargetMode_StereoToMono,
00107     sound4cl_ConverterTargetMode_MonoToStereo,
00108 };
00109
00110 const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
00111 const char *sound4cl_GetVersion();
00112 unsigned int sound4cl_GetChunkSizeInFrames();
00113 unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
00114 unsigned int sound4cl_GetChannelCount();
00115 unsigned int sound4cl_GetAudioInputCount();
00116 unsigned int sound4cl_GetAudioOutputCount();
00117 unsigned int sound4cl_GetSampleRate();
00118 typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
00119 void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
00120 void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
00121 struct sound4cl_CParameters *sound4cl_NewParameters();
00122 void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
00123 void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
00124 const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
00125 void sound4cl_FreeParameterValue(const char *value);
00126 struct sound4cl_CInstance *sound4cl_InitProcess(
00127         const char *LoginKey,
00128         const char *RadioName,
00129         const char *Access_Key_ID,
00130         const char *Access_Key_Secret,
00131         const char *save_path
00132     );
00133 struct sound4cl_CInstance *sound4cl_InitProcess2(
00134         const char *LoginKey,
00135         const char *RadioName,
00136         const char *Access_Key_ID,
00137         const char *Access_Key_Secret,
00138         const char *save_path,
00139         const struct sound4cl_CParameters *parameters
00140     );
00141 struct sound4cl_CInstance *sound4cl_InitProcess3(
00142         const char *LoginKey,
00143         const char *RadioName,
00144         const char *Access_Key_ID,
00145         const char *Access_Key_Secret,
00146         const char *save_path,
00147         const struct sound4cl_CParameters *parameters,
00148         unsigned int frames_per_chunk
00149     );
00150 unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
00151 int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
00152 void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
00153 struct sound4cl_CBus* sound4cl_NewBus();
00154 void sound4cl_FreeBus(struct sound4cl_CBus *bus);
00155 void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
00156 typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
00157 typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
00158 typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00159 typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
00160 typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
00161 typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
00162 typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
00163 typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00164 void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
00165     , sound4cl_storage_reader reader, sound4cl_storage_reader_free
00166     , sound4cl_storage_writer writer
00167     , sound4cl_storage_exists exists
00168     , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
00169     , sound4cl_storage_remove remove
```

```
00170     , sound4cl_storage_rename rename
00171     , int IsReadOnly
00172     , void* userdata
00173     );
00174 void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
      *relative_path, enum sound4cl_PresetChange_Kind change_kind);
00175 void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
00176 void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
00177 const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
00178 void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
00179 void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
00180 void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
00181 int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
00182 void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
00183 void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
00184     , float const * const *input
00185     , float * const *output);
00186 float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
00187 float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
00188 unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
00189 const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
00190 enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
00191 unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
00192 unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
00193 unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
      int nFrame, enum sound4cl_SampleFormat fmt );
00194 unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
00195 unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
00196 unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
      max_nFrame, enum sound4cl_SampleFormat fmt );
00197 void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00198 void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
      sound4cl_SampleFormat fmt);
00199 void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
00200 void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
00201 void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
      nFrame);
00202 void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
00203 void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
00204 void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
00205 struct sound4cl_CAudioConverter* sound4cl_GetAudioConverter(enum sound4cl_SampleFormat fmt, size_t
      nFrame, size_t payl_ch, size_t payl_ch_offset, sound4cl_ConverterTargetMode tartget_mode);
00206 void sound4cl_FreeAudioConverter(struct sound4cl_CAudioConverter *converter);
00207 void sound4cl_AudioConverter_From(struct sound4cl_CAudioConverter *converter, const uint8_t *payload,
      float *output);
00208 void sound4cl_AudioConverter_To(struct sound4cl_CAudioConverter *converter, const float *input,
      uint8_t *payload);
00209 struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
00210 void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
00211 const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
      *need_save);
00212 void sound4cl_FreeJsonAnswer(const char *json_str);
00213 int sound4cl_SaveState(struct sound4cl_CInstance *instance);
00214 uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port);
00215 uint64_t sound4cl_Webserver_tcp2(unsigned int listenport, unsigned int listenport_secure, const char
      *socket_ip, unsigned int socket_port, const struct sound4cl_CParameters *parameters);
00216 uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
      sound4cl_CInstance *instance);
00217 int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
00218 int sound4cl_Webserver_Status(uint64_t id);
00219 void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
      contenttype, const char* content);
00220 void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
      contenttype, char** content);
00221 void sound4cl_Webserver_FreeString(char* str);
00222
00223 //   Tracing (Windows only)
00224 #ifdef _WIN32
00225     #ifndef TLG_HAVE_EVENT_SET_INFORMATION // TraceLoggingProvider.h has not been included
00226         struct _tlgProvider_t;
00227         typedef struct _tlgProvider_t const* TraceLoggingHProvider;
00228     #endif // !TLG_HAVE_EVENT_SET_INFORMATION
00229 void sound4cl_SetInstanceTracing(struct sound4cl_CParameters *params, TraceLoggingHProvider
      tracing_provider, GUID activity_guid);
00230 void sound4cl_SetInstanceTracingProcessActivity(struct sound4cl_CInstance *instance, GUID
      activity_guid);
00231 #endif // _WIN32
00232
00233
00234 #ifdef __cplusplus
00235 } // extern "C"
00236 #endif // __cplusplus
```

Index

_log_cb
  sound4, 90
  sound4::cloudx1, 96
  sound4::cloudx1::dyn, 105
_log_cb_c
  sound4, 89
  sound4::cloudx1, 92
  sound4::cloudx1::dyn, 99
~CAudioConverter
  sound4::CAudioConverter, 122
  sound4::cloudx1::CAudioConverter, 124
  sound4::cloudx1::dyn::CAudioConverter, 127
~CBus
  sound4::CBus, 131
~CClient
  sound4::CInstance::CClient, 132
  sound4::cloudx1::CInstance::CClient, 134
  sound4::cloudx1::dyn::CInstance::CClient, 135
~CDynLoader
  sound4::cloudx1::dyn::helper::CDynLoader, 149
  sound4::helper::CDynLoader, 152
~CInstance
  sound4::CInstance, 156
  sound4::cloudx1::CInstance, 168
  sound4::cloudx1::dyn::CInstance, 180
~CPresetLoader
  sound4::cloudx1::CPresetLoader, 192
  sound4::cloudx1::dyn::CPresetLoader, 194
  sound4::CPresetLoader, 196
~MyPresetManager
  MyPresetManager, 208

ActivateAuxiliaryAudioID
  sound4::cloudx1::CInstance, 168
AddAudio
  sound4::CInstance, 156
  sound4::cloudx1::CInstance, 168
  sound4::cloudx1::dyn::CDynLib, 138
  sound4::cloudx1::dyn::CInstance, 180
AddPadAudio
  sound4::cloudx1::dyn::CDynLib, 138
Audio convert helpers, 29
  cloudx1_AudioConverter_From, 30
  cloudx1_AudioConverter_To, 30
  cloudx1_AudioConvertFrom, 31
  cloudx1_AudioConvertTo, 31
  cloudx1_AudioMonoFromLiveStereo, 32
  cloudx1_AudioMonoToLiveStereo, 32
  cloudx1_ConverterTargetMode, 30
  cloudx1_FreeAudioConverter, 32
  cloudx1_GetAudioConverter, 32
  cloudx1_MonoToStereo, 33
  cloudx1_MonoToStereo_Planar, 34
  cloudx1_StereoToMono, 34
  cloudx1_StereoToMono_Planar, 34
  ConverterTargetMode_AsInput, 30
  ConverterTargetMode_Mono, 30
  ConverterTargetMode_MonoToStereo, 30
  ConverterTargetMode_Stereo, 30
  ConverterTargetMode_StereoToMono, 30
Audio convert helpers (C++), 35
  AudioConvertFrom, 36
  AudioConvertTo, 36
  AudioMonoFromLiveStereo, 37
  AudioMonoToLiveStereo, 37
  ConverterTargetMode, 36
  ConverterTargetMode_AsInput, 36
  ConverterTargetMode_Mono, 36
  ConverterTargetMode_MonoToStereo, 36
  ConverterTargetMode_Stereo, 36
  ConverterTargetMode_StereoToMono, 36
  MonoToStereo, 37
  MonoToStereo_Planar, 38
  StereoToMono, 38
  StereoToMono_Planar, 39
Audio managing (any audio), 39
  cloudx1_AddAudio, 41
  cloudx1_AddPadAudio, 41
  cloudx1_GetAudio, 41
  cloudx1_GetBytesFromFormat, 43
  cloudx1_GetFormatFromName, 43
  cloudx1_GetFormatName, 43
  cloudx1_GetMaxPacketFrame, 44
  cloudx1_GetOutputCount, 44
  cloudx1_SampleFormat, 40
  F32_BE, 40
  F32_LE, 40
  F32_NATIVE, 40
  INVALID_FORMAT, 40
  S16_BE, 40
  S16_LE, 40
  S16_NATIVE, 40
  S24_BE, 40
  S24_LE, 40
  S24_NATIVE, 40
  S32_BE, 40
  S32_LE, 40
  S32_NATIVE, 40
AudioConverter_From

sound4::cloudx1::dyn::CDynLib, 139
AudioConverter_To
sound4::cloudx1::dyn::CDynLib, 139
AudioConvertFrom
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 139
sound4::cloudx1::dyn::helper, 108
sound4::cloudx1::helper, 113
AudioConvertTo
Audio convert helpers (C++), 36
sound4::cloudx1::dyn::CDynLib, 139
sound4::cloudx1::dyn::helper, 108
sound4::cloudx1::helper, 113
AudioMonoFromLiveStereo
Audio convert helpers (C++), 37
sound4::cloudx1::dyn::CDynLib, 139
sound4::cloudx1::dyn::helper, 109
sound4::cloudx1::helper, 113
AudioMonoToLiveStereo
Audio convert helpers (C++), 37
sound4::cloudx1::dyn::CDynLib, 139
sound4::cloudx1::dyn::helper, 109
sound4::cloudx1::helper, 114
Auxiliary outputs, 44
cloudx1_ActivateAuxiliaryAudioID, 45
cloudx1_FreeAuxiliaryAudioInfos, 45
cloudx1_GetAuxiliaryAudioEstimatedDelay, 45
cloudx1_GetAuxiliaryAudioOutputInfos, 46

buffer
sound4::AuxiliaryOutputInfos, 119
SOUND4_AuxiliaryOutputInfos, 214

CAudioConverter
sound4::CAudioConverter, 121
sound4::cloudx1::CAudioConverter, 124
sound4::cloudx1::dyn::CAudioConverter, 127
sound4::CProcessor, 203
CBus
sound4::CBus, 131
sound4::CProcessor, 203
CClient
sound4::CInstance::CClient, 132
sound4::cloudx1::CInstance::CClient, 133
sound4::cloudx1::dyn::CInstance::CClient, 135
CDynLib
sound4::cloudx1::dyn::CDynLib, 138
CDynLoader
sound4::cloudx1::dyn::helper::CDynLoader, 149
sound4::helper::CDynLoader, 152
change_kind_created
Custom preset managagement, 53
sound4, 89
change_kind_deleted
Custom preset managagement, 53
sound4, 89
change_kind_modified
Custom preset managagement, 53
sound4, 89
CHANNEL_COUNT
sound4.x1.cloud-pipe.cpp, 227
channel_count
sound4::AuxiliaryOutputInfos, 119
SOUND4_AuxiliaryOutputInfos, 214
CHUNK_SIZE
sound4.x1.cloud-pipe.cpp, 227
ChunkFrames
sound4::cloudx1, 96
sound4::cloudx1::dyn, 105
CInstance
sound4::CBus, 131
sound4::CInstance, 155
sound4::cloudx1::CInstance, 168
sound4::cloudx1::dyn::CInstance, 180
sound4::CProcessor, 203
Client's JSon config interfacing, 46
cloudx1_DeleteClient, 47
cloudx1_FreeJsonAnswer, 47
cloudx1_NewClient, 47
cloudx1_ProcessJson, 48
cloudx1_SaveState, 48
Close
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::helper::CDynLoader, 152
cloudx1_ActivateAuxiliaryAudioID
Auxiliary outputs, 45
cloudx1_AddAudio
Audio managing (any audio), 41
cloudx1_AddPadAudio
Audio managing (any audio), 41
CLOUDX1_AUDIO_INPUT_COUNT
Global setup, 59
CLOUDX1_AUDIO_OUTPUT_COUNT
Global setup, 59
cloudx1_AudioConverter_From
Audio convert helpers, 30
cloudx1_AudioConverter_To
Audio convert helpers, 30
cloudx1_AudioConvertFrom
Audio convert helpers, 31
cloudx1_AudioConvertTo
Audio convert helpers, 31
CLOUDX1_AUDIOFRAME_COUNT
Global setup, 59
cloudx1_AudioMonoFromLiveStereo
Audio convert helpers, 32
cloudx1_AudioMonoToLiveStereo
Audio convert helpers, 32
cloudx1_CAudioConverter, 191
cloudx1_CClientInstance, 191
CLOUDX1_CHANNEL_COUNT
Global setup, 59
cloudx1_CInstance, 191
cloudx1_ConverterTargetMode
Audio convert helpers, 30
cloudx1_CParameters, 191
cloudx1_custom_exists

sound4.x1.cloud.hpp, 247
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_getall
sound4.x1.cloud.hpp, 247
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_getall_free
sound4.x1.cloud.hpp, 247
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_reader
sound4.x1.cloud.hpp, 248
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_reader_free
sound4.x1.cloud.hpp, 248
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_remove
sound4.x1.cloud.hpp, 248
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_rename
sound4.x1.cloud.hpp, 248
sound4.x1.cloud_dyn.hpp, 260
cloudx1_custom_writer
sound4.x1.cloud.hpp, 248
sound4.x1.cloud_dyn.hpp, 261
CLOUDX1_DECL
sound4.x1.cloud.h, 240
cloudx1_DeleteClient
Client's JSon config interfacing, 47
cloudx1_ExitProcess
Process creation/destruction, 67
cloudx1_FreeAudioConverter
Audio convert helpers, 32
cloudx1_FreeAuxiliaryAudioInfos
Auxiliary outputs, 45
cloudx1_FreeJsonAnswer
Client's JSon config interfacing, 47
cloudx1_FreeMetadataInfos
Metadata Management, 65
cloudx1_FreeParameters
Process creation/destruction, 67
cloudx1_FreeParameterValue
Process creation/destruction, 67
cloudx1_GetAudio
Audio managing (any audio), 41
cloudx1_GetAudioConverter
Audio convert helpers, 32
cloudx1_GetAudioInputCount
Global setup, 61
cloudx1_GetAudioOutputCount
Global setup, 61
cloudx1_GetAuxiliaryAudioEstimatedDelay
Auxiliary outputs, 45
cloudx1_GetAuxiliaryAudioOutputInfos
Auxiliary outputs, 46
cloudx1_GetBufferIn
Direct processing, 55
cloudx1_GetBufferOut
Direct processing, 55
cloudx1_GetBytesFromFormat
Audio managing (any audio), 43
cloudx1_GetChannelCount
Global setup, 61
cloudx1_GetChunkSizeInFrames
Global setup, 61
cloudx1_GetEstimatedDelay
Direct processing, 56
cloudx1_GetFormatFromName
Audio managing (any audio), 43
cloudx1_GetFormatName
Audio managing (any audio), 43
cloudx1_GetMaxPacketFrame
Audio managing (any audio), 44
cloudx1_GetMetadataInfos
Metadata Management, 65
cloudx1_GetOutputCount
Audio managing (any audio), 44
cloudx1_GetParameter
Process creation/destruction, 68
cloudx1_GetPossibleChunkSizeInFrames
Global setup, 62
cloudx1_GetProcessChunkFrames
Process creation/destruction, 68
cloudx1_GetSampleRate
Global setup, 62
cloudx1_GetVersion
Global setup, 62
CLOUDX1_HAS_CLOUDBUS
sound4.x1.cloud.h, 240
CLOUDX1_HAS_WEBSERVER
sound4.x1.cloud.h, 240
cloudx1_InitProcess
Process creation/destruction, 68
cloudx1_InitProcess2
Process creation/destruction, 69
cloudx1_InitProcess3
Process creation/destruction, 69
cloudx1_loggerfn
Global setup, 60
cloudx1_LogSeverity
Global setup, 60
cloudx1_MonoToStereo
Audio convert helpers, 33
cloudx1_MonoToStereo_Planar
Audio convert helpers, 34
cloudx1_NewClient
Client's JSon config interfacing, 47
cloudx1_NewParameters
Process creation/destruction, 70
cloudx1_PresetChange_Kind
Custom preset managagement, 53
cloudx1_PresetManager_InformChange
Custom preset managagement, 53
cloudx1_ProcessAudio
Direct processing, 56
cloudx1_ProcessAudio_Planar
Direct processing, 57
cloudx1_ProcessJson

Client's JSon config interfacing, 48
CLOUDX1_SAMPLE_RATE
Global setup, 59
cloudx1_SampleFormat
Audio managing (any audio), 40
cloudx1_SaveState
Client's JSon config interfacing, 48
cloudx1_SetInstanceTracing
Instance tracing (Windows only), 64
cloudx1_SetInstanceTracingProcessActivity
Instance tracing (Windows only), 64
cloudx1_SetLoggerCallback
Global setup, 63
cloudx1_SetLogSeverity
Global setup, 63
cloudx1_SetMetadata
Metadata Management, 65
cloudx1_SetMetadataMulti
Metadata Management, 66
cloudx1_SetParameter
Process creation/destruction, 70
cloudx1_SetPresetManager
Custom preset managagement, 54
cloudx1_StartUpdateThread
Update Management, 78
cloudx1_StereoToMono
Audio convert helpers, 34
cloudx1_StereoToMono_Planar
Audio convert helpers, 34
cloudx1_StopUpdateThread
Update Management, 78
cloudx1_storage_exists
Custom preset managagement, 50
cloudx1_storage_getall
Custom preset managagement, 50
cloudx1_storage_getall_free
Custom preset managagement, 51
cloudx1_storage_reader
Custom preset managagement, 51
cloudx1_storage_reader_free
Custom preset managagement, 51
cloudx1_storage_remove
Custom preset managagement, 52
cloudx1_storage_rename
Custom preset managagement, 52
cloudx1_storage_writer
Custom preset managagement, 52
cloudx1_TerminateProcess
Process creation/destruction, 70
cloudx1_WaitUpdateThreadReady
Update Management, 79
cloudx1_Webserver
Web server, 80
cloudx1_Webserver_FreeString
Web server, 81
cloudx1_Webserver_GetAppHealth
Web server, 81
cloudx1_Webserver_SetAppHealth
Web server, 81
cloudx1_Webserver_Status
Web server, 83
cloudx1_Webserver_Stop
Web server, 83
cloudx1_Webserver_tcp
Web server, 84
cloudx1_Webserver_tcp2
Web server, 84
Configure
sound4::CAudioConverter, 122
sound4::cloudx1::CAudioConverter, 125
sound4::cloudx1::dyn::CAudioConverter, 127
ConverterTargetMode
Audio convert helpers (C++), 36
ConverterTargetMode_AsInput
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_Mono
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_MonoToStereo
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_Stereo
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConverterTargetMode_StereoToMono
Audio convert helpers, 30
Audio convert helpers (C++), 36
ConvertFrom
sound4::CAudioConverter, 122
sound4::cloudx1::CAudioConverter, 125
sound4::cloudx1::dyn::CAudioConverter, 129
ConvertTo
sound4::CAudioConverter, 123
sound4::cloudx1::CAudioConverter, 126
sound4::cloudx1::dyn::CAudioConverter, 129
CPresetLoader
sound4::cloudx1::CPresetLoader, 192
sound4::cloudx1::dyn::CPresetLoader, 194
sound4::CPresetLoader, 196
CProcessor
sound4::CProcessor, 199
Create
sound4::CInstance, 156
sound4::cloudx1::CInstance, 168
sound4::cloudx1::dyn::CInstance, 180
Custom preset managagement, 49
change_kind_created, 53
change_kind_deleted, 53
change_kind_modified, 53
cloudx1_PresetChange_Kind, 53
cloudx1_PresetManager_InformChange, 53
cloudx1_SetPresetManager, 54
cloudx1_storage_exists, 50
cloudx1_storage_getall, 50
cloudx1_storage_getall_free, 51

cloudx1_storage_reader, 51
cloudx1_storage_reader_free, 51
cloudx1_storage_remove, 52
cloudx1_storage_rename, 52
cloudx1_storage_writer, 52
fs_char, 53

Delete
sound4::CAudioConverter, 123
sound4::cloudx1::CAudioConverter, 126
sound4::cloudx1::dyn::CAudioConverter, 130
DeleteClient
sound4::cloudx1::dyn::CDynLib, 139
Deprecated List, 17
description
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 214
Direct processing, 54
cloudx1_GetBufferIn, 55
cloudx1_GetBufferOut, 55
cloudx1_GetEstimatedDelay, 56
cloudx1_ProcessAudio, 56
cloudx1_ProcessAudio_Planar, 57
doc_examples.h, 217
doc_gencpp.h, 217
doc_json.h, 217
doc_main.h, 218
doc_params.h, 218
DynFuncHolder
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 204
sound4::helper::DynFuncHolder< T >, 206

Environment variables and parameters, 13
error
Global setup, 60
sound4, 88
Examples, 57
Exists
MyPresetManager, 208
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 194
sound4::CPresetLoader, 196
ExitProcess
sound4::cloudx1::dyn::CDynLib, 140

F32_BE
Audio managing (any audio), 40
sound4, 89
F32_LE
Audio managing (any audio), 40
sound4, 89
F32_NATIVE
Audio managing (any audio), 40
sound4, 89
fatal
Global setup, 60
sound4, 88
FreeAudioConverter
sound4::cloudx1::dyn::CDynLib, 140
FreeJsonAnswer
sound4::cloudx1::dyn::CDynLib, 140
FreeMetadataInfos
sound4::cloudx1::dyn::CDynLib, 140
FreeParameters
sound4::cloudx1::dyn::CDynLib, 140
FreeParameterValue
sound4::cloudx1::dyn::CDynLib, 140
fs_char
Custom preset managagement, 53
sound4cl_cdef.h, 293

Generic C++ access to SOUND4 .CL librairies, 3
Get
sound4::CBus, 131
GetAll
MyPresetManager, 208
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 194
sound4::CPresetLoader, 196
GetAudio
sound4::CInstance, 157
sound4::cloudx1::CInstance, 169
sound4::cloudx1::dyn::CDynLib, 140
sound4::cloudx1::dyn::CInstance, 181
GetAudioConverter
sound4::cloudx1::dyn::CDynLib, 141
GetAudioInputCount
sound4::cloudx1, 92
sound4::cloudx1::dyn, 99
sound4::cloudx1::dyn::CDynLib, 141
sound4::CProcessor, 199
GetAudioOutputCount
sound4::cloudx1, 92
sound4::cloudx1::dyn, 100
sound4::cloudx1::dyn::CDynLib, 141
sound4::CProcessor, 199
GetAuxiliaryAudioOutputInfos
sound4::cloudx1::CInstance, 169
GetBufferIn
sound4::CInstance, 157
sound4::cloudx1::CInstance, 170
sound4::cloudx1::dyn::CDynLib, 141
sound4::cloudx1::dyn::CInstance, 181
GetBufferOut
sound4::CInstance, 157
sound4::cloudx1::CInstance, 170
sound4::cloudx1::dyn::CDynLib, 141
sound4::cloudx1::dyn::CInstance, 182
GetBytesFromFormat
sound4::cloudx1, 92
sound4::cloudx1::dyn, 100
sound4::cloudx1::dyn::CDynLib, 141
sound4::CProcessor, 199
GetChannelCount
sound4::cloudx1, 93
sound4::cloudx1::dyn, 100
sound4::cloudx1::dyn::CDynLib, 141

sound4::CProcessor, 200
GetChunkFrames
sound4::CInstance, 158
sound4::cloudx1::CInstance, 170
sound4::cloudx1::dyn::CInstance, 182
GetChunkSizeInFrames
sound4::cloudx1, 93
sound4::cloudx1::dyn, 101
sound4::cloudx1::dyn::CDynLib, 142
sound4::CProcessor, 200
GetEnv
sound4.x1.cloud-pipe-cxx.cpp, 218
sound4.x1.cloud-pipe-dyn.cpp, 222
sound4cl-pipe.cpp, 232
GetEstimatedDelay
sound4::CInstance, 158
sound4::cloudx1::CInstance, 171
sound4::cloudx1::dyn::CDynLib, 142
sound4::cloudx1::dyn::CInstance, 182
GetFormatFromName
sound4::cloudx1, 93
sound4::cloudx1::dyn, 101
sound4::cloudx1::dyn::CDynLib, 142
sound4::CProcessor, 200
GetFormatName
sound4::cloudx1, 94
sound4::cloudx1::dyn, 102
sound4::cloudx1::dyn::CDynLib, 142
sound4::CProcessor, 201
GetMaxPacketFrame
sound4::cloudx1::dyn::CDynLib, 142
GetMetadataInfos
sound4::CInstance, 158
sound4::cloudx1::CInstance, 171
sound4::cloudx1::dyn::CDynLib, 142
sound4::cloudx1::dyn::CInstance, 183
GetOutputCount
sound4::cloudx1::dyn::CDynLib, 142
GetParam
sound4::CInstance, 159
sound4::cloudx1::CInstance, 171
sound4::cloudx1::dyn::CInstance, 183
GetParameter
sound4::cloudx1::dyn::CDynLib, 143
GetPossibleChunkSizeInFrames
sound4::cloudx1, 94
sound4::cloudx1::dyn, 102
sound4::cloudx1::dyn::CDynLib, 143
sound4::CProcessor, 201
GetPrefixSymbol
sound4::CProcessor, 201
GetProcessChunkFrames
sound4::cloudx1::dyn::CDynLib, 143
GetSampleRate
sound4::cloudx1, 94
sound4::cloudx1::dyn, 103
sound4::cloudx1::dyn::CDynLib, 143
sound4::CProcessor, 202
GetSymbol
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::helper::CDynLoader, 152
GetSymbol_safe
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::helper::CDynLoader, 152
GetThisLibraryPath
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::helper::CDynLoader, 153
GetVersion
sound4::cloudx1, 95
sound4::cloudx1::dyn, 103
sound4::cloudx1::dyn::CDynLib, 143
sound4::CProcessor, 202
GetWebServerAppHealth
sound4::CInstance, 159
sound4::cloudx1::CInstance, 172
sound4::cloudx1::dyn::CInstance, 183
Global setup, 58
CLOUDX1_AUDIO_INPUT_COUNT, 59
CLOUDX1_AUDIO_OUTPUT_COUNT, 59
CLOUDX1_AUDIOFRAME_COUNT, 59
CLOUDX1_CHANNEL_COUNT, 59
cloudx1_GetAudioInputCount, 61
cloudx1_GetAudioOutputCount, 61
cloudx1_GetChannelCount, 61
cloudx1_GetChunkSizeInFrames, 61
cloudx1_GetPossibleChunkSizeInFrames, 62
cloudx1_GetSampleRate, 62
cloudx1_GetVersion, 62
cloudx1_loggerfn, 60
cloudx1_LogSeverity, 60
CLOUDX1_SAMPLE_RATE, 59
cloudx1_SetLoggerCallback, 63
cloudx1_SetLogSeverity, 63
error, 60
fatal, 60
info, 60
none, 60
SOUND4_GetProcessInfo, 63
verbose, 60
verbose2, 60
verbose3, 60
verbose4, 60
verbose5, 60
warning, 60

id
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 214
info
Global setup, 60
sound4, 88
InitProcess
sound4::cloudx1::dyn::CDynLib, 143
InitProcess2
sound4::cloudx1::dyn::CDynLib, 143
InitProcess3
sound4::cloudx1::dyn::CDynLib, 144

InputChannels
sound4::cloudx1, 96
sound4::cloudx1::dyn, 105
InputSampleSize
sound4::cloudx1, 96
sound4::cloudx1::dyn, 106
Instance tracing (Windows only), 64
cloudx1_SetInstanceTracing, 64
cloudx1_SetInstanceTracingProcessActivity, 64
INVALID_FORMAT
Audio managing (any audio), 40
sound4, 89
IsOK
sound4::CAudioConverter, 123
sound4::cloudx1::CAudioConverter, 126
sound4::cloudx1::dyn::CAudioConverter, 130
IsOk
sound4::CInstance, 159
sound4::cloudx1::CInstance, 172
sound4::cloudx1::dyn::CDynLib, 138
sound4::cloudx1::dyn::CInstance, 184
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 205
sound4::CProcessor, 202
sound4::helper::CDynLoader, 153
sound4::helper::DynFuncHolder< T >, 206
IsReadOnly
MyPresetManager, 209
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 195
sound4::CPresetLoader, 197

JSON syntax, 5
JSON syntax for internal setup, 9

Load
sound4::cloudx1::dyn::CDynLib, 138
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::CProcessor, 202
sound4::helper::CDynLoader, 153
log_cb_t
sound4, 88
sound4::cloudx1, 91
sound4::cloudx1::dyn, 99
LogSeverity
sound4, 88
sound4::cloudx1, 91
sound4::cloudx1::dyn, 99

main
sound4.x1.cloud-pipe-cxx.cpp, 218
sound4.x1.cloud-pipe-dyn.cpp, 222
sound4.x1.cloud-pipe.cpp, 228
sound4cl-pipe.cpp, 232
Metadata Management, 65
cloudx1_FreeMetadataInfos, 65
cloudx1_GetMetadataInfos, 65
cloudx1_SetMetadata, 65
cloudx1_SetMetadataMulti, 66
MonoToStereo
Audio convert helpers (C++), 37
sound4::cloudx1::dyn::CDynLib, 144
sound4::cloudx1::dyn::helper, 109
sound4::cloudx1::helper, 114
MonoToStereo_Planar
Audio convert helpers (C++), 38
sound4::cloudx1::dyn::CDynLib, 144
sound4::cloudx1::dyn::helper, 110
sound4::cloudx1::helper, 115
MyLogger
sound4.x1.cloud-pipe-cxx.cpp, 219
sound4.x1.cloud-pipe-dyn.cpp, 223
sound4.x1.cloud-pipe.cpp, 228
MyPresetManager, 207
~MyPresetManager, 208
Exists, 208
GetAll, 208
IsReadOnly, 209
MyPresetManager, 208
Read, 209
Remove, 209
Rename, 210
SetupPath, 210
Write, 210

name
SOUND4_ProcessInfo, 215
NewClient
sound4::CInstance, 159
sound4::cloudx1::CInstance, 172
sound4::cloudx1::dyn::CDynLib, 144
sound4::cloudx1::dyn::CInstance, 184
NewParameters
sound4::cloudx1::dyn::CDynLib, 144
none
Global setup, 60
sound4, 88

OnUpdateThreadStart
sound4::CInstance, 160
sound4::cloudx1::CInstance, 172
sound4::cloudx1::dyn::CInstance, 184
OnUpdateThreadStop
sound4::CInstance, 160
sound4::cloudx1::CInstance, 172
sound4::cloudx1::dyn::CInstance, 184
operator bool
sound4::cloudx1::dyn::helper::CDynLoader, 150
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 205
sound4::helper::CDynLoader, 153
sound4::helper::DynFuncHolder< T >, 206
operator T*
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 205
sound4::helper::DynFuncHolder< T >, 206
operator=

sound4::CInstance, 160
sound4::CInstance::CClient, 132
sound4::cloudx1::CInstance, 173
sound4::cloudx1::CInstance::CClient, 134
sound4::cloudx1::dyn::CInstance, 184
sound4::cloudx1::dyn::CInstance::CClient, 135
sound4::cloudx1::dyn::helper::CDynLoader, 151
sound4::helper::CDynLoader, 153
OutputChannels
sound4::cloudx1, 96
sound4::cloudx1::dyn, 106
OutputSampleSize
sound4::cloudx1, 97
sound4::cloudx1::dyn, 106

prefix
SOUND4_ProcessInfo, 215
PresetChange_Kind
sound4, 88
sound4::cloudx1, 91
sound4::cloudx1::dyn, 99
PresetManager_InformChange
sound4::CInstance, 160
sound4::cloudx1::CInstance, 173
sound4::cloudx1::dyn::CDynLib, 144
sound4::cloudx1::dyn::CInstance, 184
Process creation/destruction, 66
cloudx1_ExitProcess, 67
cloudx1_FreeParameters, 67
cloudx1_FreeParameterValue, 67
cloudx1_GetParameter, 68
cloudx1_GetProcessChunkFrames, 68
cloudx1_InitProcess, 68
cloudx1_InitProcess2, 69
cloudx1_InitProcess3, 69
cloudx1_NewParameters, 70
cloudx1_SetParameter, 70
cloudx1_TerminateProcess, 70
process_name
sound4::cloudx1, 97
sound4::cloudx1::dyn, 106
process_shortname
sound4::cloudx1, 97
sound4::cloudx1::dyn, 106
ProcessAnyAudio
sound4::CInstance, 161
sound4::cloudx1::CInstance, 173
sound4::cloudx1::dyn::CInstance, 185
ProcessAudio
sound4::CInstance, 161
sound4::cloudx1::CInstance, 174
sound4::cloudx1::dyn::CDynLib, 144
sound4::cloudx1::dyn::CInstance, 185
ProcessAudio_Planar
sound4::CInstance, 161
sound4::cloudx1::CInstance, 174
sound4::cloudx1::dyn::CDynLib, 145
sound4::cloudx1::dyn::CInstance, 185
ProcessJson
sound4::CInstance::CClient, 132
sound4::cloudx1::CInstance::CClient, 134
sound4::cloudx1::dyn::CDynLib, 145
sound4::cloudx1::dyn::CInstance::CClient, 135
ProcessLoop
sound4.x1.cloud-pipe-cxx.cpp, 219
sound4.x1.cloud-pipe-dyn.cpp, 223
sound4.x1.cloud-pipe.cpp, 228
sound4cl-pipe.cpp, 233

Read
MyPresetManager, 209
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 195
sound4::CPresetLoader, 197
Remove
MyPresetManager, 209
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 195
sound4::CPresetLoader, 197
Rename
MyPresetManager, 210
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 195
sound4::CPresetLoader, 197

S16_BE
Audio managing (any audio), 40
sound4, 89
S16_LE
Audio managing (any audio), 40
sound4, 89
S16_NATIVE
Audio managing (any audio), 40
sound4, 89
S24_BE
Audio managing (any audio), 40
sound4, 89
S24_LE
Audio managing (any audio), 40
sound4, 89
S24_NATIVE
Audio managing (any audio), 40
sound4, 89
S32_BE
Audio managing (any audio), 40
sound4, 89
S32_LE
Audio managing (any audio), 40
sound4, 89
S32_NATIVE
Audio managing (any audio), 40
sound4, 89
Sample using C interface, 71
Sample using C++ interface wrapper, 72
Sample using C++ interface wrapper with dynamic library loading, 73
Sample using Generic C++ interface (multi-process compatible), 75

SAMPLE_RATE
sound4.x1.cloud-pipe.cpp, 227
sample_rate
sound4::AuxiliaryOutputInfos, 120
SOUND4_AuxiliaryOutputInfos, 214
SampleFormat
sound4, 89
sound4::cloudx1, 91
sound4::cloudx1::dyn, 99
SampleRate
sound4::cloudx1, 97
sound4::cloudx1::dyn, 107
SanityCheck
sound4::cloudx1, 95
sound4::cloudx1::dyn, 103
SaveState
sound4::CInstance, 162
sound4::cloudx1::CInstance, 174
sound4::cloudx1::dyn::CDynLib, 145
sound4::cloudx1::dyn::CInstance, 187
SetBus
sound4::CInstance, 162
SetInstanceTracing
sound4::CInstance, 162
sound4::cloudx1::CInstance, 175
sound4::cloudx1::dyn::CDynLib, 145
sound4::cloudx1::dyn::CInstance, 187
SetInstanceTracingProcessActivity
sound4::CInstance, 163
sound4::cloudx1::CInstance, 175
sound4::cloudx1::dyn::CDynLib, 145
sound4::cloudx1::dyn::CInstance, 187
SetLoggerCallback
sound4::cloudx1, 95
sound4::cloudx1::dyn, 105
sound4::cloudx1::dyn::CDynLib, 145
sound4::CProcessor, 203
SetLogSeverity
sound4::cloudx1, 95
sound4::cloudx1::dyn, 105
sound4::cloudx1::dyn::CDynLib, 145
sound4::CProcessor, 203
SetMetadata
sound4::CInstance, 163
sound4::cloudx1::CInstance, 175
sound4::cloudx1::dyn::CDynLib, 146
sound4::cloudx1::dyn::CInstance, 188
SetMetadataMulti
sound4::CInstance, 163
sound4::cloudx1::CInstance, 176
sound4::cloudx1::dyn::CDynLib, 146
sound4::cloudx1::dyn::CInstance, 188
SetParam
sound4::CInstance, 164
sound4::cloudx1::CInstance, 176
sound4::cloudx1::dyn::CInstance, 188
SetParameter
sound4::cloudx1::dyn::CDynLib, 146
SetPresetManager
sound4::CInstance, 164
sound4::cloudx1::CInstance, 176
sound4::cloudx1::dyn::CDynLib, 146
sound4::cloudx1::dyn::CInstance, 189
SetupPath
MyPresetManager, 210
SetWebServerAppHealth
sound4::CInstance, 164
sound4::cloudx1::CInstance, 177
sound4::cloudx1::dyn::CInstance, 189
shortname
SOUND4_ProcessInfo, 215
sound4, 87
_log_cb, 90
_log_cb_c, 89
change_kind_created, 89
change_kind_deleted, 89
change_kind_modified, 89
error, 88
F32_BE, 89
F32_LE, 89
F32_NATIVE, 89
fatal, 88
info, 88
INVALID_FORMAT, 89
log_cb_t, 88
LogSeverity, 88
none, 88
PresetChange_Kind, 88
S16_BE, 89
S16_LE, 89
S16_NATIVE, 89
S24_BE, 89
S24_LE, 89
S24_NATIVE, 89
S32_BE, 89
S32_LE, 89
S32_NATIVE, 89
SampleFormat, 89
verbose, 88
verbose2, 88
verbose3, 88
verbose4, 88
verbose5, 88
warning, 88
SOUND4 x1.CL Library, 1
sound4.x1.cloud-pipe-cxx.cpp, 218, 220
GetEnv, 218
main, 218
MyLogger, 219
ProcessLoop, 219
Usage, 219
sound4.x1.cloud-pipe-dyn.cpp, 222, 224
GetEnv, 222
main, 222
MyLogger, 223
ProcessLoop, 223

Usage, 223
sound4.x1.cloud-pipe.cpp, 227, 229
CHANNEL_COUNT, 227
CHUNK_SIZE, 227
main, 228
MyLogger, 228
ProcessLoop, 228
SAMPLE_RATE, 227
Usage, 228
sound4.x1.cloud.h, 237
CLOUDX1_DECL, 240
CLOUDX1_HAS_CLOUDBUS, 240
CLOUDX1_HAS_WEBSERVER, 240
TraceLoggingHProvider, 240
sound4.x1.cloud.hpp, 245
cloudx1_custom_exists, 247
cloudx1_custom_getall, 247
cloudx1_custom_getall_free, 247
cloudx1_custom_reader, 248
cloudx1_custom_reader_free, 248
cloudx1_custom_remove, 248
cloudx1_custom_rename, 248
cloudx1_custom_writer, 248
sound4.x1.cloud_dyn.hpp, 257
cloudx1_custom_exists, 260
cloudx1_custom_getall, 260
cloudx1_custom_getall_free, 260
cloudx1_custom_reader, 260
cloudx1_custom_reader_free, 260
cloudx1_custom_remove, 260
cloudx1_custom_rename, 260
cloudx1_custom_writer, 261
UNIXLIKE, 259
sound4::AuxiliaryOutputInfos, 119
buffer, 119
channel_count, 119
description, 120
id, 120
sample_rate, 120
sound4::CAudioConverter, 120
~CAudioConverter, 122
CAudioConverter, 121
Configure, 122
ConvertFrom, 122
ConvertTo, 123
Delete, 123
IsOK, 123
sound4::CBus, 130
~CBus, 131
CBus, 131
CInstance, 131
Get, 131
sound4::CInstance, 154
~CInstance, 156
AddAudio, 156
CInstance, 155
Create, 156
GetAudio, 157
GetBufferIn, 157
GetBufferOut, 157
GetChunkFrames, 158
GetEstimatedDelay, 158
GetMetadataInfos, 158
GetParam, 159
GetWebServerAppHealth, 159
IsOk, 159
NewClient, 159
OnUpdateThreadStart, 160
OnUpdateThreadStop, 160
operator=, 160
PresetManager_InformChange, 160
ProcessAnyAudio, 161
ProcessAudio, 161
ProcessAudio_Planar, 161
SaveState, 162
SetBus, 162
SetInstanceTracing, 162
SetInstanceTracingProcessActivity, 163
SetMetadata, 163
SetMetadataMulti, 163
SetParam, 164
SetPresetManager, 164
SetWebServerAppHealth, 164
StartWebServer, 165
Stop, 165
StopWebServer, 165
Terminate, 166
sound4::CInstance::CClient, 131
~CClient, 132
CClient, 132
operator=, 132
ProcessJson, 132
sound4::cloudx1, 90
_log_cb, 96
_log_cb_c, 92
ChunkFrames, 96
GetAudioInputCount, 92
GetAudioOutputCount, 92
GetBytesFromFormat, 92
GetChannelCount, 93
GetChunkSizeInFrames, 93
GetFormatFromName, 93
GetFormatName, 94
GetPossibleChunkSizeInFrames, 94
GetSampleRate, 94
GetVersion, 95
InputChannels, 96
InputSampleSize, 96
log_cb_t, 91
LogSeverity, 91
OutputChannels, 96
OutputSampleSize, 97
PresetChange_Kind, 91
process_name, 97
process_shortname, 97
SampleFormat, 91

SampleRate, 97
SanityCheck, 95
SetLoggerCallback, 95
SetLogSeverity, 95
sound4::cloudx1::CAudioConverter, 124
~CAudioConverter, 124
CAudioConverter, 124
Configure, 125
ConvertFrom, 125
ConvertTo, 126
Delete, 126
IsOK, 126
sound4::cloudx1::CInstance, 166
~CInstance, 168
ActivateAuxiliaryAudioID, 168
AddAudio, 168
CInstance, 168
Create, 168
GetAudio, 169
GetAuxiliaryAudioOutputInfos, 169
GetBufferIn, 170
GetBufferOut, 170
GetChunkFrames, 170
GetEstimatedDelay, 171
GetMetadataInfos, 171
GetParam, 171
GetWebServerAppHealth, 172
IsOk, 172
NewClient, 172
OnUpdateThreadStart, 172
OnUpdateThreadStop, 172
operator=, 173
PresetManager_InformChange, 173
ProcessAnyAudio, 173
ProcessAudio, 174
ProcessAudio_Planar, 174
SaveState, 174
SetInstanceTracing, 175
SetInstanceTracingProcessActivity, 175
SetMetadata, 175
SetMetadataMulti, 176
SetParam, 176
SetPresetManager, 176
SetWebServerAppHealth, 177
StartWebServer, 177
Stop, 177
StopWebServer, 178
Terminate, 178
sound4::cloudx1::CInstance::CClient, 133
~CClient, 134
CClient, 133
operator=, 134
ProcessJson, 134
sound4::cloudx1::CPresetLoader, 192
~CPresetLoader, 192
CPresetLoader, 192
Exists, 193
GetAll, 193
IsReadOnly, 193
Read, 193
Remove, 193
Rename, 193
Write, 193
sound4::cloudx1::dyn, 98
_log_cb, 105
_log_cb_c, 99
ChunkFrames, 105
GetAudioInputCount, 99
GetAudioOutputCount, 100
GetBytesFromFormat, 100
GetChannelCount, 100
GetChunkSizeInFrames, 101
GetFormatFromName, 101
GetFormatName, 102
GetPossibleChunkSizeInFrames, 102
GetSampleRate, 103
GetVersion, 103
InputChannels, 105
InputSampleSize, 106
log_cb_t, 99
LogSeverity, 99
OutputChannels, 106
OutputSampleSize, 106
PresetChange_Kind, 99
process_name, 106
process_shortname, 106
SampleFormat, 99
SampleRate, 107
SanityCheck, 103
SetLoggerCallback, 105
SetLogSeverity, 105
sound4::cloudx1::dyn::CAudioConverter, 126
~CAudioConverter, 127
CAudioConverter, 127
Configure, 127
ConvertFrom, 129
ConvertTo, 129
Delete, 130
IsOK, 130
sound4::cloudx1::dyn::CDynLib, 136
AddAudio, 138
AddPadAudio, 138
AudioConverter_From, 139
AudioConverter_To, 139
AudioConvertFrom, 139
AudioConvertTo, 139
AudioMonoFromLiveStereo, 139
AudioMonoToLiveStereo, 139
CDynLib, 138
DeleteClient, 139
ExitProcess, 140
FreeAudioConverter, 140
FreeJsonAnswer, 140
FreeMetadataInfos, 140
FreeParameters, 140
FreeParameterValue, 140

GetAudio, 140
GetAudioConverter, 141
GetAudioInputCount, 141
GetAudioOutputCount, 141
GetBufferIn, 141
GetBufferOut, 141
GetBytesFromFormat, 141
GetChannelCount, 141
GetChunkSizeInFrames, 142
GetEstimatedDelay, 142
GetFormatFromName, 142
GetFormatName, 142
GetMaxPacketFrame, 142
GetMetadataInfos, 142
GetOutputCount, 142
GetParameter, 143
GetPossibleChunkSizeInFrames, 143
GetProcessChunkFrames, 143
GetSampleRate, 143
GetVersion, 143
InitProcess, 143
InitProcess2, 143
InitProcess3, 144
IsOk, 138
Load, 138
MonoToStereo, 144
MonoToStereo_Planar, 144
NewClient, 144
NewParameters, 144
PresetManager_InformChange, 144
ProcessAudio, 144
ProcessAudio_Planar, 145
ProcessJson, 145
SaveState, 145
SetInstanceTracing, 145
SetInstanceTracingProcessActivity, 145
SetLoggerCallback, 145
SetLogSeverity, 145
SetMetadata, 146
SetMetadataMulti, 146
SetParameter, 146
SetPresetManager, 146
StartUpdateThread, 146
StereoToMono, 146
StereoToMono_Planar, 146
StopUpdateThread, 147
TerminateProcess, 147
WaitUpdateThreadReady, 147
Webserver, 147
Webserver_FreeString, 147
Webserver_GetAppHealth, 147
Webserver_SetAppHealth, 147
Webserver_Status, 148
Webserver_Stop, 148
Webserver_tcp, 148
Webserver_tcp2, 148
sound4::cloudx1::dyn::CInstance, 178
~CInstance, 180
AddAudio, 180
CInstance, 180
Create, 180
GetAudio, 181
GetBufferIn, 181
GetBufferOut, 182
GetChunkFrames, 182
GetEstimatedDelay, 182
GetMetadataInfos, 183
GetParam, 183
GetWebServerAppHealth, 183
IsOk, 184
NewClient, 184
OnUpdateThreadStart, 184
OnUpdateThreadStop, 184
operator=, 184
PresetManager_InformChange, 184
ProcessAnyAudio, 185
ProcessAudio, 185
ProcessAudio_Planar, 185
SaveState, 187
SetInstanceTracing, 187
SetInstanceTracingProcessActivity, 187
SetMetadata, 188
SetMetadataMulti, 188
SetParam, 188
SetPresetManager, 189
SetWebServerAppHealth, 189
StartWebServer, 189
Stop, 190
StopWebServer, 190
Terminate, 190
sound4::cloudx1::dyn::CInstance::CClient, 134
~CClient, 135
CClient, 135
operator=, 135
ProcessJson, 135
sound4::cloudx1::dyn::CPresetLoader, 194
~CPresetLoader, 194
CPresetLoader, 194
Exists, 194
GetAll, 194
IsReadOnly, 195
Read, 195
Remove, 195
Rename, 195
Write, 195
sound4::cloudx1::dyn::helper, 107
AudioConvertFrom, 108
AudioConvertTo, 108
AudioMonoFromLiveStereo, 109
AudioMonoToLiveStereo, 109
MonoToStereo, 109
MonoToStereo_Planar, 110
StereoToMono, 110
StereoToMono_Planar, 111
UTF8ToWString, 111
WStringToUTF8, 112

sound4::cloudx1::dyn::helper::CDynLoader, 148
~CDynLoader, 149
CDynLoader, 149
Close, 150
GetSymbol, 150
GetSymbol_safe, 150
GetThisLibraryPath, 150
IsOk, 150
Load, 150
operator bool, 150
operator=, 151
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 204
DynFuncHolder, 204
IsOk, 205
operator bool, 205
operator T*, 205
sound4::cloudx1::dyn::helper::SampleFormat< T >, 211
sound4::cloudx1::helper, 112
AudioConvertFrom, 113
AudioConvertTo, 113
AudioMonoFromLiveStereo, 113
AudioMonoToLiveStereo, 114
MonoToStereo, 114
MonoToStereo_Planar, 115
StereoToMono, 115
StereoToMono_Planar, 115
UTF8ToWString, 116
WStringToUTF8, 116
sound4::cloudx1::helper::SampleFormat< T >, 211
sound4::CPresetLoader, 196
~CPresetLoader, 196
CPresetLoader, 196
Exists, 196
GetAll, 196
IsReadOnly, 197
Read, 197
Remove, 197
Rename, 197
Write, 197
sound4::CProcessor, 198
CAudioConverter, 203
CBus, 203
CInstance, 203
CProcessor, 199
GetAudioInputCount, 199
GetAudioOutputCount, 199
GetBytesFromFormat, 199
GetChannelCount, 200
GetChunkSizeInFrames, 200
GetFormatFromName, 200
GetFormatName, 201
GetPossibleChunkSizeInFrames, 201
GetPrefixSymbol, 201
GetSampleRate, 202
GetVersion, 202
IsOk, 202
Load, 202
SetLoggerCallback, 203
SetLogSeverity, 203
sound4::helper, 117
UTF8ToWString, 117
WStringToUTF8, 117
sound4::helper::CDynLoader, 151
~CDynLoader, 152
CDynLoader, 152
Close, 152
GetSymbol, 152
GetSymbol_safe, 152
GetThisLibraryPath, 153
IsOk, 153
Load, 153
operator bool, 153
operator=, 153
sound4::helper::DynFuncHolder< T >, 205
DynFuncHolder, 206
IsOk, 206
operator bool, 206
operator T*, 206
sound4::helper::SampleFormat< T >, 213
SOUND4_AuxiliaryOutputInfos, 213
buffer, 214
channel_count, 214
description, 214
id, 214
sample_rate, 214
SOUND4_GetProcessInfo
Global setup, 63
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 293
Web server, 80
SOUND4_ProcessInfo, 214
name, 215
prefix, 215
shortname, 215
struct_size, 215
version, 215
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 293
Web server, 80
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 293
Web server, 80
sound4cl-pipe.cpp, 232, 234
GetEnv, 232
main, 232
ProcessLoop, 233
sound4cl.hpp, 276
UNIXLIKE, 277
sound4cl_cdef.h, 292
fs_char, 293
SOUND4_INVALID_WEBSERVER_ID, 293
SOUND4_WEBSERVER_HTTP_OK, 293
SOUND4_WEBSERVER_HTTPS_OK, 293
TraceLoggingHProvider, 293

StartUpdateThread
sound4::cloudx1::dyn::CDynLib, 146
StartWebServer
sound4::CInstance, 165
sound4::cloudx1::CInstance, 177
sound4::cloudx1::dyn::CInstance, 189
StereoToMono
Audio convert helpers (C++), 38
sound4::cloudx1::dyn::CDynLib, 146
sound4::cloudx1::dyn::helper, 110
sound4::cloudx1::helper, 115
StereoToMono_Planar
Audio convert helpers (C++), 39
sound4::cloudx1::dyn::CDynLib, 146
sound4::cloudx1::dyn::helper, 111
sound4::cloudx1::helper, 115
Stop
sound4::CInstance, 165
sound4::cloudx1::CInstance, 177
sound4::cloudx1::dyn::CInstance, 190
StopUpdateThread
sound4::cloudx1::dyn::CDynLib, 147
StopWebServer
sound4::CInstance, 165
sound4::cloudx1::CInstance, 178
sound4::cloudx1::dyn::CInstance, 190
struct_size
SOUND4_ProcessInfo, 215

Terminate
sound4::CInstance, 166
sound4::cloudx1::CInstance, 178
sound4::cloudx1::dyn::CInstance, 190
TerminateProcess
sound4::cloudx1::dyn::CDynLib, 147
TraceLoggingHProvider
sound4.x1.cloud.h, 240
sound4cl_cdef.h, 293

UNIXLIKE
sound4.x1.cloud_dyn.hpp, 259
sound4cl.hpp, 277
Update Management, 78
cloudx1_StartUpdateThread, 78
cloudx1_StopUpdateThread, 78
cloudx1_WaitUpdateThreadReady, 79
Usage
sound4.x1.cloud-pipe-cxx.cpp, 219
sound4.x1.cloud-pipe-dyn.cpp, 223
sound4.x1.cloud-pipe.cpp, 228
UTF8ToWString
sound4::cloudx1::dyn::helper, 111
sound4::cloudx1::helper, 116
sound4::helper, 117

verbose
Global setup, 60
sound4, 88
verbose2
Global setup, 60
sound4, 88
verbose3
Global setup, 60
sound4, 88
verbose4
Global setup, 60
sound4, 88
verbose5
Global setup, 60
sound4, 88
version
SOUND4_ProcessInfo, 215

WaitUpdateThreadReady
sound4::cloudx1::dyn::CDynLib, 147
warning
Global setup, 60
sound4, 88
Web server, 79
cloudx1_Webserver, 80
cloudx1_Webserver_FreeString, 81
cloudx1_Webserver_GetAppHealth, 81
cloudx1_Webserver_SetAppHealth, 81
cloudx1_Webserver_Status, 83
cloudx1_Webserver_Stop, 83
cloudx1_Webserver_tcp, 84
cloudx1_Webserver_tcp2, 84
SOUND4_INVALID_WEBSERVER_ID, 80
SOUND4_WEBSERVER_HTTP_OK, 80
SOUND4_WEBSERVER_HTTPS_OK, 80
Webserver
sound4::cloudx1::dyn::CDynLib, 147
Webserver_FreeString
sound4::cloudx1::dyn::CDynLib, 147
Webserver_GetAppHealth
sound4::cloudx1::dyn::CDynLib, 147
Webserver_SetAppHealth
sound4::cloudx1::dyn::CDynLib, 147
Webserver_Status
sound4::cloudx1::dyn::CDynLib, 148
Webserver_Stop
sound4::cloudx1::dyn::CDynLib, 148
Webserver_tcp
sound4::cloudx1::dyn::CDynLib, 148
Webserver_tcp2
sound4::cloudx1::dyn::CDynLib, 148
Write
MyPresetManager, 210
sound4::cloudx1::CPresetLoader, 193
sound4::cloudx1::dyn::CPresetLoader, 195
sound4::CPresetLoader, 197
WStringToUTF8
sound4::cloudx1::dyn::helper, 112
sound4::cloudx1::helper, 116
sound4::helper, 117