SOUND4 x1.CLOUD Library [1.5.0]

Generated by Doxygen 1.9.1

1 SOUND4 x1.CLOUD Library 1
1.1 General usage . . . 1
1.2 Examples . . . 1

2 Deprecated List 3

3 Module Index 5
3.1 Modules . . . 5

4 Namespace Index 7
4.1 Namespace List . . . 7

5 Hierarchical Index 9
5.1 Class Hierarchy . . . 9

6 Class Index 13
6.1 Class List . . . 13

7 File Index 15
7.1 File List . . . 15

8 Module Documentation 17
8.1 Sample using C interface . . . 17
8.1.1 Sample to create the processing instance . . . 17
8.1.2 Sample processing loop . . . 18
8.1.3 Sample to stop the processing instance . . . 18
8.2 Environment variables and parameters . . . 19
8.2.1 Parameters defaulting from environment . . . 19
8.2.1.1 Webserver authentification setup : . . . 19
8.2.1.2 Webserver secure : . . . 19
8.2.1.3 Network access : . . . 19
8.2.2 Parameters . . . 19
8.2.3 Preset sharing . . . 20
8.3 Global setup . . . 20
8.3.1 Detailed Description . . . 21
8.3.2 Macro Definition Documentation . . . 21
8.3.2.1 CLOUDX1_AUDIO_INPUT_COUNT . . . 21
8.3.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT . . . 21
8.3.2.3 CLOUDX1_AUDIOFRAME_COUNT . . . 21
8.3.2.4 CLOUDX1_CHANNEL_COUNT . . . 22
8.3.2.5 CLOUDX1_SAMPLE_RATE . . . 22
8.3.3 Typedef Documentation . . . 22
8.3.3.1 cloudx1_loggerfn . . . 22
8.3.4 Enumeration Type Documentation . . . 22
8.3.4.1 cloudx1_LogSeverity . . . 22

8.3.5 Function Documentation . . . 23
8.3.5.1 cloudx1_GetAudioInputCount() . . . 23
8.3.5.2 cloudx1_GetAudioOutputCount() . . . 23
8.3.5.3 cloudx1_GetChannelCount() . . . 24
8.3.5.4 cloudx1_GetChunkSizeInFrames() . . . 24
8.3.5.5 cloudx1_GetPossibleChunkSizeInFrames() . . . 24
8.3.5.6 cloudx1_GetSampleRate() . . . 25
8.3.5.7 cloudx1_GetVersion() . . . 25
8.3.5.8 cloudx1_SetLoggerCallback() . . . 25
8.3.5.9 cloudx1_SetLogSeverity() . . . 26
8.3.5.10 SOUND4_GetProcessInfo() . . . 26
8.4 Process creation/destruction . . . 26
8.4.1 Detailed Description . . . 27
8.4.2 Function Documentation . . . 27
8.4.2.1 cloudx1_ExitProcess() . . . 27
8.4.2.2 cloudx1_FreeParameters() . . . 27
8.4.2.3 cloudx1_FreeParameterValue() . . . 27
8.4.2.4 cloudx1_GetParameter() . . . 27
8.4.2.5 cloudx1_GetProcessChunkFrames() . . . 28
8.4.2.6 cloudx1_InitProcess() . . . 28
8.4.2.7 cloudx1_InitProcess2() . . . 29
8.4.2.8 cloudx1_InitProcess3() . . . 29
8.4.2.9 cloudx1_NewParameters() . . . 30
8.4.2.10 cloudx1_SetParameter() . . . 31
8.4.2.11 cloudx1_TerminateProcess() . . . 31
8.5 Custom preset managagement . . . 31
8.5.1 Detailed Description . . . 32
8.5.2 Typedef Documentation . . . 32
8.5.2.1 cloudx1_storage_exists . . . 32
8.5.2.2 cloudx1_storage_getall . . . 33
8.5.2.3 cloudx1_storage_getall_free . . . 33
8.5.2.4 cloudx1_storage_reader . . . 33
8.5.2.5 cloudx1_storage_reader_free . . . 34
8.5.2.6 cloudx1_storage_remove . . . 34
8.5.2.7 cloudx1_storage_rename . . . 35
8.5.2.8 cloudx1_storage_writer . . . 35
8.5.2.9 fs_char . . . 35
8.5.3 Enumeration Type Documentation . . . 35
8.5.3.1 cloudx1_PresetChange_Kind . . . 36
8.5.4 Function Documentation . . . 36
8.5.4.1 cloudx1_PresetManager_InformChange() . . . 36
8.5.4.2 cloudx1_SetPresetManager() . . . 36

8.6 Metadata Management . . . 37
8.6.1 Detailed Description . . . 37
8.6.2 Function Documentation . . . 37
8.6.2.1 cloudx1_FreeMetadataInfos() . . . 37
8.6.2.2 cloudx1_GetMetadataInfos() . . . 38
8.6.2.3 cloudx1_SetMetadata() . . . 38
8.6.2.4 cloudx1_SetMetadataMulti() . . . 39
8.7 Update Management . . . 39
8.7.1 Detailed Description . . . 39
8.7.2 Function Documentation . . . 39
8.7.2.1 cloudx1_StartUpdateThread() . . . 39
8.7.2.2 cloudx1_StopUpdateThread() . . . 40
8.7.2.3 cloudx1_WaitUpdateThreadReady() . . . 40
8.8 Direct processing . . . 40
8.8.1 Detailed Description . . . 41
8.8.2 Function Documentation . . . 41
8.8.2.1 cloudx1_GetBufferIn() . . . 41
8.8.2.2 cloudx1_GetBufferOut() . . . 41
8.8.2.3 cloudx1_GetEstimatedDelay() . . . 42
8.8.2.4 cloudx1_ProcessAudio() . . . 42
8.8.2.5 cloudx1_ProcessAudio_Planar() . . . 43
8.9 Audio managing (any audio) . . . 43
8.9.1 Detailed Description . . . 44
8.9.2 Enumeration Type Documentation . . . 44
8.9.2.1 cloudx1_SampleFormat . . . 44
8.9.3 Function Documentation . . . 45
8.9.3.1 cloudx1_AddAudio() . . . 45
8.9.3.2 cloudx1_AddPadAudio() . . . 45
8.9.3.3 cloudx1_GetAudio() . . . 45
8.9.3.4 cloudx1_GetBytesFromFormat() . . . 46
8.9.3.5 cloudx1_GetFormatFromName() . . . 46
8.9.3.6 cloudx1_GetFormatName() . . . 47
8.9.3.7 cloudx1_GetMaxPacketFrame() . . . 47
8.9.3.8 cloudx1_GetOutputCount() . . . 47
8.10 Audio convert helpers . . . 48
8.10.1 Detailed Description . . . 48
8.10.2 Function Documentation . . . 48
8.10.2.1 cloudx1_AudioConvertFrom() . . . 48
8.10.2.2 cloudx1_AudioConvertTo() . . . 49
8.10.2.3 cloudx1_AudioMonoFromLiveStereo() . . . 49
8.10.2.4 cloudx1_AudioMonoToLiveStereo() . . . 49
8.10.2.5 cloudx1_MonoToStereo() . . . 50

8.10.2.6 cloudx1_MonoToStereo_Planar() . . . 50
8.10.2.7 cloudx1_StereoToMono() . . . 51
8.10.2.8 cloudx1_StereoToMono_Planar() . . . 51
8.11 Client's JSon config interfacing . . . 52
8.11.1 Detailed Description . . . 52
8.11.2 Function Documentation . . . 52
8.11.2.1 cloudx1_DeleteClient() . . . 52
8.11.2.2 cloudx1_FreeJsonAnswer() . . . 52
8.11.2.3 cloudx1_NewClient() . . . 53
8.11.2.4 cloudx1_ProcessJson() . . . 53
8.11.2.5 cloudx1_SaveState() . . . 54
8.12 JSON syntax . . . 54
8.12.1 Commands . . . 54
8.12.2 Answers . . . 55
8.12.3 Examples . . . 56
8.12.3.1 Get status . . . 56
8.12.3.2 Get all changes since last request . . . 56
8.12.3.3 Check license . . . 56
8.12.3.4 Set Processor name . . . 56
8.12.3.5 Get presetlist . . . 56
8.12.3.6 Load a preset . . . 56
8.12.3.7 Get information on preset on air . . . 56
8.12.3.8 Set input gain . . . 57
8.12.3.9 Set bypass . . . 57
8.12.3.10 Export a preset . . . 57
8.12.3.11 Import a preset . . . 57
8.13 JSON syntax for internal setup . . . 57
8.13.1 Get process and modules information . . . 58
8.13.2 Authentication management . . . 58
8.13.2.1 Difference between setup and admin account . . . 58
8.13.2.2 Access encoding . . . 58
8.13.2.3 Change setup account . . . 58
8.13.2.4 Get user info . . . 59
8.13.2.5 Add or modify a user . . . 59
8.13.2.6 Removes a user . . . 59
8.13.2.7 List users . . . 59
8.13.2.8 Get/set unlogged access . . . 59
8.13.2.9 Get realm . . . 59
8.13.2.10 Get API version . . . 60
8.13.2.11 Get setup is admin . . . 60
8.13.2.12 Get/set setup account . . . 60
8.13.2.13 Get/set custom storage . . . 60

8.13.3 Get process' interface description . . . 60
8.13.3.1 Get XML decription . . . 60
8.13.3.2 Get Process Image . . . 60
8.13.3.3 Get Process Ico . . . 61
8.14 Web server . . . 61
8.14.1 Detailed Description . . . 61
8.14.2 Macro Definition Documentation . . . 61
8.14.2.1 SOUND4_INVALID_WEBSERVER_ID . . . 61
8.14.2.2 SOUND4_WEBSERVER_HTTP_OK . . . 62
8.14.2.3 SOUND4_WEBSERVER_HTTPS_OK . . . 62
8.14.3 Function Documentation . . . 62
8.14.3.1 cloudx1_Webserver() . . . 62
8.14.3.2 cloudx1_Webserver_FreeString() . . . 63
8.14.3.3 cloudx1_Webserver_GetAppHealth() . . . 63
8.14.3.4 cloudx1_Webserver_SetAppHealth() . . . 63
8.14.3.5 cloudx1_Webserver_Status() . . . 64
8.14.3.6 cloudx1_Webserver_Stop() . . . 64
8.14.3.7 cloudx1_Webserver_tcp() . . . 65
8.14.3.8 cloudx1_Webserver_tcp2() . . . 65
8.15 Sample using C++ interface wrapper . . . 66
8.15.1 Setup and sanity checks . . . 66
8.15.2 Sample to create the processing instance . . . 66
8.15.3 Communicating with the instance . . . 66
8.15.4 Starting a web-server . . . 67
8.15.5 Sample processing loop . . . 67
8.16 Sample using C++ interface wrapper with dynamic library loading . . . 67
8.16.1 Dynamic loading . . . 67
8.16.2 Setup and sanity checks . . . 67
8.16.3 Sample to create the processing instance . . . 67
8.16.4 Communicating with the instance . . . 67
8.16.5 Starting a web-server . . . 68
8.16.6 Sample processing loop . . . 68
8.16.7 Custom preset manager . . . 68
8.16.8 Custom preset manager loading . . . 69
8.17 Sample using Generic C++ interface (multi-process compatible) . . . 69
8.17.1 Loading the library . . . 69
8.17.2 Setup and configure . . . 69
8.17.3 Create the processing instance . . . 69
8.17.4 Processing loop . . . 70
8.17.5 Starting a web-server . . . 70
8.17.6 Communicating with the instance . . . 70
8.17.7 Custom preset manager example . . . 70

8.17.8 Custom preset manager loading . . . 71

9 Namespace Documentation **73**

9.1 dyn Namespace Reference . . . 73

9.1.1 Detailed Description . . . 73

9.2 sound4 Namespace Reference . . . 73

9.2.1 Detailed Description . . . 74

9.2.2 Typedef Documentation . . . 74

9.2.2.1 log_cb_t . . . 74

9.2.2.2 LogSeverity . . . 74

9.2.2.3 PresetChange_Kind . . . 74

9.2.3 Variable Documentation . . . 75

9.2.3.1 error . . . 75

9.2.3.2 fatal . . . 75

9.2.3.3 info . . . 75

9.2.3.4 none . . . 75

9.2.3.5 verbose . . . 75

9.2.3.6 verbose2 . . . 76

9.2.3.7 verbose3 . . . 76

9.2.3.8 verbose4 . . . 76

9.2.3.9 verbose5 . . . 76

9.2.3.10 warning . . . 76

9.3 sound4::cloudx1 Namespace Reference . . . 76

9.3.1 Detailed Description . . . 77

9.3.2 Typedef Documentation . . . 77

9.3.2.1 log_cb_t . . . 77

9.3.2.2 LogSeverity . . . 77

9.3.2.3 PresetChange_Kind . . . 78

9.3.2.4 SampleFormat . . . 78

9.3.3 Variable Documentation . . . 78

9.3.3.1 ChunkFrames . . . 78

9.3.3.2 InputChannels . . . 78

9.3.3.3 InputSampleSize . . . 78

9.3.3.4 OutputChannels . . . 79

9.3.3.5 OutputSampleSize . . . 79

9.3.3.6 SampleRate . . . 79

9.4 sound4::cloudx1::dyn Namespace Reference . . . 79

9.4.1 Typedef Documentation . . . 80

9.4.1.1 log_cb_t . . . 80

9.4.1.2 LogSeverity . . . 80

9.4.1.3 PresetChange_Kind . . . 80

9.4.1.4 SampleFormat . . . 80

9.4.2 Variable Documentation . . . 81
9.4.2.1 ChunkFrames . . . 81
9.4.2.2 InputChannels . . . 81
9.4.2.3 InputSampleSize . . . 81
9.4.2.4 OutputChannels . . . 81
9.4.2.5 OutputSampleSize . . . 81
9.4.2.6 SampleRate . . . 82
9.5 sound4::cloudx1::dyn::helper Namespace Reference . . . 82
9.5.1 Detailed Description . . . 82
9.5.2 Typedef Documentation . . . 82
9.5.2.1 DynFunc_t . . . 82
9.5.2.2 DynLib_t . . . 83
9.6 sound4::cloudx1::helper Namespace Reference . . . 83
9.6.1 Detailed Description . . . 83
9.7 sound4::helper Namespace Reference . . . 83
9.7.1 Detailed Description . . . 83
9.7.2 Typedef Documentation . . . 84
9.7.2.1 DynFunc_t . . . 84
9.7.2.2 DynLib_t . . . 84
9.8 std Namespace Reference . . . 84

10 Class Documentation **85**
10.1 sound4::CBus Class Reference . . . 85
10.1.1 Detailed Description . . . 85
10.1.2 Constructor & Destructor Documentation . . . 85
10.1.2.1 CBus() . . . 85
10.1.2.2 ~CBus() . . . 86
10.1.3 Member Function Documentation . . . 86
10.1.3.1 Get() . . . 86
10.2 sound4::CInstance::CClient Class Reference . . . 86
10.2.1 Detailed Description . . . 86
10.2.2 Constructor & Destructor Documentation . . . 87
10.2.2.1 CClient() [1/2] . . . 87
10.2.2.2 CClient() [2/2] . . . 87
10.2.2.3 ~CClient() . . . 87
10.2.3 Member Function Documentation . . . 87
10.2.3.1 operator=() . . . 87
10.2.3.2 ProcessJson() . . . 87
10.3 sound4::cloudx1::CInstance::CClient Class Reference . . . 88
10.3.1 Detailed Description . . . 88
10.3.2 Constructor & Destructor Documentation . . . 89
10.3.2.1 CClient() [1/2] . . . 89

10.3.2.2 CClient() [2/2] . . . 89
10.3.2.3 ~CClient() . . . 89
10.3.3 Member Function Documentation . . . 89
10.3.3.1 operator=() . . . 89
10.3.3.2 ProcessJson() . . . 89
10.4 sound4::cloudx1::dyn::CInstance::CClient Class Reference . . . 90
10.4.1 Detailed Description . . . 90
10.4.2 Constructor & Destructor Documentation . . . 90
10.4.2.1 CClient() [1/2] . . . 91
10.4.2.2 CClient() [2/2] . . . 91
10.4.2.3 ~CClient() . . . 91
10.4.3 Member Function Documentation . . . 91
10.4.3.1 operator=() . . . 91
10.4.3.2 ProcessJson() . . . 91
10.5 sound4::cloudx1::dyn::CDynLib Class Reference . . . 92
10.5.1 Detailed Description . . . 93
10.5.2 Constructor & Destructor Documentation . . . 94
10.5.2.1 CDynLib() . . . 94
10.5.3 Member Function Documentation . . . 94
10.5.3.1 IsOk() . . . 94
10.5.3.2 Load() . . . 94
10.5.4 Member Data Documentation . . . 94
10.5.4.1 AddAudio . . . 95
10.5.4.2 AddPadAudio . . . 95
10.5.4.3 AudioConvertFrom . . . 95
10.5.4.4 AudioConvertTo . . . 95
10.5.4.5 AudioMonoFromLiveStereo . . . 95
10.5.4.6 AudioMonoToLiveStereo . . . 95
10.5.4.7 DeleteClient . . . 96
10.5.4.8 ExitProcess . . . 96
10.5.4.9 FreeJsonAnswer . . . 96
10.5.4.10 FreeMetadataInfos . . . 96
10.5.4.11 FreeParameters . . . 96
10.5.4.12 FreeParameterValue . . . 96
10.5.4.13 GetAudio . . . 97
10.5.4.14 GetAudioInputCount . . . 97
10.5.4.15 GetAudioOutputCount . . . 97
10.5.4.16 GetBufferIn . . . 97
10.5.4.17 GetBufferOut . . . 97
10.5.4.18 GetBytesFromFormat . . . 97
10.5.4.19 GetChannelCount . . . 98
10.5.4.20 GetChunkSizeInFrames . . . 98

10.5.4.21 GetEstimatedDelay . . . 98
10.5.4.22 GetFormatFromName . . . 98
10.5.4.23 GetFormatName . . . 98
10.5.4.24 GetMaxPacketFrame . . . 98
10.5.4.25 GetMetadataInfos . . . 99
10.5.4.26 GetOutputCount . . . 99
10.5.4.27 GetParameter . . . 99
10.5.4.28 GetPossibleChunkSizeInFrames . . . 99
10.5.4.29 GetProcessChunkFrames . . . 99
10.5.4.30 GetSampleRate . . . 99
10.5.4.31 GetVersion . . . 100
10.5.4.32 InitProcess . . . 100
10.5.4.33 InitProcess2 . . . 100
10.5.4.34 InitProcess3 . . . 100
10.5.4.35 MonoToStereo . . . 100
10.5.4.36 MonoToStereo_Planar . . . 100
10.5.4.37 NewClient . . . 101
10.5.4.38 NewParameters . . . 101
10.5.4.39 PresetManager_InformChange . . . 101
10.5.4.40 ProcessAudio . . . 101
10.5.4.41 ProcessAudio_Planar . . . 101
10.5.4.42 ProcessJson . . . 101
10.5.4.43 SaveState . . . 102
10.5.4.44 SetLoggerCallback . . . 102
10.5.4.45 SetLogSeverity . . . 102
10.5.4.46 SetMetadata . . . 102
10.5.4.47 SetMetadataMulti . . . 102
10.5.4.48 SetParameter . . . 102
10.5.4.49 SetPresetManager . . . 103
10.5.4.50 StartUpdateThread . . . 103
10.5.4.51 StereoToMono . . . 103
10.5.4.52 StereoToMono_Planar . . . 103
10.5.4.53 StopUpdateThread . . . 103
10.5.4.54 TerminateProcess . . . 103
10.5.4.55 WaitUpdateThreadReady . . . 104
10.5.4.56 Webserver . . . 104
10.5.4.57 Webserver_FreeString . . . 104
10.5.4.58 Webserver_GetAppHealth . . . 104
10.5.4.59 Webserver_SetAppHealth . . . 104
10.5.4.60 Webserver_Status . . . 104
10.5.4.61 Webserver_Stop . . . 105
10.5.4.62 Webserver_tcp . . . 105

10.5.4.63 Webserver_tcp2 . . . 105
10.6 sound4::cloudx1::dyn::helper::CDynLoader Class Reference . . . 105
10.6.1 Detailed Description . . . 106
10.6.2 Constructor & Destructor Documentation . . . 106
10.6.2.1 CDynLoader() [1/3] . . . 106
10.6.2.2 CDynLoader() [2/3] . . . 106
10.6.2.3 CDynLoader() [3/3] . . . 106
10.6.2.4 ~CDynLoader() . . . 106
10.6.3 Member Function Documentation . . . 106
10.6.3.1 Close() . . . 107
10.6.3.2 GetSymbol() . . . 107
10.6.3.3 GetSymbol_safe() . . . 107
10.6.3.4 GetThisLibraryPath() . . . 107
10.6.3.5 IsOk() . . . 107
10.6.3.6 Load() . . . 108
10.6.3.7 operator bool() . . . 108
10.6.3.8 operator=() . . . 108
10.7 sound4::helper::CDynLoader Class Reference . . . 108
10.7.1 Detailed Description . . . 109
10.7.2 Constructor & Destructor Documentation . . . 109
10.7.2.1 CDynLoader() [1/3] . . . 109
10.7.2.2 CDynLoader() [2/3] . . . 109
10.7.2.3 CDynLoader() [3/3] . . . 109
10.7.2.4 ~CDynLoader() . . . 109
10.7.3 Member Function Documentation . . . 109
10.7.3.1 Close() . . . 110
10.7.3.2 GetSymbol() . . . 110
10.7.3.3 GetSymbol_safe() . . . 110
10.7.3.4 GetThisLibraryPath() . . . 110
10.7.3.5 IsOk() . . . 110
10.7.3.6 Load() . . . 111
10.7.3.7 operator bool() . . . 111
10.7.3.8 operator=() . . . 111
10.8 sound4::CInstance Class Reference . . . 111
10.8.1 Detailed Description . . . 112
10.8.2 Constructor & Destructor Documentation . . . 113
10.8.2.1 CInstance() [1/2] . . . 113
10.8.2.2 CInstance() [2/2] . . . 113
10.8.2.3 ~CInstance() . . . 113
10.8.3 Member Function Documentation . . . 113
10.8.3.1 AddAudio() . . . 113
10.8.3.2 Create() . . . 113

10.8.3.3 GetAudio() . . . 114
10.8.3.4 GetBufferIn() . . . 114
10.8.3.5 GetBufferOut() . . . 115
10.8.3.6 GetChunkFrames() . . . 115
10.8.3.7 GetEstimatedDelay() . . . 115
10.8.3.8 GetMetadataInfos() . . . 116
10.8.3.9 GetParam() . . . 116
10.8.3.10 GetWebServerAppHealth() . . . 116
10.8.3.11 IsOk() . . . 116
10.8.3.12 NewClient() . . . 117
10.8.3.13 OnUpdateThreadStart() . . . 117
10.8.3.14 OnUpdateThreadStop() . . . 117
10.8.3.15 operator=() . . . 117
10.8.3.16 PresetManager_InformChange() . . . 117
10.8.3.17 ProcessAnyAudio() . . . 118
10.8.3.18 ProcessAudio() . . . 118
10.8.3.19 ProcessAudio_Planar() . . . 119
10.8.3.20 SetBus() . . . 119
10.8.3.21 SetMetadata() . . . 119
10.8.3.22 SetMetadataMulti() . . . 120
10.8.3.23 SetParam() . . . 120
10.8.3.24 SetPresetManager() . . . 121
10.8.3.25 SetWebServerAppHealth() . . . 121
10.8.3.26 StartWebServer() . . . 121
10.8.3.27 Stop() . . . 122
10.8.3.28 StopWebServer() . . . 122
10.8.3.29 Terminate() . . . 122
10.9 sound4::cloudx1::CInstance Class Reference . . . 122
10.9.1 Detailed Description . . . 123
10.9.2 Constructor & Destructor Documentation . . . 124
10.9.2.1 CInstance() [1/2] . . . 124
10.9.2.2 CInstance() [2/2] . . . 124
10.9.2.3 ~CInstance() . . . 124
10.9.3 Member Function Documentation . . . 124
10.9.3.1 AddAudio() . . . 124
10.9.3.2 Create() . . . 124
10.9.3.3 GetAudio() . . . 125
10.9.3.4 GetBufferIn() . . . 125
10.9.3.5 GetBufferOut() . . . 126
10.9.3.6 GetChunkFrames() . . . 126
10.9.3.7 GetEstimatedDelay() . . . 126
10.9.3.8 GetMetadataInfos() . . . 127

10.9.3.9 GetParam() . . . 127
10.9.3.10 GetWebServerAppHealth() . . . 127
10.9.3.11 IsOk() . . . 127
10.9.3.12 NewClient() . . . 128
10.9.3.13 OnUpdateThreadStart() . . . 128
10.9.3.14 OnUpdateThreadStop() . . . 128
10.9.3.15 operator=() . . . 128
10.9.3.16 PresetManager_InformChange() . . . 128
10.9.3.17 ProcessAnyAudio() . . . 129
10.9.3.18 ProcessAudio() . . . 129
10.9.3.19 ProcessAudio_Planar() . . . 130
10.9.3.20 SetMetadata() . . . 130
10.9.3.21 SetMetadataMulti() . . . 130
10.9.3.22 SetParam() . . . 131
10.9.3.23 SetPresetManager() . . . 131
10.9.3.24 SetWebServerAppHealth() . . . 132
10.9.3.25 StartWebServer() . . . 132
10.9.3.26 Stop() . . . 132
10.9.3.27 StopWebServer() . . . 133
10.9.3.28 Terminate() . . . 133
10.10 sound4::cloudx1::dyn::CInstance Class Reference . . . 133
10.10.1 Detailed Description . . . 134
10.10.2 Constructor & Destructor Documentation . . . 134
10.10.2.1 CInstance() [1/2] . . . 135
10.10.2.2 CInstance() [2/2] . . . 135
10.10.2.3 ~CInstance() . . . 135
10.10.3 Member Function Documentation . . . 135
10.10.3.1 AddAudio() . . . 135
10.10.3.2 Create() . . . 135
10.10.3.3 GetAudio() . . . 136
10.10.3.4 GetBufferIn() . . . 136
10.10.3.5 GetBufferOut() . . . 137
10.10.3.6 GetChunkFrames() . . . 137
10.10.3.7 GetEstimatedDelay() . . . 137
10.10.3.8 GetMetadataInfos() . . . 138
10.10.3.9 GetParam() . . . 138
10.10.3.10 GetWebServerAppHealth() . . . 138
10.10.3.11 IsOk() . . . 138
10.10.3.12 NewClient() . . . 139
10.10.3.13 OnUpdateThreadStart() . . . 139
10.10.3.14 OnUpdateThreadStop() . . . 139
10.10.3.15 operator=() . . . 139

10.10.3.16 PresetManager_InformChange() . . . 139
10.10.3.17 ProcessAnyAudio() . . . 140
10.10.3.18 ProcessAudio() . . . 140
10.10.3.19 ProcessAudio_Planar() . . . 141
10.10.3.20 SetMetadata() . . . 141
10.10.3.21 SetMetadataMulti() . . . 141
10.10.3.22 SetParam() . . . 142
10.10.3.23 SetPresetManager() . . . 142
10.10.3.24 SetWebServerAppHealth() . . . 143
10.10.3.25 StartWebServer() . . . 143
10.10.3.26 Stop() . . . 143
10.10.3.27 StopWebServer() . . . 144
10.10.3.28 Terminate() . . . 144
10.11 cloudx1_CClientInstance Struct Reference . . . 144
10.11.1 Detailed Description . . . 144
10.12 cloudx1_CInstance Struct Reference . . . 144
10.12.1 Detailed Description . . . 145
10.13 cloudx1_CParameters Struct Reference . . . 145
10.13.1 Detailed Description . . . 145
10.14 sound4::cloudx1::CPresetLoader Class Reference . . . 145
10.14.1 Detailed Description . . . 145
10.14.2 Constructor & Destructor Documentation . . . 146
10.14.2.1 CPresetLoader() . . . 146
10.14.2.2 ∼CPresetLoader() . . . 146
10.14.3 Member Function Documentation . . . 146
10.14.3.1 Exists() . . . 146
10.14.3.2 GetAll() . . . 146
10.14.3.3 IsReadOnly() . . . 146
10.14.3.4 Read() . . . 146
10.14.3.5 Remove() . . . 147
10.14.3.6 Rename() . . . 147
10.14.3.7 Write() . . . 147
10.15 sound4::cloudx1::dyn::CPresetLoader Class Reference . . . 147
10.15.1 Detailed Description . . . 147
10.15.2 Constructor & Destructor Documentation . . . 148
10.15.2.1 CPresetLoader() . . . 148
10.15.2.2 ∼CPresetLoader() . . . 148
10.15.3 Member Function Documentation . . . 148
10.15.3.1 Exists() . . . 148
10.15.3.2 GetAll() . . . 148
10.15.3.3 IsReadOnly() . . . 148
10.15.3.4 Read() . . . 149

10.15.3.5 Remove() . . . 149
10.15.3.6 Rename() . . . 149
10.15.3.7 Write() . . . 149
10.16 sound4::CPresetLoader Class Reference . . . 149
10.16.1 Detailed Description . . . 150
10.16.2 Constructor & Destructor Documentation . . . 150
10.16.2.1 CPresetLoader() . . . 150
10.16.2.2 ∼CPresetLoader() . . . 150
10.16.3 Member Function Documentation . . . 150
10.16.3.1 Exists() . . . 150
10.16.3.2 GetAll() . . . 151
10.16.3.3 IsReadOnly() . . . 151
10.16.3.4 Read() . . . 151
10.16.3.5 Remove() . . . 151
10.16.3.6 Rename() . . . 151
10.16.3.7 Write() . . . 152
10.17 sound4::CProcessor Class Reference . . . 152
10.17.1 Detailed Description . . . 154
10.17.2 Constructor & Destructor Documentation . . . 154
10.17.2.1 CProcessor() . . . 154
10.17.3 Member Function Documentation . . . 154
10.17.3.1 AudioConvertFrom() . . . 154
10.17.3.2 AudioConvertTo() . . . 155
10.17.3.3 AudioMonoFromLiveStereo() . . . 155
10.17.3.4 AudioMonoToLiveStereo() . . . 155
10.17.3.5 GetAudioInputCount() . . . 155
10.17.3.6 GetAudioOutputCount() . . . 156
10.17.3.7 GetBytesFromFormat() . . . 156
10.17.3.8 GetChannelCount() . . . 156
10.17.3.9 GetChunkSizeInFrames() . . . 157
10.17.3.10 GetFormatFromName() . . . 157
10.17.3.11 GetFormatName() . . . 157
10.17.3.12 GetPossibleChunkSizeInFrames() . . . 158
10.17.3.13 GetPrefixSymbol() . . . 158
10.17.3.14 GetSampleRate() . . . 158
10.17.3.15 GetVersion() . . . 159
10.17.3.16 IsOk() . . . 159
10.17.3.17 Load() . . . 159
10.17.3.18 MonoToStereo() . . . 159
10.17.3.19 MonoToStereo_Planar() . . . 160
10.17.3.20 SetLoggerCallback() . . . 160
10.17.3.21 SetLogSeverity() . . . 160

10.17.3.22 StereoToMono() . . . 160
10.17.3.23 StereoToMono_Planar() . . . 161
10.17.4 Friends And Related Function Documentation . . . 161
10.17.4.1 CBus . . . 161
10.17.4.2 CInstance . . . 161
10.17.5 Member Data Documentation . . . 161
10.17.5.1 fnAddAudio . . . 161
10.17.5.2 fnAddPadAudio . . . 161
10.17.5.3 fnAudioConvertFrom . . . 162
10.17.5.4 fnAudioConvertTo . . . 162
10.17.5.5 fnAudioMonoFromLiveStereo . . . 162
10.17.5.6 fnAudioMonoToLiveStereo . . . 162
10.17.5.7 fnDeleteClient . . . 162
10.17.5.8 fnExitProcess . . . 162
10.17.5.9 fnFreeBus . . . 163
10.17.5.10 fnFreeJsonAnswer . . . 163
10.17.5.11 fnFreeMetadataInfos . . . 163
10.17.5.12 fnFreeParameters . . . 163
10.17.5.13 fnFreeParameterValue . . . 163
10.17.5.14 fnGetAudio . . . 163
10.17.5.15 fnGetAudioInputCount . . . 164
10.17.5.16 fnGetAudioOutputCount . . . 164
10.17.5.17 fnGetBufferIn . . . 164
10.17.5.18 fnGetBufferOut . . . 164
10.17.5.19 fnGetBytesFromFormat . . . 164
10.17.5.20 fnGetChannelCount . . . 164
10.17.5.21 fnGetChunkSizeInFrames . . . 165
10.17.5.22 fnGetEstimatedDelay . . . 165
10.17.5.23 fnGetFormatFromName . . . 165
10.17.5.24 fnGetFormatName . . . 165
10.17.5.25 fnGetMaxPacketFrame . . . 165
10.17.5.26 fnGetMetadataInfos . . . 165
10.17.5.27 fnGetOutputCount . . . 166
10.17.5.28 fnGetParameter . . . 166
10.17.5.29 fnGetPossibleChunkSizeInFrames . . . 166
10.17.5.30 fnGetProcessChunkFrames . . . 166
10.17.5.31 fnGetSampleRate . . . 166
10.17.5.32 fnGetVersion . . . 166
10.17.5.33 fnInitProcess . . . 167
10.17.5.34 fnInitProcess2 . . . 167
10.17.5.35 fnInitProcess3 . . . 167
10.17.5.36 fnMonoToStereo . . . 167

10.17.5.37 fnMonoToStereo_Planar . . . 167
10.17.5.38 fnNewBus . . . 167
10.17.5.39 fnNewClient . . . 168
10.17.5.40 fnNewParameters . . . 168
10.17.5.41 fnPresetManager_InformChange . . . 168
10.17.5.42 fnProcessAudio . . . 168
10.17.5.43 fnProcessAudio_Planar . . . 168
10.17.5.44 fnProcessJson . . . 168
10.17.5.45 fnSaveState . . . 169
10.17.5.46 fnSetInstanceBus . . . 169
10.17.5.47 fnSetLoggerCallback . . . 169
10.17.5.48 fnSetLogSeverity . . . 169
10.17.5.49 fnSetMetadata . . . 169
10.17.5.50 fnSetMetadataMulti . . . 169
10.17.5.51 fnSetParameter . . . 170
10.17.5.52 fnSetPresetManager . . . 170
10.17.5.53 fnStartUpdateThread . . . 170
10.17.5.54 fnStereoToMono . . . 170
10.17.5.55 fnStereoToMono_Planar . . . 170
10.17.5.56 fnStopUpdateThread . . . 170
10.17.5.57 fnTerminateProcess . . . 171
10.17.5.58 fnWaitUpdateThreadReady . . . 171
10.17.5.59 fnWebserver . . . 171
10.17.5.60 fnWebserver_FreeString . . . 171
10.17.5.61 fnWebserver_GetAppHealth . . . 171
10.17.5.62 fnWebserver_SetAppHealth . . . 171
10.17.5.63 fnWebserver_Status . . . 172
10.17.5.64 fnWebserver_Stop . . . 172
10.17.5.65 fnWebserver_tcp . . . 172
10.17.5.66 fnWebserver_tcp2 . . . 172
10.18 sound4::cloudx1::dyn::helper::DynFuncHolder< T > Class Template Reference . . . 172
10.18.1 Detailed Description . . . 172
10.18.2 Constructor & Destructor Documentation . . . 173
10.18.2.1 DynFuncHolder() [1/2] . . . 173
10.18.2.2 DynFuncHolder() [2/2] . . . 173
10.18.3 Member Function Documentation . . . 173
10.18.3.1 IsOk() . . . 173
10.18.3.2 operator bool() . . . 173
10.18.3.3 operator T*() . . . 174
10.19 sound4::helper::DynFuncHolder< T > Class Template Reference . . . 174
10.19.1 Detailed Description . . . 174
10.19.2 Constructor & Destructor Documentation . . . 174

10.19.2.1 DynFuncHolder() [1/2] . . . 174
10.19.2.2 DynFuncHolder() [2/2] . . . 175
10.19.3 Member Function Documentation . . . 175
10.19.3.1 IsOk() . . . 175
10.19.3.2 operator bool() . . . 175
10.19.3.3 operator T∗() . . . 175
10.20 MyPresetManager Class Reference . . . 175
10.20.1 Detailed Description . . . 176
10.20.2 Constructor & Destructor Documentation . . . 176
10.20.2.1 MyPresetManager() [1/2] . . . 176
10.20.2.2 ∼MyPresetManager() [1/2] . . . 176
10.20.2.3 MyPresetManager() [2/2] . . . 176
10.20.2.4 ∼MyPresetManager() [2/2] . . . 177
10.20.3 Member Function Documentation . . . 177
10.20.3.1 Exists() [1/2] . . . 177
10.20.3.2 Exists() [2/2] . . . 177
10.20.3.3 GetAll() [1/2] . . . 177
10.20.3.4 GetAll() [2/2] . . . 177
10.20.3.5 IsReadOnly() [1/2] . . . 178
10.20.3.6 IsReadOnly() [2/2] . . . 178
10.20.3.7 Read() [1/2] . . . 178
10.20.3.8 Read() [2/2] . . . 178
10.20.3.9 Remove() [1/2] . . . 178
10.20.3.10 Remove() [2/2] . . . 179
10.20.3.11 Rename() [1/2] . . . 179
10.20.3.12 Rename() [2/2] . . . 179
10.20.3.13 SetupPath() [1/2] . . . 179
10.20.3.14 SetupPath() [2/2] . . . 179
10.20.3.15 Write() [1/2] . . . 180
10.20.3.16 Write() [2/2] . . . 180
10.21 sound4::cloudx1::dyn::helper::SampleFormat< T > Struct Template Reference . . . 180
10.21.1 Detailed Description . . . 180
10.22 sound4::cloudx1::helper::SampleFormat< T > Struct Template Reference . . . 181
10.22.1 Detailed Description . . . 181
10.23 sound4::helper::SampleFormat< T > Struct Template Reference . . . 181
10.23.1 Detailed Description . . . 181
10.24 sound4::cloudx1::dyn::helper::SampleFormat< float > Struct Reference . . . 182
10.24.1 Detailed Description . . . 182
10.24.2 Member Data Documentation . . . 182
10.24.2.1 format . . . 182
10.25 sound4::cloudx1::helper::SampleFormat< float > Struct Reference . . . 182
10.25.1 Detailed Description . . . 182

10.25.2 Member Data Documentation . . . 183
10.25.2.1 format . . . 183
10.26 sound4::helper::SampleFormat< float > Struct Reference . . . 183
10.26.1 Detailed Description . . . 183
10.26.2 Member Data Documentation . . . 183
10.26.2.1 format . . . 183
10.27 sound4::cloudx1::dyn::helper::SampleFormat< int16_t > Struct Reference . . . 183
10.27.1 Detailed Description . . . 184
10.27.2 Member Data Documentation . . . 184
10.27.2.1 format . . . 184
10.28 sound4::cloudx1::helper::SampleFormat< int16_t > Struct Reference . . . 184
10.28.1 Detailed Description . . . 184
10.28.2 Member Data Documentation . . . 184
10.28.2.1 format . . . 185
10.29 sound4::helper::SampleFormat< int16_t > Struct Reference . . . 185
10.29.1 Detailed Description . . . 185
10.29.2 Member Data Documentation . . . 185
10.29.2.1 format . . . 185
10.30 sound4::cloudx1::dyn::helper::SampleFormat< int32_t > Struct Reference . . . 185
10.30.1 Detailed Description . . . 186
10.30.2 Member Data Documentation . . . 186
10.30.2.1 format . . . 186
10.31 sound4::cloudx1::helper::SampleFormat< int32_t > Struct Reference . . . 186
10.31.1 Detailed Description . . . 186
10.31.2 Member Data Documentation . . . 186
10.31.2.1 format . . . 187
10.32 sound4::helper::SampleFormat< int32_t > Struct Reference . . . 187
10.32.1 Detailed Description . . . 187
10.32.2 Member Data Documentation . . . 187
10.32.2.1 format . . . 187
10.33 SOUND4_ProcessInfo Struct Reference . . . 187
10.33.1 Detailed Description . . . 188
10.33.2 Member Data Documentation . . . 188
10.33.2.1 name . . . 188
10.33.2.2 prefix . . . 188
10.33.2.3 shortname . . . 188
10.33.2.4 struct_size . . . 189
10.33.2.5 version . . . 189
10.34 sound4cl_SOUND4_ProcessInfo Struct Reference . . . 189
10.34.1 Detailed Description . . . 189
10.34.2 Member Data Documentation . . . 189
10.34.2.1 name . . . 190

10.34.2.2 prefix . . . 190
10.34.2.3 shortname . . . 190
10.34.2.4 struct_size . . . 190
10.34.2.5 version . . . 190

11 File Documentation **191**
11.1 sound4.x1.cloud-pipe-cxx.cpp File Reference . . . 191
11.1.1 Function Documentation . . . 191
11.1.1.1 GetEnv() . . . 191
11.1.1.2 main() . . . 192
11.1.1.3 MyLogger() . . . 192
11.1.1.4 ProcessLoop() . . . 192
11.1.1.5 Usage() . . . 192
11.2 sound4.x1.cloud-pipe-cxx.cpp . . . 193
11.3 sound4.x1.cloud-pipe-dyn.cpp File Reference . . . 194
11.3.1 Function Documentation . . . 195
11.3.1.1 GetEnv() . . . 195
11.3.1.2 main() . . . 196
11.3.1.3 MyLogger() . . . 196
11.3.1.4 ProcessLoop() . . . 196
11.3.1.5 Usage() . . . 197
11.4 sound4.x1.cloud-pipe-dyn.cpp . . . 197
11.5 sound4.x1.cloud-pipe.cpp File Reference . . . 200
11.5.1 Macro Definition Documentation . . . 200
11.5.1.1 CHANNEL_COUNT . . . 201
11.5.1.2 CHUNK_SIZE . . . 201
11.5.1.3 SAMPLE_RATE . . . 201
11.5.2 Function Documentation . . . 201
11.5.2.1 main() . . . 201
11.5.2.2 MyLogger() . . . 202
11.5.2.3 ProcessLoop() . . . 202
11.5.2.4 Usage() . . . 202
11.6 sound4.x1.cloud-pipe.cpp . . . 202
11.7 sound4.x1.cloud.h File Reference . . . 205
11.7.1 Macro Definition Documentation . . . 208
11.7.1.1 CLOUDX1_DECL . . . 208
11.7.1.2 CLOUDX1_HAS_CLOUDBUS . . . 208
11.7.1.3 CLOUDX1_HAS_WEBSERVER . . . 208
11.8 sound4.x1.cloud.h . . . 209
11.9 sound4.x1.cloud.hpp File Reference . . . 213
11.9.1 Macro Definition Documentation . . . 214
11.9.1.1 fs_strdup . . . 214

11.10 sound4.x1.cloud.hpp . . . 214
11.11 sound4.x1.cloud_dyn.hpp File Reference . . . 221
11.11.1 Detailed Description . . . 223
11.11.2 Macro Definition Documentation . . . 223
11.11.2.1 fs_strdup . . . 223
11.12 sound4.x1.cloud_dyn.hpp . . . 223
11.13 sound4cl-pipe.cpp File Reference . . . 236
11.13.1 Function Documentation . . . 237
11.13.1.1 GetEnv() . . . 237
11.13.1.2 main() . . . 237
11.13.1.3 ProcessLoop() . . . 238
11.14 sound4cl-pipe.cpp . . . 238
11.15 sound4cl.hpp File Reference . . . 241
11.15.1 Detailed Description . . . 242
11.16 sound4cl.hpp . . . 242
11.17 sound4cl_cdef.h File Reference . . . 255
11.17.1 Macro Definition Documentation . . . 257
11.17.1.1 fs_strdup . . . 258
11.17.1.2 SOUND4_INVALID_WEBSERVER_ID . . . 258
11.17.1.3 SOUND4_WEBSERVER_HTTP_OK . . . 258
11.17.1.4 SOUND4_WEBSERVER_HTTPS_OK . . . 258
11.17.2 Typedef Documentation . . . 258
11.17.2.1 fs_char . . . 258
11.17.2.2 sound4cl_loggerfn . . . 258
11.17.2.3 sound4cl_storage_exists . . . 259
11.17.2.4 sound4cl_storage_getall . . . 259
11.17.2.5 sound4cl_storage_getall_free . . . 259
11.17.2.6 sound4cl_storage_reader . . . 259
11.17.2.7 sound4cl_storage_reader_free . . . 259
11.17.2.8 sound4cl_storage_remove . . . 259
11.17.2.9 sound4cl_storage_rename . . . 260
11.17.2.10 sound4cl_storage_writer . . . 260
11.17.3 Enumeration Type Documentation . . . 260
11.17.3.1 sound4cl_LogSeverity . . . 260
11.17.3.2 sound4cl_PresetChange_Kind . . . 260
11.17.3.3 sound4cl_SampleFormat . . . 261
11.17.4 Function Documentation . . . 261
11.17.4.1 sound4cl_AddAudio() . . . 261
11.17.4.2 sound4cl_AddPadAudio() . . . 261
11.17.4.3 sound4cl_AudioConvertFrom() . . . 262
11.17.4.4 sound4cl_AudioConvertTo() . . . 262
11.17.4.5 sound4cl_AudioMonoFromLiveStereo() . . . 262

11.17.4.6 sound4cl_AudioMonoToLiveStereo() . . . 262
11.17.4.7 sound4cl_DeleteClient() . . . 262
11.17.4.8 sound4cl_ExitProcess() . . . 262
11.17.4.9 sound4cl_FreeBus() . . . 263
11.17.4.10 sound4cl_FreeJsonAnswer() . . . 263
11.17.4.11 sound4cl_FreeMetadataInfos() . . . 263
11.17.4.12 sound4cl_FreeParameters() . . . 263
11.17.4.13 sound4cl_FreeParameterValue() . . . 263
11.17.4.14 sound4cl_GetAudio() . . . 263
11.17.4.15 sound4cl_GetAudioInputCount() . . . 264
11.17.4.16 sound4cl_GetAudioOutputCount() . . . 264
11.17.4.17 sound4cl_GetBufferIn() . . . 264
11.17.4.18 sound4cl_GetBufferOut() . . . 264
11.17.4.19 sound4cl_GetBytesFromFormat() . . . 264
11.17.4.20 sound4cl_GetChannelCount() . . . 264
11.17.4.21 sound4cl_GetChunkSizeInFrames() . . . 264
11.17.4.22 sound4cl_GetEstimatedDelay() . . . 265
11.17.4.23 sound4cl_GetFormatFromName() . . . 265
11.17.4.24 sound4cl_GetFormatName() . . . 265
11.17.4.25 sound4cl_GetMaxPacketFrame() . . . 265
11.17.4.26 sound4cl_GetMetadataInfos() . . . 265
11.17.4.27 sound4cl_GetOutputCount() . . . 265
11.17.4.28 sound4cl_GetParameter() . . . 265
11.17.4.29 sound4cl_GetPossibleChunkSizeInFrames() . . . 266
11.17.4.30 sound4cl_GetProcessChunkFrames() . . . 266
11.17.4.31 sound4cl_GetSampleRate() . . . 266
11.17.4.32 sound4cl_GetVersion() . . . 266
11.17.4.33 sound4cl_InitProcess() . . . 266
11.17.4.34 sound4cl_InitProcess2() . . . 266
11.17.4.35 sound4cl_InitProcess3() . . . 267
11.17.4.36 sound4cl_MonoToStereo() . . . 267
11.17.4.37 sound4cl_MonoToStereo_Planar() . . . 267
11.17.4.38 sound4cl_NewBus() . . . 267
11.17.4.39 sound4cl_NewClient() . . . 267
11.17.4.40 sound4cl_NewParameters() . . . 267
11.17.4.41 sound4cl_PresetManager_InformChange() . . . 268
11.17.4.42 sound4cl_ProcessAudio() . . . 268
11.17.4.43 sound4cl_ProcessAudio_Planar() . . . 268
11.17.4.44 sound4cl_ProcessJson() . . . 268
11.17.4.45 sound4cl_SaveState() . . . 268
11.17.4.46 sound4cl_SetInstanceBus() . . . 268
11.17.4.47 sound4cl_SetLoggerCallback() . . . 269

11.17.4.48 sound4cl_SetLogSeverity() 269
11.17.4.49 sound4cl_SetMetadata() 269
11.17.4.50 sound4cl_SetMetadataMulti() 269
11.17.4.51 sound4cl_SetParameter() 269
11.17.4.52 sound4cl_SetPresetManager() 269
11.17.4.53 sound4cl_SOUND4_GetProcessInfo() 270
11.17.4.54 sound4cl_StartUpdateThread() 270
11.17.4.55 sound4cl_StereoToMono() 270
11.17.4.56 sound4cl_StereoToMono_Planar() 270
11.17.4.57 sound4cl_StopUpdateThread() 270
11.17.4.58 sound4cl_TerminateProcess() 270
11.17.4.59 sound4cl_WaitUpdateThreadReady() 271
11.17.4.60 sound4cl_Webserver() 271
11.17.4.61 sound4cl_Webserver_FreeString() 271
11.17.4.62 sound4cl_Webserver_GetAppHealth() 271
11.17.4.63 sound4cl_Webserver_SetAppHealth() 271
11.17.4.64 sound4cl_Webserver_Status() 271
11.17.4.65 sound4cl_Webserver_Stop() 272
11.17.4.66 sound4cl_Webserver_tcp() 272
11.17.4.67 sound4cl_Webserver_tcp2() 272
11.18 sound4cl_cdef.h 272

Index **275**

Chapter 1

SOUND4 x1.CLOUD Library

1.1 General 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

To access the presets/settings, you can either use JSON socket from Update Management or do all manually with Client's JSon config interfacing using JSON syntax

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

You can also use the C++ wrapper from header sound4.x1.cloud.hpp or the one with dynamic library loading sound4.x1.cloud_dyn.hpp.

Note

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

1.2 Examples

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

Chapter 2

Deprecated List

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

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Sample using C interface . . . 17
Environment variables and parameters . . . 19
Global setup . . . 20
Process creation/destruction . . . 26
Custom preset managagement . . . 31
Metadata Management . . . 37
Update Management . . . 39
Direct processing . . . 40
Audio managing (any audio) . . . 43
Audio convert helpers . . . 48
Client's JSon config interfacing . . . 52
JSON syntax . . . 54
JSON syntax for internal setup . . . 57
Web server . . . 61
Sample using C++ interface wrapper . . . 66
Sample using C++ interface wrapper with dynamic library loading . . . 67
Sample using Generic C++ interface (multi-process compatible) . . . 69

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

dyn 73
sound4 73
sound4::cloudx1 76
sound4::cloudx1::dyn 79
sound4::cloudx1::dyn::helper 82
sound4::cloudx1::helper 83
sound4::helper 83
std 84

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

sound4::CBus . . . 85
sound4::CInstance::CClient . . . 86
sound4::cloudx1::CInstance::CClient . . . 88
sound4::cloudx1::dyn::CInstance::CClient . . . 90
sound4::cloudx1::dyn::CDynLib . . . 92
sound4::cloudx1::dyn::helper::CDynLoader . . . 105
sound4::helper::CDynLoader . . . 108
sound4::CInstance . . . 111
sound4::cloudx1::CInstance . . . 122
sound4::cloudx1::dyn::CInstance . . . 133
cloudx1_CClientInstance . . . 144
cloudx1_CInstance . . . 144
cloudx1_CParameters . . . 145
sound4::cloudx1::CPresetLoader . . . 145
sound4::cloudx1::dyn::CPresetLoader . . . 147
MyPresetManager . . . 175
sound4::CPresetLoader . . . 149
MyPresetManager . . . 175
sound4::CProcessor . . . 152
sound4::cloudx1::dyn::helper::DynFuncHolder< T > . . . 172
sound4::helper::DynFuncHolder< T > . . . 174
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AddAudio) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AddPadAudio) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_DeleteClient) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_ExitProcess) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_FreeParameters) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_FreeParameterValue) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetAudio) > . . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetAudioInputCount) > . . . 172

sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetAudioOutputCount) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetBufferIn) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetBufferOut) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetChannelCount) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetChunkSizeInFrames) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetFormatName) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetOutputCount) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetParameter) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames) > . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetSampleRate) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_GetVersion) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_InitProcess) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_InitProcess2) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_InitProcess3) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_MonoToStereo) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_NewClient) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_NewParameters) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange) > . . 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_ProcessAudio) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_ProcessJson) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SaveState) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetLoggerCallback) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetLogSeverity) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetMetadata) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetParameter) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_SetPresetManager) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_StereoToMono) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_TerminateProcess) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_Status) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp) > 172
sound4::cloudx1::dyn::helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2) > 172
sound4::helper::DynFuncHolder< decltype(sound4cl_AddAudio) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_AddPadAudio) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_DeleteClient) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_ExitProcess) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_FreeBus) > 174

sound4::helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_FreeParameters) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetAudio) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetBufferIn) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetBufferOut) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetChannelCount) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetFormatName) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetOutputCount) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetParameter) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetSampleRate) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_GetVersion) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_InitProcess) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_InitProcess2) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_InitProcess3) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_MonoToStereo) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_NewBus) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_NewClient) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_NewParameters) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_ProcessAudio) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_ProcessJson) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SaveState) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetMetadata) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetParameter) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_SetPresetManager) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_StereoToMono) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_TerminateProcess) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_Status) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp) > 174
sound4::helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2) > 174
sound4::cloudx1::dyn::helper::SampleFormat< T > 180

sound4::cloudx1::helper::SampleFormat< T > 181
sound4::helper::SampleFormat< T > 181
sound4::cloudx1::dyn::helper::SampleFormat< float > 182
sound4::cloudx1::helper::SampleFormat< float > 182
sound4::helper::SampleFormat< float > 183
sound4::cloudx1::dyn::helper::SampleFormat< int16_t > 183
sound4::cloudx1::helper::SampleFormat< int16_t > 184
sound4::helper::SampleFormat< int16_t > 185
sound4::cloudx1::dyn::helper::SampleFormat< int32_t > 185
sound4::cloudx1::helper::SampleFormat< int32_t > 186
sound4::helper::SampleFormat< int32_t > 187
SOUND4_ProcessInfo 187
sound4cl_SOUND4_ProcessInfo 189

Chapter 6

Class Index

6.1 Class List

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

sound4::CBus
Shared Bus . . . 85
sound4::CInstance::CClient . . . 86
sound4::cloudx1::CInstance::CClient . . . 88
sound4::cloudx1::dyn::CInstance::CClient . . . 90
sound4::cloudx1::dyn::CDynLib
Dynamic library interface . . . 92
sound4::cloudx1::dyn::helper::CDynLoader
Helper to load dynamic library . . . 105
sound4::helper::CDynLoader
Helper to load dynamic library . . . 108
sound4::CInstance
Instance handling class . . . 111
sound4::cloudx1::CInstance . . . 122
sound4::cloudx1::dyn::CInstance . . . 133
cloudx1_CClientInstance . . . 144
cloudx1_CInstance . . . 144
cloudx1_CParameters . . . 145
sound4::cloudx1::CPresetLoader . . . 145
sound4::cloudx1::dyn::CPresetLoader . . . 147
sound4::CPresetLoader
Custom preset handler helper . . . 149
sound4::CProcessor
Dynamic library interface . . . 152
sound4::cloudx1::dyn::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 172
sound4::helper::DynFuncHolder< T >
Helper to wrap a function pointer . . . 174
MyPresetManager
[Custom preset manager] . . . 175
sound4::cloudx1::dyn::helper::SampleFormat< T >
Helper for Sample format types . . . 180
sound4::cloudx1::helper::SampleFormat< T >
Helper for Sample format types . . . 181
sound4::helper::SampleFormat< T >
Helper for Sample format types . . . 181

sound4::cloudx1::dyn::helper::SampleFormat< float > 182
sound4::cloudx1::helper::SampleFormat< float > 182
sound4::helper::SampleFormat< float > 183
sound4::cloudx1::dyn::helper::SampleFormat< int16_t > 183
sound4::cloudx1::helper::SampleFormat< int16_t > 184
sound4::helper::SampleFormat< int16_t > 185
sound4::cloudx1::dyn::helper::SampleFormat< int32_t > 185
sound4::cloudx1::helper::SampleFormat< int32_t > 186
sound4::helper::SampleFormat< int32_t > 187
SOUND4_ProcessInfo 187
sound4cl_SOUND4_ProcessInfo 189

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

sound4.x1.cloud-pipe-cxx.cpp
C++ sample using C++ interface wrapper 191
sound4.x1.cloud-pipe-dyn.cpp
C++ sample using C++ interface wrapper with dynamic library loading 194
sound4.x1.cloud-pipe.cpp
C++ sample using only C interface 200
sound4.x1.cloud.h
C interface for library 205
sound4.x1.cloud.hpp
C++ wrapper for library 213
sound4.x1.cloud_dyn.hpp
C++ wrapper for dynamic loading library 221
sound4cl-pipe.cpp
C++ sample using C++ interface wrapper with dynamic library loading Works with any SOUND4 processor provided 236
sound4cl.hpp
C++ wrapper for dynamic loading library 241
sound4cl_cdef.h
C function signature definition for all sound4.cl libraries 255

Chapter 8

Module Documentation

8.1 Sample using C interface

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

8.1.1 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);
     }*/
```

8.1.2 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;
        }
    }
```

8.1.3 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);
```

8.2 Environment variables and parameters

8.2.1 Parameters defaulting from environment

The following environment variables are used by software library:

Note

Those variables can be passed as parameters with cloudx1_InitProcess2

8.2.1.1 Webserver authentification setup :

- `SETUP_USER` see cloudx1_InitProcess
- `SETUP_SECRET` see cloudx1_InitProcess
- `ADMIN_USER` see cloudx1_InitProcess
- `ADMIN_SECRET` see cloudx1_InitProcess

8.2.1.2 Webserver secure :

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

8.2.1.3 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_CHECK` : if "false", do not check HTTPS certificates

8.2.2 Parameters

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

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=notempty` : force the storage in read-only mode
- `STATE_RO=notempty` : force the state in read-only mode (but can save presets/auth)
- `WAIT_LICENSE=notempty` : force to wait the license load before starting (always in file mode)
- `LOG_HEADER=string` : use string as header for logs in instance as much as possible

8.2.3 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.

8.3 Global setup

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)

8.3.1 Detailed Description

Get information on the library. Set logging mode

8.3.2 Macro Definition Documentation

8.3.2.1 CLOUDX1_AUDIO_INPUT_COUNT

```
#define CLOUDX1_AUDIO_INPUT_COUNT 1
```

This version number of inputs for this process.

Remarks

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

Definition at line 169 of file sound4.x1.cloud.h.

8.3.2.2 CLOUDX1_AUDIO_OUTPUT_COUNT

```
#define CLOUDX1_AUDIO_OUTPUT_COUNT 1
```

This version number of outputs for this process.

Remarks

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

Definition at line 175 of file sound4.x1.cloud.h.

8.3.2.3 CLOUDX1_AUDIOFRAME_COUNT

```
#define CLOUDX1_AUDIOFRAME_COUNT 64
```

This version number of frames in a process chunk.

Remarks

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

Definition at line 157 of file sound4.x1.cloud.h.

8.3.2.4 CLOUDX1_CHANNEL_COUNT

```
#define CLOUDX1_CHANNEL_COUNT 2
```

This version number of channels.

Remarks

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

Definition at line 163 of file sound4.x1.cloud.h.

8.3.2.5 CLOUDX1_SAMPLE_RATE

```
#define CLOUDX1_SAMPLE_RATE 48000
```

This input/output sample-rate.

Remarks

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

Definition at line 181 of file sound4.x1.cloud.h.

8.3.3 Typedef Documentation

8.3.3.1 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 296 of file sound4.x1.cloud.h.

8.3.4 Enumeration Type Documentation

8.3.4.1 cloudx1_LogSeverity

```
enum cloudx1_LogSeverity
```

The logging levels

Enumerator

none	
fatal	
error	
warning	
info	
verbose	
verbose2	
verbose3	
verbose4	
verbose5	

Definition at line 277 of file sound4.x1.cloud.h.

8.3.5 Function Documentation

8.3.5.1 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

8.3.5.2 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

8.3.5.3 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

8.3.5.4 cloudx1_GetChunkSizeInFrames()

```
unsigned int cloudx1_GetChunkSizeInFrames ( )
```

Get the 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

8.3.5.5 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

8.3.5.6 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

8.3.5.7 cloudx1_GetVersion()

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

Get the process version string

Returns

version string

8.3.5.8 cloudx1_SetLoggerCallback()

```
void cloudx1_SetLoggerCallback (
            cloudx1_loggerfn logger )
```

Set a callback for logging

If not set, a default logger prints to stderr.

Warning

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

Parameters

logger	A C callback called for logging.

8.3.5.9 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

8.3.5.10 SOUND4_GetProcessInfo()

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

Get this processor description

Common to all SOUND4 processors.

Returns

the process description

8.4 Process creation/destruction

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)

8.4.1 Detailed Description

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

8.4.2 Function Documentation

8.4.2.1 cloudx1_ExitProcess()

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

Quit and destroy the instance

Parameters

instance	the instance to destroy

8.4.2.2 cloudx1_FreeParameters()

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

Free the parameters created with cloudx1_NewParameters.

8.4.2.3 cloudx1_FreeParameterValue()

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

Free a parameter value returned by cloudx1_GetParameter

Parameters

value	the returned value

8.4.2.4 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

8.4.2.5 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

8.4.2.6 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

8.4.2.7 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

8.4.2.8 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.

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

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 (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

8.4.2.9 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).

8.4.2.10 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

8.4.2.11 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

8.5 Custom preset managagement

This allows to provide a customized preset management.

Typedefs

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

Enumerations

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

Functions

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

8.5.1 Detailed Description

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

By default, the library store data on disk or in memory (see `save_path` parameter in cloudx1_InitProcess2), but you may use these functions to store data.

It can be used for remote storage, database storage, ...

Known files:

- Files with name ending with .pst are presets
- `state.json` is the instance state file, holding settings and unsaved preset changes.
- `auth.json` is the storage for web authentication settings.
- *.onair are used for sharing preset on air name between instances (see `STATE_ONAIR` in Parameters).

Attention

state.json name can be overriden per instance with `STATE_FILENAME` in parameters, see Parameters.

File names are in OS native characters (`wchar_t` for Windows, `char` in **UTF-8** for others)

8.5.2 Typedef Documentation

8.5.2.1 cloudx1_storage_exists

```
typedef int(* cloudx1_storage_exists) (const fs_char *filename, void *userdata)
```

Function to call to test if a stored file exists.

Parameters

filename	the name of the file to test.
userdata	the userdata passed to cloudx1_SetPresetManager

Returns

-1 if not exists, 0 if it exists

Definition at line 603 of file sound4.x1.cloud.h.

8.5.2.2 cloudx1_storage_getall

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

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

Parameters

userdata	the userdata passed to cloudx1_SetPresetManager

Returns

A NULL terminated array of file names

Definition at line 612 of file sound4.x1.cloud.h.

8.5.2.3 cloudx1_storage_getall_free

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

Function to call to free the result of getall.

Parameters

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

Definition at line 620 of file sound4.x1.cloud.h.

8.5.2.4 cloudx1_storage_reader

```
typedef char*(* cloudx1_storage_reader) (const fs_char *filename, void *userdata)
```

Function to call to read a stored file, returning its content.

Parameters

filename	the name of the file to read.
userdata	the userdata passed to cloudx1_SetPresetManager

Returns

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

Definition at line 576 of file sound4.x1.cloud.h.

8.5.2.5 cloudx1_storage_reader_free

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

Function to call to free the result of cloudx1_storage_reader.

Parameters

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

Definition at line 584 of file sound4.x1.cloud.h.

8.5.2.6 cloudx1_storage_remove

```
typedef int(* cloudx1_storage_remove) (const fs_char *filename, void *userdata)
```

Function to call to remove a stored file.

Parameters

filename	the name of the file to remove.
userdata	the userdata passed to cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 629 of file sound4.x1.cloud.h.

8.5.2.7 cloudx1_storage_rename

```
typedef int(* cloudx1_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)
```

Function to call to rename a stored file.

Parameters

from	The source file name (must exist)
to	The destination file name (must not exist)
userdata	the userdata passed to cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 639 of file sound4.x1.cloud.h.

8.5.2.8 cloudx1_storage_writer

```
typedef int(* cloudx1_storage_writer) (const fs_char *filename, const char *content, void *userdata)
```

Function to call to write a stored file.

Parameters

filename	the name of the file to write.
content	the **UTF-8** content to write to the file
userdata	the userdata passed to cloudx1_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 594 of file sound4.x1.cloud.h.

8.5.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 564 of file sound4.x1.cloud.h.

8.5.3 Enumeration Type Documentation

8.5.3.1 cloudx1_PresetChange_Kind

```
enum cloudx1_PresetChange_Kind
```

Kind of change on a storage path

Enumerator

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

Definition at line 672 of file sound4.x1.cloud.h.

8.5.4 Function Documentation

8.5.4.1 cloudx1_PresetManager_InformChange()

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

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

Parameters

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

8.5.4.2 cloudx1_SetPresetManager()

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

```
int IsReadOnly,
void * userdata )
```

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

To dynamically update when storage change, see cloudx1_PresetManager_InformChange

Parameters

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

8.6 Metadata Management

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)

8.6.1 Detailed Description

Metadata handling.

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

8.6.2 Function Documentation

8.6.2.1 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

8.6.2.2 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

8.6.2.3 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.

8.6.2.4 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.

8.7 Update Management

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)

8.7.1 Detailed Description

Manage the process update.

8.7.2 Function Documentation

8.7.2.1 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

8.7.2.2 cloudx1_StopUpdateThread()

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

Stop the update thread

Parameters

instance	the processing instance

8.7.2.3 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

8.8 Direct processing

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)

8.8.1 Detailed Description

Optimized processing with direct in->out.

8.8.2 Function Documentation

8.8.2.1 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.

8.8.2.2 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.

8.8.2.3 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.

Returns

Estimated delay in frames

8.8.2.4 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

8.8.2.5 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)

8.9 Audio managing (any audio)

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)

8.9.1 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.

8.9.2 Enumeration Type Documentation

8.9.2.1 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 895 of file sound4.x1.cloud.h.

8.9.3 Function Documentation

8.9.3.1 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

8.9.3.2 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

8.9.3.3 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

8.9.3.4 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

8.9.3.5 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.

8.9.3.6 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

8.9.3.7 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.

8.9.3.8 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

8.10 Audio convert helpers

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)

8.10.1 Detailed Description

8.10.2 Function Documentation

8.10.2.1 cloudx1_AudioConvertFrom()

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

Convert the payload to floating-point array

Warning

Output buffer must be 64-byte aligned

Parameters

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

8.10.2.2 cloudx1_AudioConvertTo()

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

Convert floating-point array to payload

Warning

Input buffer must be 64-byte aligned

Parameters

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

8.10.2.3 cloudx1_AudioMonoFromLiveStereo()

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

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

Parameters

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

8.10.2.4 cloudx1_AudioMonoToLiveStereo()

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

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

Parameters

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

8.10.2.5 cloudx1_MonoToStereo()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

8.10.2.6 cloudx1_MonoToStereo_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

8.10.2.7 cloudx1_StereoToMono()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

8.10.2.8 cloudx1_StereoToMono_Planar()

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

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

Warning

Buffers must be 64-byte aligned

Parameters

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

8.11 Client's JSon config interfacing

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)

8.11.1 Detailed Description

Allow low-level JSON configuration. see also JSON syntax

8.11.2 Function Documentation

8.11.2.1 cloudx1_DeleteClient()

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

Deletes a client

Parameters

client	the client instance

8.11.2.2 cloudx1_FreeJsonAnswer()

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

Free a JSON answer returned by cloudx1_ProcessJson

Parameters

json_str	the JSON request

8.11.2.3 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

8.11.2.4 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)

8.11.2.5 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

8.12 JSON syntax

JSON protocol to configure the process

8.12.1 Commands

`{ "get" : 'object' }`

get values.
If `object` is null, get all parameters, and so show the possibilities
`{ "set" : 'object' }`

set values.
`object` can be any subpart, only those will be changed
`{ "change" : 'object' }`

like get, but shows only changes since last request.
If `object` is null, get all changes.
Each call resets the state for this connection to current.

Note

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

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

load the preset

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

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

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

delete the preset.
can also be an array of presetnames

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

Create a preset from factory and load it

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

Rename a preset

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

Import a preset

```
{ "export" : "`presetname`"  }
```

Export a preset: get preset's content so it can be imported later

```
{ "new" : { "name": "`presetname`", "factory": "`presetname`" } }
```

Create a new preset from a factory.
Can use array for multiple.

```
{ "undo" : `count`  }
```

Undo last(s) `count` preset change

```
{ "redo" : `count`  }
```

Redo last(s) `count` preset change

See also JSON syntax for internal setup

8.12.2 Answers

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

```
{ "get" : ... }
{ "change" : ... }
```

For other commands, if there is no error the answer is

```
{ "OK" : {"command" : "original command" } }
```

If there is an error, answer has the syntax

```
{ "ERROR" : {"command" : "faulty command", "message" : "explaining"} }
```

and may have more properties. The message may be empty.

Note

In x1.Cloud version <= 1.2.2, the `OK` and `ERROR` answers had the syntax

```
{ "OK" : "original command" }
{ "ERROR" : "faulty command" }
```

8.12.3 Examples

8.12.3.1 Get status

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

answer:

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

8.12.3.2 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":{...}},
        ...
        }
```

8.12.3.3 Check license

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

answer:

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

8.12.3.4 Set Processor name

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

answer:

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

8.12.3.5 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"},
            ...
        ]}}
```

8.12.3.6 Load a preset

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

answer:

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

8.12.3.7 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"}}}
```

8.12.3.8 Set input gain

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

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

8.12.3.9 Set bypass

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

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

8.12.3.10 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

8.12.3.11 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"}
```

8.13 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.

8.13.1 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}
       }}}
```

8.13.2 Authentication management

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

Secret stored is

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

8.13.2.1 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.

8.13.2.2 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

8.13.2.3 Change setup account

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

8.13.2.3.1 Set the setup/admin account

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

answer:

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

8.13.2.3.2 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"}}
```

8.13.2.3.3 Remove the setup/admin account

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

answer:

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

8.13.2.4 Get user info

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

answer:

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

8.13.2.5 Add or modify a user

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

answer:

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

8.13.2.6 Removes a user

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

answer:

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

8.13.2.7 List users

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

answer:

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

8.13.2.8 Get/set unlogged access

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

answer:

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

8.13.2.9 Get realm

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

answer:

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

8.13.2.10 Get API version

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

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

8.13.2.11 Get setup is admin

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

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

8.13.2.12 Get/set setup account

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

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

8.13.2.13 Get/set custom storage

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

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

8.13.3 Get process' interface description

8.13.3.1 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 !

8.13.3.2 Get Process Image

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

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

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

8.13.3.3 Get Process Ico

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

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

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

8.14 Web server

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)

8.14.1 Detailed Description

Run a web server to configure the processor

See also how to setup authentication with Environment variables and parameters. Their use is documented in cloudx1_InitProcess.

8.14.2 Macro Definition Documentation

8.14.2.1 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 1571 of file sound4.x1.cloud.h.

8.14.2.2 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 1576 of file sound4.x1.cloud.h.

8.14.2.3 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 1580 of file sound4.x1.cloud.h.

8.14.3 Function Documentation

8.14.3.1 cloudx1_Webserver()

```
uint64_t cloudx1_Webserver (
            unsigned int listenport,
            unsigned int listenport_secure,
            struct cloudx1_CInstance * instance )
```

Run a web server, connecting to local instance internally (no expose port needed)

This function returns after connecting to the process, and continue to run in background See cloudx1_Webserver_Stop too stop it gracefully

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

Warning

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

Parameters

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

Returns

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

8.14.3.2 cloudx1_Webserver_FreeString()

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

Free a a string returned by cloudx1_Webserver_GetAppHealth

Parameters

str	the string to free

8.14.3.3 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)

8.14.3.4 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

8.14.3.5 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

8.14.3.6 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

8.14.3.7 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

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

Returns

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

8.14.3.8 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

8.15 Sample using C++ interface wrapper

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

8.15.1 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");
```

8.15.2 Sample to create the processing instance

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

8.15.3 Communicating with the instance

See also

JSON syntax

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

8.15.4 Starting a web-server

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

8.15.5 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;
    }
}
```

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

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

8.16.1 Dynamic loading

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

8.16.2 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");
```

8.16.3 Sample to create the processing instance

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

8.16.4 Communicating with the instance

See also

JSON syntax

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

8.16.5 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;
}
```

8.16.6 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;
    }
}
```

8.16.7 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;
            if ( !std::filesystem::exists(m_preset_path ) ) {
                return false;
            }
            std::error_code ec;
            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;
            for (const auto & entry : std::filesystem::directory_iterator(m_preset_path.c_str(),
   std::filesystem::directory_options::skip_permission_denied)) {
                if (entry.is_regular_file()) {
                    list.push_back(entry.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;
        }
};
```

8.16.8 Custom preset manager loading

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

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

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

It uses sound4cl.hpp header.

See full example cpp/sound4cl-pipe.cpp

8.17.1 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;
    }
```

8.17.2 Setup and configure

```
    processor.SetLogSeverity(sound4::verbose5);
    processor.SetLoggerCallback([](sound4::LogSeverity severity,const std::string& msg) {
            std::cerr « msg « std::endl;
        });
    sound4::CInstance instance(processor);
    instance.SetParam("ADMIN_USER","admin");
    instance.SetParam("ADMIN_SECRET","admin");
```

8.17.3 Create the processing instance

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

8.17.4 Processing loop

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

8.17.5 Starting a web-server

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

8.17.6 Communicating with the instance

See also

JSON syntax

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

8.17.7 Custom preset manager example

```
class MyPresetManager : public sound4::CPresetLoader {
public:
    MyPresetManager() = default;
    virtual ~MyPresetManager() = default;
    bool SetupPath(const std::filesystem::path& a_preset_path)
        {
            m_preset_path=a_preset_path;
            if ( !std::filesystem::exists(m_preset_path ) ) {
                return false;
            }
            std::error_code ec;
            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;
            for (const auto & entry : std::filesystem::directory_iterator(m_preset_path.c_str(),
       std::filesystem::directory_options::skip_permission_denied)) {
                if (entry.is_regular_file()) {
                    list.push_back(entry.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;
        }
};
```

8.17.8 Custom preset manager loading

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

Chapter 9

Namespace Documentation

9.1 dyn Namespace Reference

9.1.1 Detailed Description

Dynamic loading namespace

9.2 sound4 Namespace Reference

Namespaces

- cloudx1
- helper

Classes

- class CProcessor

  Dynamic library interface.
- class CBus

  Shared Bus.
- class CPresetLoader

  Custom preset handler helper.
- class CInstance

  Instance handling class.

Typedefs

- using LogSeverity = sound4cl_LogSeverity
- using log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using PresetChange_Kind = sound4cl_PresetChange_Kind

Variables

- const LogSeverity none = (LogSeverity) sound4cl_none
- const LogSeverity fatal = (LogSeverity) sound4cl_fatal
- const LogSeverity error = (LogSeverity) sound4cl_error
- const LogSeverity warning = (LogSeverity) sound4cl_warning
- const LogSeverity info = (LogSeverity) sound4cl_info
- const LogSeverity verbose = (LogSeverity) sound4cl_verbose
- const LogSeverity verbose2 = (LogSeverity) sound4cl_verbose2
- const LogSeverity verbose3 = (LogSeverity) sound4cl_verbose3
- const LogSeverity verbose4 = (LogSeverity) sound4cl_verbose4
- const LogSeverity verbose5 = (LogSeverity) sound4cl_verbose5

9.2.1 Detailed Description

SOUND4 namespace

9.2.2 Typedef Documentation

9.2.2.1 log_cb_t

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

Log callback type

Definition at line 252 of file sound4cl.hpp.

9.2.2.2 LogSeverity

```
using sound4::LogSeverity = typedef sound4cl_LogSeverity
```

Log Severity levels

Definition at line 237 of file sound4cl.hpp.

9.2.2.3 PresetChange_Kind

```
using sound4::PresetChange_Kind = typedef sound4cl_PresetChange_Kind
```

Definition at line 743 of file sound4cl.hpp.

9.2.3 Variable Documentation

9.2.3.1 error

```
const LogSeverity sound4::error = (LogSeverity) sound4cl_error
```

Definition at line 240 of file sound4cl.hpp.

9.2.3.2 fatal

```
const LogSeverity sound4::fatal = (LogSeverity) sound4cl_fatal
```

Definition at line 239 of file sound4cl.hpp.

9.2.3.3 info

```
const LogSeverity sound4::info = (LogSeverity) sound4cl_info
```

Definition at line 242 of file sound4cl.hpp.

9.2.3.4 none

```
const LogSeverity sound4::none = (LogSeverity) sound4cl_none
```

Definition at line 238 of file sound4cl.hpp.

9.2.3.5 verbose

```
const LogSeverity sound4::verbose = (LogSeverity) sound4cl_verbose
```

Definition at line 243 of file sound4cl.hpp.

9.2.3.6 verbose2

```
const LogSeverity sound4::verbose2 = (LogSeverity) sound4cl_verbose2
```

Definition at line 244 of file sound4cl.hpp.

9.2.3.7 verbose3

```
const LogSeverity sound4::verbose3 = (LogSeverity) sound4cl_verbose3
```

Definition at line 245 of file sound4cl.hpp.

9.2.3.8 verbose4

```
const LogSeverity sound4::verbose4 = (LogSeverity) sound4cl_verbose4
```

Definition at line 246 of file sound4cl.hpp.

9.2.3.9 verbose5

```
const LogSeverity sound4::verbose5 = (LogSeverity) sound4cl_verbose5
```

Definition at line 247 of file sound4cl.hpp.

9.2.3.10 warning

```
const LogSeverity sound4::warning = (LogSeverity) sound4cl_warning
```

Definition at line 241 of file sound4cl.hpp.

9.3 sound4::cloudx1 Namespace Reference

Namespaces

- dyn
- helper

Classes

- class CPresetLoader
- class CInstance

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

Variables

- const size_t ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
- const size_t InputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_INPUT_COUNT
- const size_t OutputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_OUTPUT_COUNT
- const size_t InputSampleSize =ChunkFrames∗InputChannels
- const size_t OutputSampleSize =ChunkFrames∗OutputChannels
- const size_t SampleRate =CLOUDX1_SAMPLE_RATE

9.3.1 Detailed Description

cloudx1 namespace

9.3.2 Typedef Documentation

9.3.2.1 log_cb_t

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

Log callback type

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

9.3.2.2 LogSeverity

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

Log Severity levels

Definition at line 316 of file sound4.x1.cloud.hpp.

9.3.2.3 PresetChange_Kind

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

Definition at line 380 of file sound4.x1.cloud.hpp.

9.3.2.4 SampleFormat

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

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

9.3.3 Variable Documentation

9.3.3.1 ChunkFrames

```
const size_t sound4::cloudx1::ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
```

Number of frames in a chunk

Definition at line 57 of file sound4.x1.cloud.hpp.

9.3.3.2 InputChannels

```
const size_t sound4::cloudx1::InputChannels =CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT
```

Number of float values in an input frame

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

9.3.3.3 InputSampleSize

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

Number of float values for an input chunk

Definition at line 69 of file sound4.x1.cloud.hpp.

9.3.3.4 OutputChannels

```
const size_t sound4::cloudx1::OutputChannels =CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT
```

Number of float values in an output frame

Definition at line 65 of file sound4.x1.cloud.hpp.

9.3.3.5 OutputSampleSize

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

Number of float values for an output chunk

Definition at line 73 of file sound4.x1.cloud.hpp.

9.3.3.6 SampleRate

```
const size_t sound4::cloudx1::SampleRate =CLOUDX1_SAMPLE_RATE
```

Sample rate of library

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

9.4 sound4::cloudx1::dyn Namespace Reference

Namespaces

- helper

Classes

- class CDynLib

  Dynamic library interface.
- class CPresetLoader
- class CInstance

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

Variables

- const size_t ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
- const size_t InputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_INPUT_COUNT
- const size_t OutputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_OUTPUT_COUNT
- const size_t InputSampleSize =ChunkFrames∗InputChannels
- const size_t OutputSampleSize =ChunkFrames∗OutputChannels
- const size_t SampleRate =CLOUDX1_SAMPLE_RATE

9.4.1 Typedef Documentation

9.4.1.1 log_cb_t

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

Log callback type

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

9.4.1.2 LogSeverity

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

Log Severity levels

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

9.4.1.3 PresetChange_Kind

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

Definition at line 799 of file sound4.x1.cloud_dyn.hpp.

9.4.1.4 SampleFormat

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

Definition at line 460 of file sound4.x1.cloud_dyn.hpp.

9.4.2 Variable Documentation

9.4.2.1 ChunkFrames

```
const size_t sound4::cloudx1::dyn::ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
```

Number of frames in a chunk

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

9.4.2.2 InputChannels

```
const size_t sound4::cloudx1::dyn::InputChannels =CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT
```

Number of float values in an input frame

Definition at line 468 of file sound4.x1.cloud_dyn.hpp.

9.4.2.3 InputSampleSize

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

Number of float values for an input chunk

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

9.4.2.4 OutputChannels

```
const size_t sound4::cloudx1::dyn::OutputChannels =CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT
```

Number of float values in an output frame

Definition at line 472 of file sound4.x1.cloud_dyn.hpp.

9.4.2.5 OutputSampleSize

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

Number of float values for an output chunk

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

9.4.2.6 SampleRate

```
const size_t sound4::cloudx1::dyn::SampleRate =CLOUDX1_SAMPLE_RATE
```

Sample rate of library

Definition at line 484 of file sound4.x1.cloud_dyn.hpp.

9.5 sound4::cloudx1::dyn::helper Namespace Reference

Classes

- class DynFuncHolder

  Helper to wrap a function pointer.
- class CDynLoader

  Helper to load dynamic library.
- struct SampleFormat

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

Typedefs

- using DynLib_t = HMODULE
- using DynFunc_t = FARPROC

9.5.1 Detailed Description

helper functions for unicode strings in Windows

9.5.2 Typedef Documentation

9.5.2.1 DynFunc_t

```
using sound4::cloudx1::dyn::helper::DynFunc_t = typedef FARPROC
```

Definition at line 77 of file sound4.x1.cloud_dyn.hpp.

9.5.2.2 DynLib_t

```
using sound4::cloudx1::dyn::helper::DynLib_t = typedef HMODULE
```

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

9.6 sound4::cloudx1::helper Namespace Reference

Classes

- struct SampleFormat

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

9.6.1 Detailed Description

helper functions for unicode strings in Windows

9.7 sound4::helper Namespace Reference

Classes

- struct SampleFormat

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

  Helper to wrap a function pointer.
- class CDynLoader

  Helper to load dynamic library.

Typedefs

- using DynLib_t = HMODULE
- using DynFunc_t = FARPROC

9.7.1 Detailed Description

helper functions for unicode strings in Windows

9.7.2 Typedef Documentation

9.7.2.1 DynFunc_t

```
using sound4::helper::DynFunc_t = typedef FARPROC
```

Definition at line 106 of file sound4cl.hpp.

9.7.2.2 DynLib_t

```
using sound4::helper::DynLib_t = typedef HMODULE
```

Definition at line 105 of file sound4cl.hpp.

9.8 std Namespace Reference

Chapter 10

Class Documentation

10.1 sound4::CBus Class Reference

Shared Bus.

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

Public Member Functions

- CBus (CProcessor &dynlib)
- ~CBus ()
- sound4cl_CBus * Get () const

10.1.1 Detailed Description

Some processor can share a bus for realtime data exchange

Definition at line 703 of file sound4cl.hpp.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 CBus()

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

Definition at line 705 of file sound4cl.hpp.

10.1.2.2 ~CBus()

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

Definition at line 712 of file sound4cl.hpp.

10.1.3 Member Function Documentation

10.1.3.1 Get()

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

Definition at line 719 of file sound4cl.hpp.

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

- sound4cl.hpp

10.2 sound4::CInstance::CClient Class Reference

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

Public Member Functions

- CClient (sound4cl_CInstance *instance, CProcessor &dynlib)
- CClient (const CInstance &)=delete
- CClient & operator= (const CInstance &)=delete
- ~CClient ()
- std::string ProcessJson (const std::string &request, bool *NeedSave=nullptr)

  Process a JSON request and returns the answer.

10.2.1 Detailed Description

Low-level JSON client.

See also

JSON syntax

Definition at line 1215 of file sound4cl.hpp.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 CClient() [1/2]

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

Definition at line 1217 of file sound4cl.hpp.

10.2.2.2 CClient() [2/2]

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

10.2.2.3 ~CClient()

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

Definition at line 1227 of file sound4cl.hpp.

10.2.3 Member Function Documentation

10.2.3.1 operator=()

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

10.2.3.2 ProcessJson()

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

Parameters

request	the JSON request.

See also

JSON syntax

Parameters

NeedSave	if non null, returns if the saving is required (maybe delayed)

Returns

the JSON answer

Definition at line 1241 of file sound4cl.hpp.

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

- sound4cl.hpp

10.3 sound4::cloudx1::CInstance::CClient Class Reference

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

Public Member Functions

- CClient (cloudx1_CInstance *instance)
- CClient (const CInstance &)=delete
- CClient & operator= (const CInstance &)=delete
- ~CClient ()
- std::string ProcessJson (const std::string &request, bool *NeedSave=nullptr)

  Process a JSON request and returns the answer.

10.3.1 Detailed Description

Low-level JSON client.

See also

JSON syntax

Definition at line 827 of file sound4.x1.cloud.hpp.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 CClient() [1/2]

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

Definition at line 829 of file sound4.x1.cloud.hpp.

10.3.2.2 CClient() [2/2]

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

10.3.2.3 ~CClient()

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

Definition at line 838 of file sound4.x1.cloud.hpp.

10.3.3 Member Function Documentation

10.3.3.1 operator=()

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

10.3.3.2 ProcessJson()

```
std::string sound4::cloudx1::CInstance::CClient::ProcessJson (
            const std::string & request,
            bool * NeedSave = nullptr )  [inline]
```

Parameters

request	the JSON request.

See also

JSON syntax

Parameters

NeedSave	if non null, returns if the saving is required (maybe delayed)

Returns

the JSON answer

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

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

- sound4.x1.cloud.hpp

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

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

Public Member Functions

- CClient (cloudx1_CInstance *instance, CDynLib &dynlib)
- CClient (const CInstance &)=delete
- CClient & operator= (const CInstance &)=delete
- ~CClient ()
- std::string ProcessJson (const std::string &request, bool *NeedSave=nullptr)

  Process a JSON request and returns the answer.

10.4.1 Detailed Description

Low-level JSON client.

See also

JSON syntax

Definition at line 1270 of file sound4.x1.cloud_dyn.hpp.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 CClient() [1/2]

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

Definition at line 1272 of file sound4.x1.cloud_dyn.hpp.

10.4.2.2 CClient() [2/2]

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

10.4.2.3 ~CClient()

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

Definition at line 1282 of file sound4.x1.cloud_dyn.hpp.

10.4.3 Member Function Documentation

10.4.3.1 operator=()

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

10.4.3.2 ProcessJson()

```
std::string sound4::cloudx1::dyn::CInstance::CClient::ProcessJson (
            const std::string & request,
            bool * NeedSave = nullptr )  [inline]
```

Parameters

request	the JSON request.

See also

JSON syntax

Parameters

NeedSave	if non null, returns if the saving is required (maybe delayed)

Returns

the JSON answer

Definition at line 1296 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.5 sound4::cloudx1::dyn::CDynLib Class Reference

Dynamic library interface.

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

Public Member Functions

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

  Loads the library.
- bool IsOk () const

  Check if the library was loaded correctly.

Public Attributes

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

- helper::DynFuncHolder< decltype(cloudx1_TerminateProcess) > TerminateProcess
- helper::DynFuncHolder< decltype(cloudx1_ExitProcess) > ExitProcess
- helper::DynFuncHolder< decltype(cloudx1_StartUpdateThread) > StartUpdateThread
- helper::DynFuncHolder< decltype(cloudx1_StopUpdateThread) > StopUpdateThread
- helper::DynFuncHolder< decltype(cloudx1_WaitUpdateThreadReady) > WaitUpdateThreadReady
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio) > ProcessAudio
- helper::DynFuncHolder< decltype(cloudx1_ProcessAudio_Planar) > ProcessAudio_Planar
- helper::DynFuncHolder< decltype(cloudx1_GetBufferIn) > GetBufferIn
- helper::DynFuncHolder< decltype(cloudx1_GetBufferOut) > GetBufferOut
- helper::DynFuncHolder< decltype(cloudx1_GetEstimatedDelay) > GetEstimatedDelay
- helper::DynFuncHolder< decltype(cloudx1_GetFormatName) > GetFormatName
- helper::DynFuncHolder< decltype(cloudx1_GetFormatFromName) > GetFormatFromName
- helper::DynFuncHolder< decltype(cloudx1_GetBytesFromFormat) > GetBytesFromFormat
- helper::DynFuncHolder< decltype(cloudx1_GetMaxPacketFrame) > GetMaxPacketFrame
- helper::DynFuncHolder< decltype(cloudx1_AddAudio) > AddAudio
- helper::DynFuncHolder< decltype(cloudx1_AddPadAudio) > AddPadAudio
- helper::DynFuncHolder< decltype(cloudx1_GetOutputCount) > GetOutputCount
- helper::DynFuncHolder< decltype(cloudx1_GetAudio) > GetAudio
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertFrom) > AudioConvertFrom
- helper::DynFuncHolder< decltype(cloudx1_AudioConvertTo) > AudioConvertTo
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono) > StereoToMono
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo) > MonoToStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoFromLiveStereo) > AudioMonoFromLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_AudioMonoToLiveStereo) > AudioMonoToLiveStereo
- helper::DynFuncHolder< decltype(cloudx1_NewClient) > NewClient
- helper::DynFuncHolder< decltype(cloudx1_DeleteClient) > DeleteClient
- helper::DynFuncHolder< decltype(cloudx1_ProcessJson) > ProcessJson
- helper::DynFuncHolder< decltype(cloudx1_FreeJsonAnswer) > FreeJsonAnswer
- helper::DynFuncHolder< decltype(cloudx1_SaveState) > SaveState
- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp) > Webserver_tcp
- helper::DynFuncHolder< decltype(cloudx1_Webserver_tcp2) > Webserver_tcp2
- helper::DynFuncHolder< decltype(cloudx1_Webserver) > Webserver
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Stop) > Webserver_Stop
- helper::DynFuncHolder< decltype(cloudx1_Webserver_Status) > Webserver_Status
- helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar) > StereoToMono_Planar
- helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar) > MonoToStereo_Planar
- helper::DynFuncHolder< decltype(cloudx1_SetMetadata) > SetMetadata
- helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos) > GetMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos) > FreeMetadataInfos
- helper::DynFuncHolder< decltype(cloudx1_SetPresetManager) > SetPresetManager
- helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange) > PresetManager_InformChange
- helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames) > GetPossibleChunkSizeInFrames
- helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames) > GetProcessChunkFrames
- helper::DynFuncHolder< decltype(cloudx1_InitProcess3) > InitProcess3
- helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti) > SetMetadataMulti
- helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth) > Webserver_SetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth) > Webserver_GetAppHealth
- helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString) > Webserver_FreeString

10.5.1 Detailed Description

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

Definition at line 210 of file sound4.x1.cloud_dyn.hpp.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 CDynLib()

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

10.5.3 Member Function Documentation

10.5.3.1 IsOk()

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

Returns

true if OK.

false if not OK.

Definition at line 450 of file sound4.x1.cloud_dyn.hpp.

10.5.3.2 Load()

```
bool sound4::cloudx1::dyn::CDynLib::Load (
            const std::filesystem::path & path = {} ) [inline]
```

Parameters

path	the path of the library to load.

Returns

true if library was loaded correctly.

false if library load failed. Do not use in this case !

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

10.5.4 Member Data Documentation

10.5.4.1 AddAudio

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

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

10.5.4.2 AddPadAudio

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

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

10.5.4.3 AudioConvertFrom

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

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

10.5.4.4 AudioConvertTo

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

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

10.5.4.5 AudioMonoFromLiveStereo

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

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

10.5.4.6 AudioMonoToLiveStereo

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

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

10.5.4.7 DeleteClient

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

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

10.5.4.8 ExitProcess

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

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

10.5.4.9 FreeJsonAnswer

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

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

10.5.4.10 FreeMetadataInfos

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

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

10.5.4.11 FreeParameters

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

Definition at line 227 of file sound4.x1.cloud_dyn.hpp.

10.5.4.12 FreeParameterValue

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

Definition at line 230 of file sound4.x1.cloud_dyn.hpp.

10.5.4.13 GetAudio

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

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

10.5.4.14 GetAudioInputCount

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

Definition at line 221 of file sound4.x1.cloud_dyn.hpp.

10.5.4.15 GetAudioOutputCount

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

Definition at line 222 of file sound4.x1.cloud_dyn.hpp.

10.5.4.16 GetBufferIn

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

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

10.5.4.17 GetBufferOut

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

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

10.5.4.18 GetBytesFromFormat

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

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

10.5.4.19 GetChannelCount

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

Definition at line 220 of file sound4.x1.cloud_dyn.hpp.

10.5.4.20 GetChunkSizeInFrames

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

Definition at line 219 of file sound4.x1.cloud_dyn.hpp.

10.5.4.21 GetEstimatedDelay

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

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

10.5.4.22 GetFormatFromName

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

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

10.5.4.23 GetFormatName

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

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

10.5.4.24 GetMaxPacketFrame

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

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

10.5.4.25 GetMetadataInfos

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

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

10.5.4.26 GetOutputCount

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

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

10.5.4.27 GetParameter

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

Definition at line 229 of file sound4.x1.cloud_dyn.hpp.

10.5.4.28 GetPossibleChunkSizeInFrames

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

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

10.5.4.29 GetProcessChunkFrames

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

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

10.5.4.30 GetSampleRate

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

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

10.5.4.31 GetVersion

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

Definition at line 218 of file sound4.x1.cloud_dyn.hpp.

10.5.4.32 InitProcess

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

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

10.5.4.33 InitProcess2

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

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

10.5.4.34 InitProcess3

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

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

10.5.4.35 MonoToStereo

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

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

10.5.4.36 MonoToStereo_Planar

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

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

10.5.4.37 NewClient

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

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

10.5.4.38 NewParameters

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

Definition at line 226 of file sound4.x1.cloud_dyn.hpp.

10.5.4.39 PresetManager_InformChange

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

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

10.5.4.40 ProcessAudio

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

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

10.5.4.41 ProcessAudio_Planar

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

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

10.5.4.42 ProcessJson

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

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

10.5.4.43 SaveState

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

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

10.5.4.44 SetLoggerCallback

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

Definition at line 224 of file sound4.x1.cloud_dyn.hpp.

10.5.4.45 SetLogSeverity

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

Definition at line 225 of file sound4.x1.cloud_dyn.hpp.

10.5.4.46 SetMetadata

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

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

10.5.4.47 SetMetadataMulti

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

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

10.5.4.48 SetParameter

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

Definition at line 228 of file sound4.x1.cloud_dyn.hpp.

10.5.4.49 SetPresetManager

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

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

10.5.4.50 StartUpdateThread

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

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

10.5.4.51 StereoToMono

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

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

10.5.4.52 StereoToMono_Planar

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

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

10.5.4.53 StopUpdateThread

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

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

10.5.4.54 TerminateProcess

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

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

10.5.4.55 WaitUpdateThreadReady

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

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

10.5.4.56 Webserver

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

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

10.5.4.57 Webserver_FreeString

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

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

10.5.4.58 Webserver_GetAppHealth

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

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

10.5.4.59 Webserver_SetAppHealth

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

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

10.5.4.60 Webserver_Status

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

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

10.5.4.61 Webserver_Stop

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

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

10.5.4.62 Webserver_tcp

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

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

10.5.4.63 Webserver_tcp2

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

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

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

- sound4.x1.cloud_dyn.hpp

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

Helper to load dynamic library.

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

Public Member Functions

- CDynLoader ()=default
- CDynLoader (CDynLoader &&)=default
- CDynLoader (const CDynLoader &)=delete
- CDynLoader & operator= (CDynLoader const &)=delete
- operator bool () const
- bool IsOk () const
- bool Load (const std::filesystem::path &dynlib, uint32_t loadflags=LOAD_LIBRARY_SEARCH_DLL_LOAD↩ _DIR|LOAD_LIBRARY_SEARCH_APPLICATION_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_↩ LIBRARY_SEARCH_SYSTEM32)
- 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)

10.6.1 Detailed Description

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

10.6.2 Constructor & Destructor Documentation

10.6.2.1 CDynLoader() [1/3]

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

10.6.2.2 CDynLoader() [2/3]

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

10.6.2.3 CDynLoader() [3/3]

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

10.6.2.4 ~CDynLoader()

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

Definition at line 198 of file sound4.x1.cloud_dyn.hpp.

10.6.3 Member Function Documentation

10.6.3.1 Close()

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

Definition at line 159 of file sound4.x1.cloud_dyn.hpp.

10.6.3.2 GetSymbol()

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

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

10.6.3.3 GetSymbol_safe()

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

Definition at line 180 of file sound4.x1.cloud_dyn.hpp.

10.6.3.4 GetThisLibraryPath()

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

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

10.6.3.5 IsOk()

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

Definition at line 113 of file sound4.x1.cloud_dyn.hpp.

10.6.3.6 Load()

```
bool sound4::cloudx1::dyn::helper::CDynLoader::Load (
            const std::filesystem::path & dynlib,
            uint32_t loadflags = LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR|LOAD_LIBRARY_SEARCH_APPLICATION↩
_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_LIBRARY_SEARCH_SYSTEM32 )  [inline]
```

Definition at line 153 of file sound4.x1.cloud_dyn.hpp.

10.6.3.7 operator bool()

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

Definition at line 110 of file sound4.x1.cloud_dyn.hpp.

10.6.3.8 operator=()

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

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

- sound4.x1.cloud_dyn.hpp

10.7 sound4::helper::CDynLoader Class Reference

Helper to load dynamic library.

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

Public Member Functions

- CDynLoader ()=default
- CDynLoader (CDynLoader &&)=default
- CDynLoader (const CDynLoader &)=delete
- CDynLoader & operator= (CDynLoader const &)=delete
- operator bool () const
- bool IsOk () const
- bool Load (const std::filesystem::path &dynlib, uint32_t loadflags=LOAD_LIBRARY_SEARCH_DLL_LOAD↩_DIR|LOAD_LIBRARY_SEARCH_APPLICATION_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_↩LIBRARY_SEARCH_SYSTEM32)
- 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)

10.7.1 Detailed Description

Definition at line 130 of file sound4cl.hpp.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 CDynLoader() [1/3]

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

10.7.2.2 CDynLoader() [2/3]

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

10.7.2.3 CDynLoader() [3/3]

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

10.7.2.4 ~CDynLoader()

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

Definition at line 227 of file sound4cl.hpp.

10.7.3 Member Function Documentation

10.7.3.1 Close()

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

Definition at line 188 of file sound4cl.hpp.

10.7.3.2 GetSymbol()

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

Definition at line 201 of file sound4cl.hpp.

10.7.3.3 GetSymbol_safe()

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

Definition at line 209 of file sound4cl.hpp.

10.7.3.4 GetThisLibraryPath()

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

Definition at line 213 of file sound4cl.hpp.

10.7.3.5 IsOk()

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

Definition at line 142 of file sound4cl.hpp.

10.7.3.6 Load()

```
bool sound4::helper::CDynLoader::Load (
            const std::filesystem::path & dynlib,
            uint32_t loadflags = LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR|LOAD_LIBRARY_SEARCH_APPLICATION↩
_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_LIBRARY_SEARCH_SYSTEM32 ) [inline]
```

Definition at line 182 of file sound4cl.hpp.

10.7.3.7 operator bool()

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

Definition at line 139 of file sound4cl.hpp.

10.7.3.8 operator=()

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

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

- sound4cl.hpp

10.8 sound4::CInstance Class Reference

Instance handling class.

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

Classes

- class CClient

Public Member Functions

- CInstance (CProcessor &dynlib)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- ~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 ()
- 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 ()

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)

10.8.1 Detailed Description

Holds an instance of processing

Definition at line 750 of file sound4cl.hpp.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 755 of file sound4cl.hpp.

10.8.2.2 CInstance() [2/2]

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

10.8.2.3 ~CInstance()

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

Destructor, stopping any running tasks

Definition at line 767 of file sound4cl.hpp.

10.8.3 Member Function Documentation

10.8.3.1 AddAudio()

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

Definition at line 1273 of file sound4cl.hpp.

10.8.3.2 Create()

```
bool sound4::CInstance::Create (
            const std::string & LoginKey,
            const std::string & RadioName,
            const std::string & Access_Key_ID,
            const std::string & Access_Key_Secret,
            const std::filesystem::path & save_path,
            int json_port = 0,
            unsigned int frames_per_chunk = 0 )  [inline]
```

Creates and initialize the instance, and starts its update thread

See also

sound4cl_InitProcess2

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 sound4cl_GetPossibleChunkSizeInFrames, or 0 to set default

Returns

false if failed

Definition at line 854 of file sound4cl.hpp.

10.8.3.3 GetAudio()

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

Definition at line 1279 of file sound4cl.hpp.

10.8.3.4 GetBufferIn()

```
float* sound4::CInstance::GetBufferIn ( )  [inline]
```

Gets the input buffer used in ProcessAudio

Input has GetAudioInputCount() ∗ 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]

Definition at line 1128 of file sound4cl.hpp.

10.8.3.5 GetBufferOut()

```
float* sound4::CInstance::GetBufferOut ( ) [inline]
```

Gets the output buffer used in ProcessAudio

Output has GetAudioOutputCount() ∗ 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]

Definition at line 1144 of file sound4cl.hpp.

10.8.3.6 GetChunkFrames()

```
unsigned int sound4::CInstance::GetChunkFrames ( ) [inline]
```

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

The value reflects the frames_per_chunk parameter passed to Create

Returns

the number of audio frames in each chunk of processing

Definition at line 900 of file sound4cl.hpp.

10.8.3.7 GetEstimatedDelay()

```
unsigned int sound4::CInstance::GetEstimatedDelay ( ) [inline]
```

Gets an estimated delay of current processing

See also

sound4cl_GetEstimatedDelay

Definition at line 1112 of file sound4cl.hpp.

10.8.3.8 GetMetadataInfos()

```
std::vector< std::tuple<std::string,std::string> > sound4::CInstance::GetMetadataInfos ( )
[inline]
```

Get information on known metadata keys.

Returns

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

Definition at line 989 of file sound4cl.hpp.

10.8.3.9 GetParam()

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

Gets a parameter value

See also

sound4cl_GetParameter

Definition at line 793 of file sound4cl.hpp.

10.8.3.10 GetWebServerAppHealth()

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

Get the web server /apphealth status info

Definition at line 1066 of file sound4cl.hpp.

10.8.3.11 IsOk()

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

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

Definition at line 775 of file sound4cl.hpp.

10.8.3.12 NewClient()

```
std::shared_ptr<CClient> sound4::CInstance::NewClient ( ) [inline]
```

Creates a low-level JSON client

See also

JSON syntax

Returns

a new client to communicate with the processing

Definition at line 1265 of file sound4cl.hpp.

10.8.3.13 OnUpdateThreadStart()

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

Definition at line 1011 of file sound4cl.hpp.

10.8.3.14 OnUpdateThreadStop()

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

Definition at line 1013 of file sound4cl.hpp.

10.8.3.15 operator=()

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

10.8.3.16 PresetManager_InformChange()

```
void sound4::CInstance::PresetManager_InformChange (
            const std::filesystem::path & relative_path,
            PresetChange_Kind change_kind ) [inline]
```

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

Parameters

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

Definition at line 914 of file sound4cl.hpp.

10.8.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector<T> sound4::CInstance::ProcessAnyAudio (
            const std::vector< T > input )  [inline]
```

Process audio (different formats, less optimized)

Supports for int16_t, int32_t and float (32) in native endianness

Attention

the return count is independent from the input count

Template Parameters

T	the C type for one audio sample

Parameters

input	audio to process

Returns

processed audio

Definition at line 1191 of file sound4cl.hpp.

10.8.3.18 ProcessAudio()

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

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

See also

sound4cl_ProcessAudio

Definition at line 1154 of file sound4cl.hpp.

10.8.3.19 ProcessAudio_Planar()

```
void sound4::CInstance::ProcessAudio_Planar (
            float const *const * input,
            float *const * output )  [inline]
```

Process one audio chunk, but with planar audio

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

See also

sound4cl_ProcessAudio_Planar

Parameters

input	has GetAudioInputCount() ∗ GetChannelCount() channels
output	has GetAudioOutputCount() ∗ GetChannelCount() channels

Definition at line 1173 of file sound4cl.hpp.

10.8.3.20 SetBus()

```
void sound4::CInstance::SetBus (
            const CBus & bus )  [inline]
```

Set the instance bus

Definition at line 805 of file sound4cl.hpp.

10.8.3.21 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 932 of file sound4cl.hpp.

10.8.3.22 SetMetadataMulti()

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

Adds multiple metadata to the process.

This works like CInstance::SetMetadata but with multiple metadata at once.

See also

CInstance::SetMetadata

Parameters

list	a map of key/value pair.

Definition at line 953 of file sound4cl.hpp.

10.8.3.23 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

sound4cl_SetParameter

Definition at line 783 of file sound4cl.hpp.

10.8.3.24 SetPresetManager()

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

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

See also

sound4cl_SetPresetManager

Must be done before Create

Parameters

preset_manager	The custom preset manager

Definition at line 821 of file sound4cl.hpp.

10.8.3.25 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 1057 of file sound4cl.hpp.

10.8.3.26 StartWebServer()

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

Starts a Web server on requested ports

See also

sound4cl_Webserver

Definition at line 1020 of file sound4cl.hpp.

10.8.3.27 Stop()

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

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

See also

sound4cl_ExitProcess,
sound4cl_StopUpdateThread and
sound4cl_Webserver_Stop

Definition at line 1095 of file sound4cl.hpp.

10.8.3.28 StopWebServer()

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

Stops the Web server

See also

sound4cl_Webserver_Stop

Definition at line 1045 of file sound4cl.hpp.

10.8.3.29 Terminate()

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

Terminate the processing. This is useful only with file processing, to get the final status.

See also

sound4cl_TerminateProcess

Definition at line 1084 of file sound4cl.hpp.

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

- sound4cl.hpp

10.9 sound4::cloudx1::CInstance Class Reference

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

Classes

- class CClient

Public Member Functions

- CInstance ()
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- ~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 &AWS_Access_ID, const std::string &AWS_Access_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 ()
- 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 ()

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)

10.9.1 Detailed Description

Instance handling class

Definition at line 385 of file sound4.x1.cloud.hpp.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 390 of file sound4.x1.cloud.hpp.

10.9.2.2 CInstance() [2/2]

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

10.9.2.3 ~CInstance()

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

Destructor, stopping any running tasks

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

10.9.3 Member Function Documentation

10.9.3.1 AddAudio()

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

Definition at line 883 of file sound4.x1.cloud.hpp.

10.9.3.2 Create()

```
bool sound4::cloudx1::CInstance::Create (
            const std::string & LoginKey,
            const std::string & RadioName,
            const std::string & AWS_Access_ID,
            const std::string & AWS_Access_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

See also

cloudx1_InitProcess2

Parameters

LoginKey	SOUND4 Login key
RadioName	SOUND4 RadioName
AWS_Access_ID	SOUND4 AWS access ID
AWS_Access_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 487 of file sound4.x1.cloud.hpp.

10.9.3.3 GetAudio()

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

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

10.9.3.4 GetBufferIn()

```
std::array<float,InputSampleSize>& sound4::cloudx1::CInstance::GetBufferIn ( )  [inline]
```

Gets the input buffer used in ProcessAudio

Input has GetAudioInputCount() ∗ 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]

Definition at line 738 of file sound4.x1.cloud.hpp.

10.9.3.5 GetBufferOut()

```
std::array<float,OutputSampleSize>& sound4::cloudx1::CInstance::GetBufferOut ( ) [inline]
```

Gets the output buffer used in ProcessAudio

Output has GetAudioOutputCount() ∗ 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]

Definition at line 755 of file sound4.x1.cloud.hpp.

10.9.3.6 GetChunkFrames()

```
unsigned int sound4::cloudx1::CInstance::GetChunkFrames ( ) [inline]
```

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

The value reflects the frames_per_chunk parameter passed to Create

Returns

the number of audio frames in each chunk of processing

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

10.9.3.7 GetEstimatedDelay()

```
unsigned int sound4::cloudx1::CInstance::GetEstimatedDelay ( ) [inline]
```

Gets an estimated delay of current processing

See also

cloudx1_GetEstimatedDelay

Definition at line 722 of file sound4.x1.cloud.hpp.

10.9.3.8 GetMetadataInfos()

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

Get information on known metadata keys.

Returns

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

Definition at line 602 of file sound4.x1.cloud.hpp.

10.9.3.9 GetParam()

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

Gets a parameter value

See also

cloudx1_GetParameter

Definition at line 427 of file sound4.x1.cloud.hpp.

10.9.3.10 GetWebServerAppHealth()

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

Get the web server /apphealth status info

Definition at line 673 of file sound4.x1.cloud.hpp.

10.9.3.11 IsOk()

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

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

Definition at line 409 of file sound4.x1.cloud.hpp.

10.9.3.12 NewClient()

```
std::shared_ptr<CClient> sound4::cloudx1::CInstance::NewClient ( ) [inline]
```

Creates a low-level JSON client

See also

JSON syntax

Returns

a new client to communicate with the processing

Definition at line 875 of file sound4.x1.cloud.hpp.

10.9.3.13 OnUpdateThreadStart()

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

Definition at line 622 of file sound4.x1.cloud.hpp.

10.9.3.14 OnUpdateThreadStop()

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

Definition at line 624 of file sound4.x1.cloud.hpp.

10.9.3.15 operator=()

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

10.9.3.16 PresetManager_InformChange()

```
void sound4::cloudx1::CInstance::PresetManager_InformChange (
            const std::filesystem::path & relative_path,
            PresetChange_Kind change_kind ) [inline]
```

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

Parameters

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

Definition at line 538 of file sound4.x1.cloud.hpp.

10.9.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector<T> sound4::cloudx1::CInstance::ProcessAnyAudio (
            const std::vector< T > input )  [inline]
```

Process audio (different formats, less optimized)

Supports for int16_t, int32_t and float (32) in native endianness

Attention

the return count is independent from the input count due to CLOUDX1_AUDIOFRAME_COUNT

Template Parameters

T	the C type for one audio sample

Parameters

input	audio to process

Returns

processed audio

Definition at line 803 of file sound4.x1.cloud.hpp.

10.9.3.18 ProcessAudio()

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

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

See also

cloudx1_ProcessAudio

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

10.9.3.19 ProcessAudio_Planar()

```
void sound4::cloudx1::CInstance::ProcessAudio_Planar (
            float const *const * input,
            float *const * output )  [inline]
```

Process one audio chunk, but with planar audio

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

See also

cloudx1_ProcessAudio_Planar

Parameters

input	has GetAudioInputCount() ∗ GetChannelCount() channels
output	has GetAudioOutputCount() ∗ GetChannelCount() channels

Definition at line 785 of file sound4.x1.cloud.hpp.

10.9.3.20 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 554 of file sound4.x1.cloud.hpp.

10.9.3.21 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 CInstance::SetMetadata but with multiple metadata at once.

See also

CInstance::SetMetadata

Parameters

list	a map of key/value pair.

Definition at line 572 of file sound4.x1.cloud.hpp.

10.9.3.22 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

cloudx1_SetParameter

Definition at line 417 of file sound4.x1.cloud.hpp.

10.9.3.23 SetPresetManager()

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

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

See also

cloudx1_SetPresetManager

Must be done before Create

Parameters

preset_manager	The custom preset manager

Definition at line 445 of file sound4.x1.cloud.hpp.

10.9.3.24 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 665 of file sound4.x1.cloud.hpp.

10.9.3.25 StartWebServer()

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

Starts a Web server on requested ports

See also

cloudx1_Webserver

Definition at line 632 of file sound4.x1.cloud.hpp.

10.9.3.26 Stop()

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

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

See also

cloudx1_ExitProcess,
cloudx1_StopUpdateThread and
cloudx1_Webserver_Stop

Definition at line 703 of file sound4.x1.cloud.hpp.

10.9.3.27 StopWebServer()

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

Stops the Web server

See also

cloudx1_Webserver_Stop

Definition at line 654 of file sound4.x1.cloud.hpp.

10.9.3.28 Terminate()

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

Terminate the processing. This is useful only with file processing, to get the final status.

See also

cloudx1_TerminateProcess

Definition at line 692 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

10.10 sound4::cloudx1::dyn::CInstance Class Reference

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

Classes

- class CClient

Public Member Functions

- CInstance (CDynLib &dynlib)
- CInstance (const CInstance &)=delete
- CInstance & operator= (const CInstance &)=delete
- ~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 ()
- 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 ()

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)

10.10.1 Detailed Description

Instance handling class

Definition at line 804 of file sound4.x1.cloud_dyn.hpp.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 CInstance() [1/2]

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

Creates the instance, with default parameters

Definition at line 809 of file sound4.x1.cloud_dyn.hpp.

10.10.2.2 CInstance() [2/2]

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

10.10.2.3 ~CInstance()

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

Destructor, stopping any running tasks

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

10.10.3 Member Function Documentation

10.10.3.1 AddAudio()

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

Definition at line 1328 of file sound4.x1.cloud_dyn.hpp.

10.10.3.2 Create()

```
bool sound4::cloudx1::dyn::CInstance::Create (
            const std::string & LoginKey,
            const std::string & RadioName,
            const std::string & Access_Key_ID,
            const std::string & Access_Key_Secret,
            const std::filesystem::path & save_path,
            int json_port = 0,
            unsigned int frames_per_chunk = 64 )  [inline]
```

Creates and initialize the instance, and starts its update thread

See also

cloudx1_InitProcess2

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 909 of file sound4.x1.cloud_dyn.hpp.

10.10.3.3 GetAudio()

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

Definition at line 1334 of file sound4.x1.cloud_dyn.hpp.

10.10.3.4 GetBufferIn()

```
std::array<float,InputSampleSize>& sound4::cloudx1::dyn::CInstance::GetBufferIn ( )  [inline]
```

Gets the input buffer used in ProcessAudio

Input has GetAudioInputCount() ∗ 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]

Definition at line 1181 of file sound4.x1.cloud_dyn.hpp.

10.10.3.5 GetBufferOut()

```
std::array<float,OutputSampleSize>& sound4::cloudx1::dyn::CInstance::GetBufferOut ( ) [inline]
```

Gets the output buffer used in ProcessAudio

Output has GetAudioOutputCount() ∗ 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]

Definition at line 1198 of file sound4.x1.cloud_dyn.hpp.

10.10.3.6 GetChunkFrames()

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

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

The value reflects the frames_per_chunk parameter passed to Create

Returns

the number of audio frames in each chunk of processing

Definition at line 952 of file sound4.x1.cloud_dyn.hpp.

10.10.3.7 GetEstimatedDelay()

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

Gets an estimated delay of current processing

See also

cloudx1_GetEstimatedDelay

Definition at line 1165 of file sound4.x1.cloud_dyn.hpp.

10.10.3.8 GetMetadataInfos()

```
std::vector< std::tuple<std::string,std::string> > sound4::cloudx1::dyn::CInstance::Get↩
MetadataInfos ( ) [inline]
```

Get information on known metadata keys.

Returns

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

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

10.10.3.9 GetParam()

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

Gets a parameter value

See also

cloudx1_GetParameter

Definition at line 847 of file sound4.x1.cloud_dyn.hpp.

10.10.3.10 GetWebServerAppHealth()

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

Get the web server /apphealth status info

Definition at line 1115 of file sound4.x1.cloud_dyn.hpp.

10.10.3.11 IsOk()

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

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

Definition at line 829 of file sound4.x1.cloud_dyn.hpp.

10.10.3.12 NewClient()

```
std::shared_ptr<CClient> sound4::cloudx1::dyn::CInstance::NewClient ( ) [inline]
```

Creates a low-level JSON client

See also

JSON syntax

Returns

a new client to communicate with the processing

Definition at line 1320 of file sound4.x1.cloud_dyn.hpp.

10.10.3.13 OnUpdateThreadStart()

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

Definition at line 1063 of file sound4.x1.cloud_dyn.hpp.

10.10.3.14 OnUpdateThreadStop()

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

Definition at line 1065 of file sound4.x1.cloud_dyn.hpp.

10.10.3.15 operator=()

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

10.10.3.16 PresetManager_InformChange()

```
void sound4::cloudx1::dyn::CInstance::PresetManager_InformChange (
            const std::filesystem::path & relative_path,
            PresetChange_Kind change_kind ) [inline]
```

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

Parameters

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

Definition at line 966 of file sound4.x1.cloud_dyn.hpp.

10.10.3.17 ProcessAnyAudio()

```
template<typename T >
std::vector<T> sound4::cloudx1::dyn::CInstance::ProcessAnyAudio (
            const std::vector< T > input )  [inline]
```

Process audio (different formats, less optimized)

Supports for int16_t, int32_t and float (32) in native endianness

Attention

the return count is independent from the input count due to CLOUDX1_AUDIOFRAME_COUNT

Template Parameters

T	the C type for one audio sample

Parameters

input	audio to process

Returns

processed audio

Definition at line 1246 of file sound4.x1.cloud_dyn.hpp.

10.10.3.18 ProcessAudio()

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

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

See also

cloudx1_ProcessAudio

Definition at line 1209 of file sound4.x1.cloud_dyn.hpp.

10.10.3.19 ProcessAudio_Planar()

```
void sound4::cloudx1::dyn::CInstance::ProcessAudio_Planar (
            float const *const * input,
            float *const * output )  [inline]
```

Process one audio chunk, but with planar audio

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

See also

cloudx1_ProcessAudio_Planar

Parameters

input	has GetAudioInputCount() ∗ GetChannelCount() channels
output	has GetAudioOutputCount() ∗ GetChannelCount() channels

Definition at line 1228 of file sound4.x1.cloud_dyn.hpp.

10.10.3.20 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 984 of file sound4.x1.cloud_dyn.hpp.

10.10.3.21 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 CInstance::SetMetadata but with multiple metadata at once.

See also

CInstance::SetMetadata

Parameters

list	a map of key/value pair.

Definition at line 1005 of file sound4.x1.cloud_dyn.hpp.

10.10.3.22 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

cloudx1_SetParameter

Definition at line 837 of file sound4.x1.cloud_dyn.hpp.

10.10.3.23 SetPresetManager()

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

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

See also

cloudx1_SetPresetManager

Must be done before Create

Parameters

preset_manager	The custom preset manager

Definition at line 876 of file sound4.x1.cloud_dyn.hpp.

10.10.3.24 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 1106 of file sound4.x1.cloud_dyn.hpp.

10.10.3.25 StartWebServer()

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

Starts a Web server on requested ports

See also

cloudx1_Webserver

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

10.10.3.26 Stop()

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

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

See also

cloudx1_ExitProcess,
cloudx1_StopUpdateThread and
cloudx1_Webserver_Stop

Definition at line 1146 of file sound4.x1.cloud_dyn.hpp.

10.10.3.27 StopWebServer()

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

Stops the Web server

See also

cloudx1_Webserver_Stop

Definition at line 1095 of file sound4.x1.cloud_dyn.hpp.

10.10.3.28 Terminate()

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

Terminate the processing. This is useful only with file processing, to get the final status.

See also

cloudx1_TerminateProcess

Definition at line 1135 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.11 cloudx1_CClientInstance Struct Reference

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

10.11.1 Detailed Description

Abstract pointer for a client instance

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

- sound4.x1.cloud.h

10.12 cloudx1_CInstance Struct Reference

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

10.12.1 Detailed Description

Abstract holder for a process instance

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

- sound4.x1.cloud.h

10.13 cloudx1_CParameters Struct Reference

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

10.13.1 Detailed Description

Abstract holder for a process instance parameters

Definition at line 332 of file sound4.x1.cloud.h.

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

- sound4.x1.cloud.h

10.14 sound4::cloudx1::CPresetLoader Class Reference

```
#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

10.14.1 Detailed Description

Custom preset handler helper Derive from this class and call CInstance::SetPresetManager

Definition at line 367 of file sound4.x1.cloud.hpp.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 CPresetLoader()

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

10.14.2.2 ∼CPresetLoader()

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

10.14.3 Member Function Documentation

10.14.3.1 Exists()

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

10.14.3.2 GetAll()

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

10.14.3.3 IsReadOnly()

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

10.14.3.4 Read()

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

10.14.3.5 Remove()

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

10.14.3.6 Rename()

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

10.14.3.7 Write()

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

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

- sound4.x1.cloud.hpp

10.15 sound4::cloudx1::dyn::CPresetLoader Class Reference

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

Inherited by MyPresetManager.

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

10.15.1 Detailed Description

Custom preset handler helper Derive from this class and call CInstance::SetPresetManager

Definition at line 786 of file sound4.x1.cloud_dyn.hpp.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 CPresetLoader()

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

10.15.2.2 ~CPresetLoader()

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

10.15.3 Member Function Documentation

10.15.3.1 Exists()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.2 GetAll()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.3 IsReadOnly()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.4 Read()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.5 Remove()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.6 Rename()

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

Implemented in MyPresetManager, and MyPresetManager.

10.15.3.7 Write()

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

Implemented in MyPresetManager, and MyPresetManager.

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

- sound4.x1.cloud_dyn.hpp

10.16 sound4::CPresetLoader Class Reference

Custom preset handler helper.

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

Inherited by MyPresetManager.

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

10.16.1 Detailed Description

Derive from this class and call CInstance::SetPresetManager

Definition at line 730 of file sound4cl.hpp.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 CPresetLoader()

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

10.16.2.2 ~CPresetLoader()

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

10.16.3 Member Function Documentation

10.16.3.1 Exists()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.2 GetAll()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.3 IsReadOnly()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.4 Read()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.5 Remove()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.6 Rename()

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

Implemented in MyPresetManager, and MyPresetManager.

10.16.3.7 Write()

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

Implemented in MyPresetManager, and MyPresetManager.

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

- sound4cl.hpp

10.17 sound4::CProcessor Class Reference

Dynamic library interface.

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

Public Member Functions

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

  Loads the library.
- bool IsOk () const

  Check if the library was loaded correctly.
- void AudioConvertFrom (const uint8_t ∗payload, float ∗output, size_t nSpl, sound4cl_SampleFormat fmt)
- void AudioConvertTo (const float ∗input, uint8_t ∗payload, size_t nSpl, sound4cl_SampleFormat fmt)
- void StereoToMono (const float ∗input, float ∗output, size_t nSpl)
- void MonoToStereo (const float ∗input, float ∗output, size_t nSpl)
- void StereoToMono_Planar (const float ∗inputL, const float ∗inputR, float ∗output, size_t nSpl)
- void MonoToStereo_Planar (const float ∗input, float ∗outputL, float ∗outputR, size_t nSpl)
- 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 sound4cl_SampleFormat fmt)
- sound4cl_SampleFormat GetFormatFromName (const std::string &name)
- unsigned int GetBytesFromFormat (const sound4cl_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)

Protected Attributes

- helper::DynFuncHolder< decltype(sound4cl_GetVersion) > fnGetVersion
- helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames) > fnGetChunkSizeInFrames
- helper::DynFuncHolder< decltype(sound4cl_GetChannelCount) > fnGetChannelCount
- helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount) > fnGetAudioInputCount
- helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount) > fnGetAudioOutputCount
- helper::DynFuncHolder< decltype(sound4cl_GetSampleRate) > fnGetSampleRate
- helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback) > fnSetLoggerCallback
- helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity) > fnSetLogSeverity
- helper::DynFuncHolder< decltype(sound4cl_NewParameters) > fnNewParameters
- helper::DynFuncHolder< decltype(sound4cl_FreeParameters) > fnFreeParameters
- helper::DynFuncHolder< decltype(sound4cl_SetParameter) > fnSetParameter
- helper::DynFuncHolder< decltype(sound4cl_GetParameter) > fnGetParameter
- helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue) > fnFreeParameterValue
- helper::DynFuncHolder< decltype(sound4cl_InitProcess) > fnInitProcess
- helper::DynFuncHolder< decltype(sound4cl_InitProcess2) > fnInitProcess2
- helper::DynFuncHolder< decltype(sound4cl_TerminateProcess) > fnTerminateProcess
- helper::DynFuncHolder< decltype(sound4cl_ExitProcess) > fnExitProcess
- helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread) > fnStartUpdateThread
- helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread) > fnStopUpdateThread
- helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady) > fnWaitUpdateThreadReady
- helper::DynFuncHolder< decltype(sound4cl_ProcessAudio) > fnProcessAudio
- helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar) > fnProcessAudio_Planar
- helper::DynFuncHolder< decltype(sound4cl_GetBufferIn) > fnGetBufferIn
- helper::DynFuncHolder< decltype(sound4cl_GetBufferOut) > fnGetBufferOut
- helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay) > fnGetEstimatedDelay
- helper::DynFuncHolder< decltype(sound4cl_GetFormatName) > fnGetFormatName
- helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName) > fnGetFormatFromName
- helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat) > fnGetBytesFromFormat
- helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame) > fnGetMaxPacketFrame
- helper::DynFuncHolder< decltype(sound4cl_AddAudio) > fnAddAudio
- helper::DynFuncHolder< decltype(sound4cl_AddPadAudio) > fnAddPadAudio
- helper::DynFuncHolder< decltype(sound4cl_GetOutputCount) > fnGetOutputCount
- helper::DynFuncHolder< decltype(sound4cl_GetAudio) > fnGetAudio
- helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom) > fnAudioConvertFrom
- helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo) > fnAudioConvertTo
- helper::DynFuncHolder< decltype(sound4cl_StereoToMono) > fnStereoToMono
- helper::DynFuncHolder< decltype(sound4cl_MonoToStereo) > fnMonoToStereo
- helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo
- helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo) > fnAudioMonoToLiveStereo
- helper::DynFuncHolder< decltype(sound4cl_NewClient) > fnNewClient
- helper::DynFuncHolder< decltype(sound4cl_DeleteClient) > fnDeleteClient
- helper::DynFuncHolder< decltype(sound4cl_ProcessJson) > fnProcessJson
- helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer) > fnFreeJsonAnswer
- helper::DynFuncHolder< decltype(sound4cl_SaveState) > fnSaveState
- helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp) > fnWebserver_tcp
- helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2) > fnWebserver_tcp2
- helper::DynFuncHolder< decltype(sound4cl_Webserver) > fnWebserver
- helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop) > fnWebserver_Stop
- helper::DynFuncHolder< decltype(sound4cl_Webserver_Status) > fnWebserver_Status
- helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar) > fnStereoToMono_Planar
- helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar) > fnMonoToStereo_Planar
- helper::DynFuncHolder< decltype(sound4cl_SetMetadata) > fnSetMetadata
- helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos) > fnGetMetadataInfos

- helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos) > fnFreeMetadataInfos
- helper::DynFuncHolder< decltype(sound4cl_SetPresetManager) > fnSetPresetManager
- helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange) > fnPresetManager_InformChange
- helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames) > fnGetPossibleChunkSizeInFrames
- helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames) > fnGetProcessChunkFrames
- helper::DynFuncHolder< decltype(sound4cl_InitProcess3) > fnInitProcess3
- helper::DynFuncHolder< decltype(sound4cl_NewBus) > fnNewBus
- helper::DynFuncHolder< decltype(sound4cl_FreeBus) > fnFreeBus
- helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus) > fnSetInstanceBus
- helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti) > fnSetMetadataMulti
- helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth) > fnWebserver_SetAppHealth
- helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth) > fnWebserver_GetAppHealth
- helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString) > fnWebserver_FreeString

Friends

- class CBus
- class CInstance

10.17.1 Detailed Description

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

Definition at line 270 of file sound4cl.hpp.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 CProcessor()

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

10.17.3 Member Function Documentation

10.17.3.1 AudioConvertFrom()

```
void sound4::CProcessor::AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            sound4cl_SampleFormat fmt ) [inline]
```

Conversion helpers

Definition at line 563 of file sound4cl.hpp.

10.17.3.2 AudioConvertTo()

```
void sound4::CProcessor::AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            sound4cl_SampleFormat fmt )  [inline]
```

Definition at line 565 of file sound4cl.hpp.

10.17.3.3 AudioMonoFromLiveStereo()

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

Definition at line 575 of file sound4cl.hpp.

10.17.3.4 AudioMonoToLiveStereo()

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

Definition at line 577 of file sound4cl.hpp.

10.17.3.5 GetAudioInputCount()

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

Get the number of inputs for this process

Remarks

All inputs have sound4cl_GetChannelCount() channels

Returns

The process input count

Definition at line 635 of file sound4cl.hpp.

10.17.3.6 GetAudioOutputCount()

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

Get the number of outputs for this process

Remarks

All outputs have sound4cl_GetChannelCount() channels

Returns

The process outputs count

Definition at line 645 of file sound4cl.hpp.

10.17.3.7 GetBytesFromFormat()

```
unsigned int sound4::CProcessor::GetBytesFromFormat (
            const sound4cl_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 682 of file sound4cl.hpp.

10.17.3.8 GetChannelCount()

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

Get the number of channels needed by this process

Returns

The process channels count

Definition at line 625 of file sound4cl.hpp.

10.17.3.9 GetChunkSizeInFrames()

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

Get the 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 sound4cl_GetChannelCount samples

Returns

The process chunk size in frame

Definition at line 596 of file sound4cl.hpp.

10.17.3.10 GetFormatFromName()

```
sound4cl_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 673 of file sound4cl.hpp.

10.17.3.11 GetFormatName()

```
std::string sound4::CProcessor::GetFormatName (
            const sound4cl_SampleFormat fmt ) [inline]
```

Convert format to string

Parameters

fmt	the format

Returns

the string name of the format

Definition at line 664 of file sound4cl.hpp.

10.17.3.12 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 sound4cl_GetChannelCount samples

Returns

An vector of possible process chunk size in frame

Definition at line 607 of file sound4cl.hpp.

10.17.3.13 GetPrefixSymbol()

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

Definition at line 364 of file sound4cl.hpp.

10.17.3.14 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 655 of file sound4cl.hpp.

10.17.3.15 GetVersion()

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

Get the process version string

Returns

version string

Definition at line 585 of file sound4cl.hpp.

10.17.3.16 IsOk()

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

Returns

true if OK.

false if not OK.

Definition at line 555 of file sound4cl.hpp.

10.17.3.17 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 379 of file sound4cl.hpp.

10.17.3.18 MonoToStereo()

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

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

Definition at line 569 of file sound4cl.hpp.

10.17.3.19 MonoToStereo_Planar()

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

Definition at line 573 of file sound4cl.hpp.

10.17.3.20 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 694 of file sound4cl.hpp.

10.17.3.21 SetLogSeverity()

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

Sets the log severity (global)

Definition at line 688 of file sound4cl.hpp.

10.17.3.22 StereoToMono()

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

Definition at line 567 of file sound4cl.hpp.

10.17.3.23 StereoToMono_Planar()

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

Definition at line 571 of file sound4cl.hpp.

10.17.4 Friends And Related Function Documentation

10.17.4.1 CBus

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

Definition at line 276 of file sound4cl.hpp.

10.17.4.2 CInstance

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

Definition at line 277 of file sound4cl.hpp.

10.17.5 Member Data Documentation

10.17.5.1 fnAddAudio

```
helper::DynFuncHolder< decltype(sound4cl_AddAudio ) > sound4::CProcessor::fnAddAudio  [protected]
```

Definition at line 309 of file sound4cl.hpp.

10.17.5.2 fnAddPadAudio

```
helper::DynFuncHolder< decltype(sound4cl_AddPadAudio ) > sound4::CProcessor::fnAddPadAudio
[protected]
```

Definition at line 310 of file sound4cl.hpp.

10.17.5.3 fnAudioConvertFrom

```
helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom ) > sound4::CProcessor::fnAudio↩
ConvertFrom [protected]
```

Definition at line 313 of file sound4cl.hpp.

10.17.5.4 fnAudioConvertTo

```
helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo ) > sound4::CProcessor::fnAudio↩
ConvertTo [protected]
```

Definition at line 314 of file sound4cl.hpp.

10.17.5.5 fnAudioMonoFromLiveStereo

```
helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > sound4::CProcessor::fn↩
AudioMonoFromLiveStereo [protected]
```

Definition at line 317 of file sound4cl.hpp.

10.17.5.6 fnAudioMonoToLiveStereo

```
helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo ) > sound4::CProcessor::fn↩
AudioMonoToLiveStereo [protected]
```

Definition at line 318 of file sound4cl.hpp.

10.17.5.7 fnDeleteClient

```
helper::DynFuncHolder< decltype(sound4cl_DeleteClient ) > sound4::CProcessor::fnDeleteClient
[protected]
```

Definition at line 320 of file sound4cl.hpp.

10.17.5.8 fnExitProcess

```
helper::DynFuncHolder< decltype(sound4cl_ExitProcess ) > sound4::CProcessor::fnExitProcess
[protected]
```

Definition at line 296 of file sound4cl.hpp.

10.17.5.9 fnFreeBus

```
helper::DynFuncHolder< decltype(sound4cl_FreeBus ) > sound4::CProcessor::fnFreeBus [protected]
```

Definition at line 352 of file sound4cl.hpp.

10.17.5.10 fnFreeJsonAnswer

```
helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer ) > sound4::CProcessor::fnFreeJson↩
Answer [protected]
```

Definition at line 322 of file sound4cl.hpp.

10.17.5.11 fnFreeMetadataInfos

```
helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos ) > sound4::CProcessor::fnFree↩
MetadataInfos [protected]
```

Definition at line 339 of file sound4cl.hpp.

10.17.5.12 fnFreeParameters

```
helper::DynFuncHolder< decltype(sound4cl_FreeParameters ) > sound4::CProcessor::fnFreeParameters
[protected]
```

Definition at line 289 of file sound4cl.hpp.

10.17.5.13 fnFreeParameterValue

```
helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue ) > sound4::CProcessor::fnFree↩
ParameterValue [protected]
```

Definition at line 292 of file sound4cl.hpp.

10.17.5.14 fnGetAudio

```
helper::DynFuncHolder< decltype(sound4cl_GetAudio ) > sound4::CProcessor::fnGetAudio [protected]
```

Definition at line 312 of file sound4cl.hpp.

10.17.5.15 fnGetAudioInputCount

```
helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount ) > sound4::CProcessor::fnGet↩
AudioInputCount [protected]
```

Definition at line 283 of file sound4cl.hpp.

10.17.5.16 fnGetAudioOutputCount

```
helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount ) > sound4::CProcessor::fnGet↩
AudioOutputCount [protected]
```

Definition at line 284 of file sound4cl.hpp.

10.17.5.17 fnGetBufferIn

```
helper::DynFuncHolder< decltype(sound4cl_GetBufferIn ) > sound4::CProcessor::fnGetBufferIn
[protected]
```

Definition at line 302 of file sound4cl.hpp.

10.17.5.18 fnGetBufferOut

```
helper::DynFuncHolder< decltype(sound4cl_GetBufferOut ) > sound4::CProcessor::fnGetBufferOut
[protected]
```

Definition at line 303 of file sound4cl.hpp.

10.17.5.19 fnGetBytesFromFormat

```
helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat ) > sound4::CProcessor::fnGet↩
BytesFromFormat [protected]
```

Definition at line 307 of file sound4cl.hpp.

10.17.5.20 fnGetChannelCount

```
helper::DynFuncHolder< decltype(sound4cl_GetChannelCount ) > sound4::CProcessor::fnGetChannel↩
Count [protected]
```

Definition at line 282 of file sound4cl.hpp.

10.17.5.21 fnGetChunkSizeInFrames

```
helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames ) > sound4::CProcessor::fnGet↩
ChunkSizeInFrames [protected]
```

Definition at line 281 of file sound4cl.hpp.

10.17.5.22 fnGetEstimatedDelay

```
helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay ) > sound4::CProcessor::fnGet↩
EstimatedDelay [protected]
```

Definition at line 304 of file sound4cl.hpp.

10.17.5.23 fnGetFormatFromName

```
helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName ) > sound4::CProcessor::fnGet↩
FormatFromName [protected]
```

Definition at line 306 of file sound4cl.hpp.

10.17.5.24 fnGetFormatName

```
helper::DynFuncHolder< decltype(sound4cl_GetFormatName ) > sound4::CProcessor::fnGetFormatName
[protected]
```

Definition at line 305 of file sound4cl.hpp.

10.17.5.25 fnGetMaxPacketFrame

```
helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame ) > sound4::CProcessor::fnGetMax↩
PacketFrame [protected]
```

Definition at line 308 of file sound4cl.hpp.

10.17.5.26 fnGetMetadataInfos

```
helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos ) > sound4::CProcessor::fnGet↩
MetadataInfos [protected]
```

Definition at line 338 of file sound4cl.hpp.

10.17.5.27 fnGetOutputCount

```
helper::DynFuncHolder< decltype(sound4cl_GetOutputCount ) > sound4::CProcessor::fnGetOutput↩
Count [protected]
```

Definition at line 311 of file sound4cl.hpp.

10.17.5.28 fnGetParameter

```
helper::DynFuncHolder< decltype(sound4cl_GetParameter ) > sound4::CProcessor::fnGetParameter
[protected]
```

Definition at line 291 of file sound4cl.hpp.

10.17.5.29 fnGetPossibleChunkSizeInFrames

```
helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames ) > sound4::CProcessor↩
::fnGetPossibleChunkSizeInFrames [protected]
```

Definition at line 346 of file sound4cl.hpp.

10.17.5.30 fnGetProcessChunkFrames

```
helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames ) > sound4::CProcessor::fnGet↩
ProcessChunkFrames [protected]
```

Definition at line 347 of file sound4cl.hpp.

10.17.5.31 fnGetSampleRate

```
helper::DynFuncHolder< decltype(sound4cl_GetSampleRate ) > sound4::CProcessor::fnGetSampleRate
[protected]
```

Definition at line 285 of file sound4cl.hpp.

10.17.5.32 fnGetVersion

```
helper::DynFuncHolder< decltype(sound4cl_GetVersion ) > sound4::CProcessor::fnGetVersion
[protected]
```

Definition at line 280 of file sound4cl.hpp.

10.17.5.33 fnInitProcess

```
helper::DynFuncHolder< decltype(sound4cl_InitProcess ) > sound4::CProcessor::fnInitProcess
[protected]
```

Definition at line 293 of file sound4cl.hpp.

10.17.5.34 fnInitProcess2

```
helper::DynFuncHolder< decltype(sound4cl_InitProcess2 ) > sound4::CProcessor::fnInitProcess2
[protected]
```

Definition at line 294 of file sound4cl.hpp.

10.17.5.35 fnInitProcess3

```
helper::DynFuncHolder< decltype(sound4cl_InitProcess3 ) > sound4::CProcessor::fnInitProcess3
[protected]
```

Definition at line 348 of file sound4cl.hpp.

10.17.5.36 fnMonoToStereo

```
helper::DynFuncHolder< decltype(sound4cl_MonoToStereo ) > sound4::CProcessor::fnMonoToStereo
[protected]
```

Definition at line 316 of file sound4cl.hpp.

10.17.5.37 fnMonoToStereo_Planar

```
helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar ) > sound4::CProcessor::fnMono↩
ToStereo_Planar [protected]
```

Definition at line 334 of file sound4cl.hpp.

10.17.5.38 fnNewBus

```
helper::DynFuncHolder< decltype(sound4cl_NewBus ) > sound4::CProcessor::fnNewBus [protected]
```

Definition at line 351 of file sound4cl.hpp.

10.17.5.39 fnNewClient

```
helper::DynFuncHolder< decltype(sound4cl_NewClient ) > sound4::CProcessor::fnNewClient [protected]
```

Definition at line 319 of file sound4cl.hpp.

10.17.5.40 fnNewParameters

```
helper::DynFuncHolder< decltype(sound4cl_NewParameters ) > sound4::CProcessor::fnNewParameters
[protected]
```

Definition at line 288 of file sound4cl.hpp.

10.17.5.41 fnPresetManager_InformChange

```
helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange ) > sound4::CProcessor↩
::fnPresetManager_InformChange [protected]
```

Definition at line 343 of file sound4cl.hpp.

10.17.5.42 fnProcessAudio

```
helper::DynFuncHolder< decltype(sound4cl_ProcessAudio ) > sound4::CProcessor::fnProcessAudio
[protected]
```

Definition at line 300 of file sound4cl.hpp.

10.17.5.43 fnProcessAudio_Planar

```
helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar ) > sound4::CProcessor::fn↩
ProcessAudio_Planar [protected]
```

Definition at line 301 of file sound4cl.hpp.

10.17.5.44 fnProcessJson

```
helper::DynFuncHolder< decltype(sound4cl_ProcessJson ) > sound4::CProcessor::fnProcessJson
[protected]
```

Definition at line 321 of file sound4cl.hpp.

10.17.5.45 fnSaveState

```
helper::DynFuncHolder< decltype(sound4cl_SaveState ) > sound4::CProcessor::fnSaveState [protected]
```

Definition at line 323 of file sound4cl.hpp.

10.17.5.46 fnSetInstanceBus

```
helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus ) > sound4::CProcessor::fnSetInstance↩
Bus [protected]
```

Definition at line 353 of file sound4cl.hpp.

10.17.5.47 fnSetLoggerCallback

```
helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback ) > sound4::CProcessor::fnSet↩
LoggerCallback [protected]
```

Definition at line 286 of file sound4cl.hpp.

10.17.5.48 fnSetLogSeverity

```
helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity ) > sound4::CProcessor::fnSetLog↩
Severity [protected]
```

Definition at line 287 of file sound4cl.hpp.

10.17.5.49 fnSetMetadata

```
helper::DynFuncHolder< decltype(sound4cl_SetMetadata ) > sound4::CProcessor::fnSetMetadata
[protected]
```

Definition at line 337 of file sound4cl.hpp.

10.17.5.50 fnSetMetadataMulti

```
helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti ) > sound4::CProcessor::fnSet↩
MetadataMulti [protected]
```

Definition at line 356 of file sound4cl.hpp.

10.17.5.51 fnSetParameter

```
helper::DynFuncHolder< decltype(sound4cl_SetParameter ) > sound4::CProcessor::fnSetParameter
[protected]
```

Definition at line 290 of file sound4cl.hpp.

10.17.5.52 fnSetPresetManager

```
helper::DynFuncHolder< decltype(sound4cl_SetPresetManager ) > sound4::CProcessor::fnSet↩
PresetManager [protected]
```

Definition at line 342 of file sound4cl.hpp.

10.17.5.53 fnStartUpdateThread

```
helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread ) > sound4::CProcessor::fnStart↩
UpdateThread [protected]
```

Definition at line 297 of file sound4cl.hpp.

10.17.5.54 fnStereoToMono

```
helper::DynFuncHolder< decltype(sound4cl_StereoToMono ) > sound4::CProcessor::fnStereoToMono
[protected]
```

Definition at line 315 of file sound4cl.hpp.

10.17.5.55 fnStereoToMono_Planar

```
helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar ) > sound4::CProcessor::fn↩
StereoToMono_Planar [protected]
```

Definition at line 333 of file sound4cl.hpp.

10.17.5.56 fnStopUpdateThread

```
helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread ) > sound4::CProcessor::fnStop↩
UpdateThread [protected]
```

Definition at line 298 of file sound4cl.hpp.

10.17.5.57 fnTerminateProcess

```
helper::DynFuncHolder< decltype(sound4cl_TerminateProcess ) > sound4::CProcessor::fnTerminate↩
Process [protected]
```

Definition at line 295 of file sound4cl.hpp.

10.17.5.58 fnWaitUpdateThreadReady

```
helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady ) > sound4::CProcessor::fn↩
WaitUpdateThreadReady [protected]
```

Definition at line 299 of file sound4cl.hpp.

10.17.5.59 fnWebserver

```
helper::DynFuncHolder< decltype(sound4cl_Webserver ) > sound4::CProcessor::fnWebserver [protected]
```

Definition at line 328 of file sound4cl.hpp.

10.17.5.60 fnWebserver_FreeString

```
helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString ) > sound4::CProcessor::fn↩
Webserver_FreeString [protected]
```

Definition at line 361 of file sound4cl.hpp.

10.17.5.61 fnWebserver_GetAppHealth

```
helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth ) > sound4::CProcessor::fn↩
Webserver_GetAppHealth [protected]
```

Definition at line 360 of file sound4cl.hpp.

10.17.5.62 fnWebserver_SetAppHealth

```
helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth ) > sound4::CProcessor::fn↩
Webserver_SetAppHealth [protected]
```

Definition at line 359 of file sound4cl.hpp.

10.17.5.63 fnWebserver_Status

`helper::DynFuncHolder< decltype(sound4cl_Webserver_Status ) > sound4::CProcessor::fnWebserver↩_Status [protected]`

Definition at line 330 of file sound4cl.hpp.

10.17.5.64 fnWebserver_Stop

`helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop ) > sound4::CProcessor::fnWebserver_↩Stop [protected]`

Definition at line 329 of file sound4cl.hpp.

10.17.5.65 fnWebserver_tcp

`helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp ) > sound4::CProcessor::fnWebserver_tcp [protected]`

Definition at line 326 of file sound4cl.hpp.

10.17.5.66 fnWebserver_tcp2

`helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2 ) > sound4::CProcessor::fnWebserver_↩tcp2 [protected]`

Definition at line 327 of file sound4cl.hpp.

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

- sound4cl.hpp

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

Helper to wrap a function pointer.

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

Public Member Functions

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

10.18.1 Detailed Description

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

Parameters

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

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

10.18.2 Constructor & Destructor Documentation

10.18.2.1 DynFuncHolder() [1/2]

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

10.18.2.2 DynFuncHolder() [2/2]

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

Definition at line 91 of file sound4.x1.cloud_dyn.hpp.

10.18.3 Member Function Documentation

10.18.3.1 IsOk()

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

Definition at line 93 of file sound4.x1.cloud_dyn.hpp.

10.18.3.2 operator bool()

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

Definition at line 92 of file sound4.x1.cloud_dyn.hpp.

10.18.3.3 operator T∗()

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

Definition at line 94 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.19 sound4::helper::DynFuncHolder< T > Class Template Reference

Helper to wrap a function pointer.

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

Public Member Functions

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

10.19.1 Detailed Description

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

Parameters

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

Definition at line 115 of file sound4cl.hpp.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 DynFuncHolder() [1/2]

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

10.19.2.2 DynFuncHolder() [2/2]

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

Definition at line 120 of file sound4cl.hpp.

10.19.3 Member Function Documentation

10.19.3.1 IsOk()

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

Definition at line 122 of file sound4cl.hpp.

10.19.3.2 operator bool()

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

Definition at line 121 of file sound4cl.hpp.

10.19.3.3 operator T∗()

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

Definition at line 123 of file sound4cl.hpp.

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

- sound4cl.hpp

10.20 MyPresetManager Class Reference

[Custom preset manager]

Inherits sound4::cloudx1::dyn::CPresetLoader, and sound4::CPresetLoader.

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

10.20.1 Detailed Description

Definition at line 119 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 MyPresetManager() [1/2]

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

10.20.2.2 ~MyPresetManager() [1/2]

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

10.20.2.3 MyPresetManager() [2/2]

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

10.20.2.4 ~MyPresetManager() [2/2]

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

10.20.3 Member Function Documentation

10.20.3.1 Exists() [1/2]

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

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

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

10.20.3.2 Exists() [2/2]

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

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

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

10.20.3.3 GetAll() [1/2]

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

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

Definition at line 153 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.4 GetAll() [2/2]

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

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

Definition at line 137 of file sound4cl-pipe.cpp.

10.20.3.5 IsReadOnly() [1/2]

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

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

Definition at line 136 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.6 IsReadOnly() [2/2]

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

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

Definition at line 120 of file sound4cl-pipe.cpp.

10.20.3.7 Read() [1/2]

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

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

Definition at line 163 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.8 Read() [2/2]

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

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

Definition at line 147 of file sound4cl-pipe.cpp.

10.20.3.9 Remove() [1/2]

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

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

Definition at line 142 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.10 Remove() [2/2]

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

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

Definition at line 126 of file sound4cl-pipe.cpp.

10.20.3.11 Rename() [1/2]

```
virtual bool MyPresetManager::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [inline], [override], [virtual]
```

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

Definition at line 147 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.12 Rename() [2/2]

```
virtual bool MyPresetManager::Rename (
            const std::filesystem::path & from,
            const std::filesystem::path & to )  [inline], [override], [virtual]
```

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

Definition at line 131 of file sound4cl-pipe.cpp.

10.20.3.13 SetupPath() [1/2]

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

Definition at line 124 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.14 SetupPath() [2/2]

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

Definition at line 108 of file sound4cl-pipe.cpp.

10.20.3.15 Write() [1/2]

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

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

Definition at line 173 of file sound4.x1.cloud-pipe-dyn.cpp.

10.20.3.16 Write() [2/2]

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

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

Definition at line 157 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

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

Helper for Sample format types.

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

10.21.1 Detailed Description

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

Template Parameters

T	the C type for one audio sample

Definition at line 523 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

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

Helper for Sample format types.

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

10.22.1 Detailed Description

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

Template Parameters

T	the C type for one audio sample

Definition at line 116 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

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

Helper for Sample format types.

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

10.23.1 Detailed Description

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

Template Parameters

T	the C type for one audio sample

Definition at line 85 of file sound4cl.hpp.

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

- sound4cl.hpp

10.24 sound4::cloudx1::dyn::helper::SampleFormat< float > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = F32_NATIVE

10.24.1 Detailed Description

Definition at line 534 of file sound4.x1.cloud_dyn.hpp.

10.24.2 Member Data Documentation

10.24.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::dyn::helper::SampleFormat< float >::format =
F32_NATIVE
```

Definition at line 535 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.25 sound4::cloudx1::helper::SampleFormat< float > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = F32_NATIVE

10.25.1 Detailed Description

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

10.25.2 Member Data Documentation

10.25.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::helper::SampleFormat< float >::format = F32_NATIVE
```

Definition at line 128 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

10.26 sound4::helper::SampleFormat< float > Struct Reference

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

Public Attributes

- const sound4cl_SampleFormat format = F32_NATIVE

10.26.1 Detailed Description

Definition at line 96 of file sound4cl.hpp.

10.26.2 Member Data Documentation

10.26.2.1 format

```
const sound4cl_SampleFormat sound4::helper::SampleFormat< float >::format = F32_NATIVE
```

Definition at line 97 of file sound4cl.hpp.

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

- sound4cl.hpp

10.27 sound4::cloudx1::dyn::helper::SampleFormat< int16_t > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = S16_NATIVE

10.27.1 Detailed Description

Definition at line 526 of file sound4.x1.cloud_dyn.hpp.

10.27.2 Member Data Documentation

10.27.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::dyn::helper::SampleFormat< int16_t >::format =
S16_NATIVE
```

Definition at line 527 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.28 sound4::cloudx1::helper::SampleFormat< int16_t > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = S16_NATIVE

10.28.1 Detailed Description

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

10.28.2 Member Data Documentation

10.28.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::helper::SampleFormat< int16_t >::format = S16_NATIVE
```

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

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

- sound4.x1.cloud.hpp

10.29 sound4::helper::SampleFormat< int16_t > Struct Reference

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

Public Attributes

- const sound4cl_SampleFormat format = S16_NATIVE

10.29.1 Detailed Description

Definition at line 88 of file sound4cl.hpp.

10.29.2 Member Data Documentation

10.29.2.1 format

```
const sound4cl_SampleFormat sound4::helper::SampleFormat< int16_t >::format = S16_NATIVE
```

Definition at line 89 of file sound4cl.hpp.

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

- sound4cl.hpp

10.30 sound4::cloudx1::dyn::helper::SampleFormat< int32_t > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = S32_NATIVE

10.30.1 Detailed Description

Definition at line 530 of file sound4.x1.cloud_dyn.hpp.

10.30.2 Member Data Documentation

10.30.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::dyn::helper::SampleFormat< int32_t >::format =
S32_NATIVE
```

Definition at line 531 of file sound4.x1.cloud_dyn.hpp.

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

- sound4.x1.cloud_dyn.hpp

10.31 sound4::cloudx1::helper::SampleFormat< int32_t > Struct Reference

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

Public Attributes

- const cloudx1_SampleFormat format = S32_NATIVE

10.31.1 Detailed Description

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

10.31.2 Member Data Documentation

10.31.2.1 format

```
const cloudx1_SampleFormat sound4::cloudx1::helper::SampleFormat< int32_t >::format = S32_NATIVE
```

Definition at line 124 of file sound4.x1.cloud.hpp.

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

- sound4.x1.cloud.hpp

10.32 sound4::helper::SampleFormat< int32_t > Struct Reference

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

Public Attributes

- const sound4cl_SampleFormat format = S32_NATIVE

10.32.1 Detailed Description

Definition at line 92 of file sound4cl.hpp.

10.32.2 Member Data Documentation

10.32.2.1 format

```
const sound4cl_SampleFormat sound4::helper::SampleFormat< int32_t >::format = S32_NATIVE
```

Definition at line 93 of file sound4cl.hpp.

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

- sound4cl.hpp

10.33 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.

10.33.1 Detailed Description

Information on this processor.

See also

SOUND4_GetProcessInfo

Definition at line 188 of file sound4.x1.cloud.h.

10.33.2 Member Data Documentation

10.33.2.1 name

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

Definition at line 190 of file sound4.x1.cloud.h.

10.33.2.2 prefix

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

Definition at line 191 of file sound4.x1.cloud.h.

10.33.2.3 shortname

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

Definition at line 192 of file sound4.x1.cloud.h.

10.33.2.4 struct_size

```
size_t SOUND4_ProcessInfo::struct_size
```

Definition at line 189 of file sound4.x1.cloud.h.

10.33.2.5 version

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

Definition at line 193 of file sound4.x1.cloud.h.

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

- sound4.x1.cloud.h

10.34 sound4cl_SOUND4_ProcessInfo Struct Reference

```
#include <sound4cl_cdef.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.

10.34.1 Detailed Description

Definition at line 34 of file sound4cl_cdef.h.

10.34.2 Member Data Documentation

10.34.2.1 name

```
const char* sound4cl_SOUND4_ProcessInfo::name
```

Definition at line 36 of file sound4cl_cdef.h.

10.34.2.2 prefix

```
const char* sound4cl_SOUND4_ProcessInfo::prefix
```

Definition at line 37 of file sound4cl_cdef.h.

10.34.2.3 shortname

```
const char* sound4cl_SOUND4_ProcessInfo::shortname
```

Definition at line 38 of file sound4cl_cdef.h.

10.34.2.4 struct_size

```
size_t sound4cl_SOUND4_ProcessInfo::struct_size
```

Definition at line 35 of file sound4cl_cdef.h.

10.34.2.5 version

```
const char* sound4cl_SOUND4_ProcessInfo::version
```

Definition at line 39 of file sound4cl_cdef.h.

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

- sound4cl_cdef.h

Chapter 11

File Documentation

11.1 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)

11.1.1 Function Documentation

11.1.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 39 of file sound4.x1.cloud-pipe-cxx.cpp.

11.1.1.2 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 104 of file sound4.x1.cloud-pipe-cxx.cpp.

11.1.1.3 MyLogger()

```
void MyLogger (
            sound4::cloudx1::LogSeverity severity,
            const std::string & msg )
```

Definition at line 99 of file sound4.x1.cloud-pipe-cxx.cpp.

11.1.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::cloudx1::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 48 of file sound4.x1.cloud-pipe-cxx.cpp.

11.1.1.5 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 79 of file sound4.x1.cloud-pipe-cxx.cpp.

11.2 sound4.x1.cloud-pipe-cxx.cpp

```
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 // ------------------------------------------------------------
00039 std::string GetEnv(const std::string& name)
00040 {
00041     char *data=getenv(name.c_str());
00042     if (!data) return {};
00043     return std::string(data);
00044 }
00045
00046
00047 // Reads from stdin, write to stdout
00048 void ProcessLoop(sound4::cloudx1::CInstance &instance)
00049 {
00050 bool stopthread=false;
00051
00052     #if defined(_WIN32) || defined(WIN32)
00053         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00054         _setmode(_fileno(stdin), _O_BINARY);
00055         _setmode(_fileno(stdout), _O_BINARY);
00056     #endif
00057
00059     while (!stopthread)
00060     {
00061         // Read a chunk from stdin
00062         if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::cloudx1::InputSampleSize,
       stdin) != sound4::cloudx1::InputSampleSize) {
00063             std::cerr « "ProcessLoop: input stopped"« std::endl;
00064             stopthread=true;
00065             break;
00066         }
00067         if (stopthread) break;
00068         // Process it
00069         instance.ProcessAudio();
00070         // Write it to stdout
00071         if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::cloudx1::OutputSampleSize,
       stdout) != sound4::cloudx1::OutputSampleSize) {
00072             std::cerr « "Failed to write audio output" « std::endl;
00073             stopthread=true;
00074         }
00075     }
00077 }
00078
00079 void Usage(const char *progname)
00080 {
00081     std::cerr « "Usage: " « progname « std::endl;;
00082     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00083     std::cerr « "Environment:"« std::endl;;
00084     std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
       std::endl;;
00085     std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00086     std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00087     std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00088     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00089     std::cerr « std::endl;
00090     std::cerr « "Example:"« std::endl;
00091     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;;
00092     #if defined(_WIN32) || defined(WIN32)
00093         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;;
00094     #else
00095         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;;
00096     #endif
00097 }
00098
00099 void MyLogger(sound4::cloudx1::LogSeverity severity,const std::string& msg)
00100 {
00101     std::cerr « msg « std::endl;
00102 }
```

```
int main(int argc, const char **argv)
{
#if (CLOUDX1_HAS_WEBSERVER)
    int listen_port = 8080;
#endif // (CLOUDX1_HAS_WEBSERVER)
    auto store_path_str=GetEnv("S4STORE_PATH");

    // Get credentials from environment
    auto LoginKey=GetEnv("S4LOGINKEY");
    if (LoginKey.empty()) {
        LoginKey=GetEnv("S4USERID");
    }
    auto RadioName=GetEnv("RADIO_NAME");
    auto KeyId=GetEnv("ACCESS_KEY_ID");
    auto AccessKey=GetEnv("ACCESS_KEY_SECRET");

    const char *progname=argv[0];

    if (argc>1) {
        if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
            Usage(progname);
            return 0;
        }
    }
    if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
        std::cerr « "Missing credentials in environment" « std::endl;
        Usage(progname);
        return 1;
    }
    std::filesystem::path store_path;
    if (!store_path_str.empty()) {
        store_path = std::filesystem::path(store_path_str);
        if (!std::filesystem::exists(store_path) || !std::filesystem::is_directory(store_path)) {
            std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
std::endl;
            Usage(progname);
            return 1;
        }
    }

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

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

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

#if (CLOUDX1_HAS_WEBSERVER)
    std::cerr « "Starting web server" « std::endl;
    if (!instance.StartWebServer(listen_port,0)) {
        std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
        return 1;
    }
#endif // (CLOUDX1_HAS_WEBSERVER)

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

    ProcessLoop(instance);

    std::cerr « "Stopping now" « std::endl;
    instance.Stop();

    std::cerr « "End" « std::endl;
    return 0;
}
```

11.3 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]

11.3.1 Function Documentation

11.3.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 51 of file sound4.x1.cloud-pipe-dyn.cpp.

11.3.1.2 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 194 of file sound4.x1.cloud-pipe-dyn.cpp.

11.3.1.3 MyLogger()

```
void MyLogger (
            sound4::cloudx1::dyn::LogSeverity severity,
            const std::string & msg )
```

Definition at line 113 of file sound4.x1.cloud-pipe-dyn.cpp.

11.3.1.4 ProcessLoop()

```
void ProcessLoop (
            sound4::cloudx1::dyn::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 60 of file sound4.x1.cloud-pipe-dyn.cpp.

11.3.1.5 Usage()

```
void Usage (
            const char * progname,
            sound4::cloudx1::dyn::CDynLib & dynlib )
```

Definition at line 91 of file sound4.x1.cloud-pipe-dyn.cpp.

11.4 sound4.x1.cloud-pipe-dyn.cpp

```
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 // ------------------------------------------------------------
00051 std::string GetEnv(const std::string& name)
00052 {
00053     char *data=getenv(name.c_str());
00054     if (!data) return {};
00055     return std::string(data);
00056 }
00057
00058
00059 // Reads from stdin, write to stdout
00060 void ProcessLoop(sound4::cloudx1::dyn::CInstance &instance)
00061 {
00062 bool stopthread=false;
00063
00064     #if defined(_WIN32) || defined(WIN32)
00065         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00066         _setmode(_fileno(stdin), _O_BINARY);
00067         _setmode(_fileno(stdout), _O_BINARY);
00068     #endif
00069
00071     while (!stopthread)
00072     {
00073         // Read a chunk from stdin
00074         if (std::fread(instance.GetBufferIn().data(), sizeof(float),
       sound4::cloudx1::dyn::InputSampleSize, stdin) != sound4::cloudx1::dyn::InputSampleSize) {
00075             std::cerr « "ProcessLoop: input stopped"« std::endl;
00076             stopthread=true;
00077             break;
00078         }
00079         if (stopthread) break;
00080         // Process it
00081         instance.ProcessAudio();
00082         // Write it to stdout
00083         if (fwrite(instance.GetBufferOut().data(), sizeof(float),
       sound4::cloudx1::dyn::OutputSampleSize, stdout) != sound4::cloudx1::dyn::OutputSampleSize) {
00084             std::cerr « "Failed to write audio output" « std::endl;
00085             stopthread=true;
00086         }
00087     }
00089 }
00090
00091 void Usage(const char *progname, sound4::cloudx1::dyn::CDynLib &dynlib)
00092 {
00093     std::cerr « "Usage: " « progname « std::endl;;
00094     std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
00095     std::cerr « "Environment:"« std::endl;;
00096     std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"«
       std::endl;;
00097     std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
00098     std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
00099     std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
00100     std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
00101     if (dynlib.IsOk()) {
00102         std::cerr « std::endl;
```

```
        std::cerr « "Example:"« std::endl;
        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;;
        #if defined(_WIN32) || defined(WIN32)
            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;;
        #else
            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;;
        #endif
    }
}

void MyLogger(sound4::cloudx1::dyn::LogSeverity severity,const std::string& msg)
{
    std::cerr « msg « std::endl;
}

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;
            if ( !std::filesystem::exists(m_preset_path ) ) {
                return false;
            }
            std::error_code ec;
            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;
            for (const auto & entry : std::filesystem::directory_iterator(m_preset_path.c_str(), std::filesystem::directory_options::skip_permission_denied)) {
                if (entry.is_regular_file()) {
                    list.push_back(entry.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;
        }
};

int main(int argc, const char **argv)
{
#if (CLOUDX1_HAS_WEBSERVER)
    int listen_port = 8080;
#endif // (CLOUDX1_HAS_WEBSERVER)
    auto store_path_str=GetEnv("S4STORE_PATH");

    // Get credentials from environment
    auto LoginKey=GetEnv("S4LOGINKEY");
    if (LoginKey.empty()) {
        LoginKey=GetEnv("S4USERID");
    }
    auto RadioName=GetEnv("RADIO_NAME");
    auto KeyId=GetEnv("ACCESS_KEY_ID");
    auto AccessKey=GetEnv("ACCESS_KEY_SECRET");

    const char *progname=argv[0];


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

    if (argc>1) {
        if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
            Usage(progname,dynlib);
            return 0;
        }
    }
    if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
        std::cerr « "Missing credentials in environment" « std::endl;
        Usage(progname,dynlib);
        return 1;
    }
    std::filesystem::path store_path;
    if (!store_path_str.empty()) {
        store_path = std::filesystem::path(store_path_str);
        if (!std::filesystem::exists(store_path) || !std::filesystem::is_directory(store_path)) {
            std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" «
std::endl;
            Usage(progname,dynlib);
            return 1;
        }
    }

    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");

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

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

#if (CLOUDX1_HAS_WEBSERVER)
    std::cerr « "Starting web server" « std::endl;
    if (!instance.StartWebServer(listen_port,0)) {
        std::cerr « "Failed to create the web server on port " « listen_port « std::endl;
        return 1;
    }
#endif // (CLOUDX1_HAS_WEBSERVER)

```

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

    ProcessLoop(instance);

    std::cerr « "Stopping now" « std::endl;
    instance.Stop();

    std::cerr « "End" « std::endl;
    return 0;
}
```

11.5 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 <experimental/filesystem>
#include <io.h>
#include <fcntl.h>
#include <stringapiset.h>
#include <sound4.x1.cloud.h>
```

Namespaces

- std

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[[maybe_unused]], const char ∗str)
- int main (int argc, char ∗const argv[ ])

11.5.1 Macro Definition Documentation

11.5.1.1 CHANNEL_COUNT

```
#define CHANNEL_COUNT 2
```

Definition at line 28 of file sound4.x1.cloud-pipe.cpp.

11.5.1.2 CHUNK_SIZE

```
#define CHUNK_SIZE 64
```

Definition at line 27 of file sound4.x1.cloud-pipe.cpp.

11.5.1.3 SAMPLE_RATE

```
#define SAMPLE_RATE 48000
```

Definition at line 26 of file sound4.x1.cloud-pipe.cpp.

11.5.2 Function Documentation

11.5.2.1 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 90 of file sound4.x1.cloud-pipe.cpp.

11.5.2.2 MyLogger()

```
void MyLogger (
            cloudx1_LogSeverity severe] [[maybe_unused],
            const char * str )
```

Definition at line 84 of file sound4.x1.cloud-pipe.cpp.

11.5.2.3 ProcessLoop()

```
void ProcessLoop (
            struct cloudx1_CInstance * pInstance )
```

[Processing Loop]

[Processing Loop]

Definition at line 31 of file sound4.x1.cloud-pipe.cpp.

11.5.2.4 Usage()

```
void Usage (
            const char * progname )
```

Definition at line 64 of file sound4.x1.cloud-pipe.cpp.

11.6 sound4.x1.cloud-pipe.cpp

```
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 // Choose good filesystem version depending on compiler.
00010 #if __has_include(<filesystem>)
00011     #include <filesystem>
00012 #else //if __cpp_lib_experimental_filesystem
00013     // We need the alias from std::experimental::filesystem to std::filesystem
00014     #include <experimental/filesystem>
00015     namespace std {
00016         namespace filesystem = experimental::filesystem;
00017     }
00018 #endif
00019 #if defined(_WIN32) || defined(WIN32)
00020     #include <io.h>
00021     #include <fcntl.h>
00022     #include <stringapiset.h>   // UTF-8 conversion
00023 #endif
00024 #include <sound4.x1.cloud.h>
00025
00026 #define SAMPLE_RATE       48000
00027 #define CHUNK_SIZE        64
00028 #define CHANNEL_COUNT     2
00029
00030 // Reads from stdin, write to stdout
00031 void ProcessLoop(struct cloudx1_CInstance *pInstance)
00032 {
00034 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;
        }
    }
}

void Usage(const char *progname)
{
    std::cerr « "Usage: " « progname « std::endl;;
    std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
    std::cerr « "Environment:"« std::endl;;
    std::cerr « " - S4STORE_PATH            : path to store config [default: ./state]"« std::endl;;
    std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
    std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
    std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
    std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;;
    std::cerr « std::endl;
    std::cerr « "Example:"« std::endl;
    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;;
    #if defined(_WIN32) || defined(WIN32)
        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;;
    #else
        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;;
    #endif
}

void MyLogger(cloudx1_LogSeverity severe [[maybe_unused]], const char *str)
{
    std::cerr « str « std::endl;
}


int main(int argc, char *const argv[])
{
    struct cloudx1_CInstance *pInstance = NULL;
    const char *store_path_str=getenv("S4STORE_PATH");
    // Get credentials from environment
    char *pLoginKey=getenv("S4LOGINKEY");
    if (!pLoginKey) {
        pLoginKey=getenv("S4USERID");
    }
    char *pRadioName=getenv("RADIO_NAME");
    char *pKeyId=getenv("ACCESS_KEY_ID");
    char *pAccessKey=getenv("ACCESS_KEY_SECRET");

    const char *progname=argv[0];

    if (argc>1) {
        if ((strcasecmp(argv[1],"-h")==0) || (strcasecmp(argv[1],"--help")==0)) {
            Usage(progname);
            return 0;
        }
    }
    if (!pLoginKey || !pRadioName || !pKeyId || !pAccessKey) {
        std::cerr « "Missing credentials in environment" « std::endl;
        Usage(progname);
        return 1;
    }
    std::filesystem::path fs_store_path;
```

```
    if (!store_path_str || store_path_str[0]==0) {
        fs_store_path=std::filesystem::current_path() / "state";
    } else {
        fs_store_path = std::filesystem::path(store_path_str);
    }
    if (!std::filesystem::exists(fs_store_path) || !std::filesystem::is_directory(fs_store_path)) {
        std::cerr « "Storage folder '"«fs_store_path.c_str()«"' is not a valid directory" « std::endl;
        Usage(progname);
        return 1;
    }
    std::string store_path;
    #ifdef _WIN32
    {
        // Convert path to UTF-8
        std::wstring wstr(fs_store_path.wstring());
        if (wstr.empty()) return {};
        int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
        std::string str(size, 0);
        WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
        store_path=str;
    }
    #else // !_WIN32
        store_path=fs_store_path;
    #endif // !_WIN32

    // Optional: Use our own logging, instead of default color output to stderr.
    //cloudx1_SetLoggerCallback(&MyLogger);

    //cloudx1_SetLogSeverity(cloudx1_LogSeverity::info);
    cloudx1_SetLogSeverity(cloudx1_LogSeverity::verbose5);

    std::cerr « "Process version: " « cloudx1_GetVersion() « std::endl;
    // Do some sanity checks
    if (cloudx1_GetSampleRate()!=SAMPLE_RATE) {
      std::cerr « "Bad library sampling rate, expecting '"«SAMPLE_RATE«"' but has "«
cloudx1_GetSampleRate() « std::endl;
      return 1;
    }
    if (cloudx1_GetChunkSizeInFrames()!=CHUNK_SIZE) {
      std::cerr « "Bad library frame size, expecting '"«CHUNK_SIZE«"' but has "«
cloudx1_GetChunkSizeInFrames() « std::endl;
      return 1;
    }
    if (cloudx1_GetChannelCount()!=CHANNEL_COUNT) {
      std::cerr « "Bad library channel count, expecting '"«CHANNEL_COUNT«"' but has "«
cloudx1_GetChannelCount() « std::endl;
      return 1;
    }

    // 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);
     }*/

    std::cerr « "Estimated delay " « ( cloudx1_GetEstimatedDelay(pInstance)*1000.0/SAMPLE_RATE ) « " ms" « std::endl;

    ProcessLoop(pInstance);

#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);

    return 0;
}
```

11.7 sound4.x1.cloud.h File Reference

C interface for library.

```
#include <stdint.h>
```

Classes

- struct SOUND4_ProcessInfo

Macros

- #define CLOUDX1_DECL __declspec(dllimport)
- #define CLOUDX1_HAS_WEBSERVER 1
- #define CLOUDX1_HAS_CLOUDBUS 0
- #define CLOUDX1_AUDIOFRAME_COUNT 64
- #define CLOUDX1_CHANNEL_COUNT 2
- #define CLOUDX1_AUDIO_INPUT_COUNT 1
- #define CLOUDX1_AUDIO_OUTPUT_COUNT 1
- #define CLOUDX1_SAMPLE_RATE 48000
- #define SOUND4_INVALID_WEBSERVER_ID ((uint64_t)-1)
- #define SOUND4_WEBSERVER_HTTP_OK (1<<0)
- #define SOUND4_WEBSERVER_HTTPS_OK (1<<1)

Typedefs

- typedef void(∗ cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char ∗)
- typedef wchar_t fs_char
- typedef char ∗(∗ cloudx1_storage_reader) (const fs_char ∗filename, void ∗userdata)
- typedef void(∗ cloudx1_storage_reader_free) (char ∗content, void ∗userdata)
- typedef int(∗ cloudx1_storage_writer) (const fs_char ∗filename, const char ∗content, void ∗userdata)
- typedef int(∗ cloudx1_storage_exists) (const fs_char ∗filename, void ∗userdata)
- typedef fs_char ∗∗(∗ cloudx1_storage_getall) (void ∗userdata)
- typedef void(∗ cloudx1_storage_getall_free) (fs_char ∗∗all, void ∗userdata)
- typedef int(∗ cloudx1_storage_remove) (const fs_char ∗filename, void ∗userdata)
- typedef int(∗ cloudx1_storage_rename) (const fs_char ∗from, const fs_char ∗to, void ∗userdata)

Enumerations

- enum cloudx1_LogSeverity {
  none = 0 , fatal = 1 , error = 2 , warning = 3 ,
  info = 4 , verbose = 5 , verbose2 = 6 , verbose3 = 7 ,
  verbose4 = 8 , verbose5 = 9 }
- enum cloudx1_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted }
- enum cloudx1_SampleFormat {
  INVALID_FORMAT =0 , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE =S16_LE , S24_NATIVE =S24_LE , S32_NATIVE =S32_LE ,
  F32_NATIVE =F32_LE }

Functions

- const struct SOUND4_ProcessInfo ∗ SOUND4_GetProcessInfo ()
- const char ∗ cloudx1_GetVersion ()
- unsigned int cloudx1_GetChunkSizeInFrames ()
- unsigned int ∗ cloudx1_GetPossibleChunkSizeInFrames ()
- unsigned int cloudx1_GetChannelCount ()
- unsigned int cloudx1_GetAudioInputCount ()
- unsigned int cloudx1_GetAudioOutputCount ()
- unsigned int cloudx1_GetSampleRate ()
- void cloudx1_SetLoggerCallback (cloudx1_loggerfn logger)

- void cloudx1_SetLogSeverity (enum cloudx1_LogSeverity severity)
- struct cloudx1_CParameters * cloudx1_NewParameters ()
- void cloudx1_FreeParameters (struct cloudx1_CParameters *params)
- void cloudx1_SetParameter (struct cloudx1_CParameters *params, const char *name, const char *value)
- const char * cloudx1_GetParameter (struct cloudx1_CParameters *params, const char *name)
- void cloudx1_FreeParameterValue (const char *value)
- struct cloudx1_CInstance * cloudx1_InitProcess (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path)
- struct cloudx1_CInstance * cloudx1_InitProcess2 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters)
- struct cloudx1_CInstance * cloudx1_InitProcess3 (const char *LoginKey, const char *RadioName, const char *Access_Key_ID, const char *Access_Key_Secret, const char *save_path, const struct cloudx1_CParameters *parameters, unsigned int frames_per_chunk)
- unsigned int cloudx1_GetProcessChunkFrames (struct cloudx1_CInstance *instance)
- int cloudx1_TerminateProcess (struct cloudx1_CInstance *instance)
- void cloudx1_ExitProcess (struct cloudx1_CInstance *instance)
- void cloudx1_SetPresetManager (struct cloudx1_CParameters *params, cloudx1_storage_reader reader, cloudx1_storage_reader_free, cloudx1_storage_writer writer, cloudx1_storage_exists exists, cloudx1_storage_getall getall, cloudx1_storage_getall_free getall_free, cloudx1_storage_remove remove, cloudx1_storage_rename rename, int IsReadOnly, void *userdata)
- void cloudx1_PresetManager_InformChange (struct cloudx1_CInstance *instance, const fs_char *relative↩_path, enum cloudx1_PresetChange_Kind change_kind)
- void cloudx1_SetMetadata (struct cloudx1_CInstance *instance, const char *key, const char *value)
- void cloudx1_SetMetadataMulti (struct cloudx1_CInstance *instance, const char **keyvalue)
- const char ** cloudx1_GetMetadataInfos (struct cloudx1_CInstance *instance)
- void cloudx1_FreeMetadataInfos (struct cloudx1_CInstance *instance, const char **infos)
- void cloudx1_StartUpdateThread (struct cloudx1_CInstance *instance, unsigned int port)
- void cloudx1_StopUpdateThread (struct cloudx1_CInstance *instance)
- int cloudx1_WaitUpdateThreadReady (struct cloudx1_CInstance *instance, int milliseconds)
- void cloudx1_ProcessAudio (struct cloudx1_CInstance *instance, const float *input, float *output)
- void cloudx1_ProcessAudio_Planar (struct cloudx1_CInstance *instance, float const *const *input, float *const *output)
- float * cloudx1_GetBufferIn (struct cloudx1_CInstance *instance)
- float * cloudx1_GetBufferOut (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetEstimatedDelay (struct cloudx1_CInstance *instance)
- const char * cloudx1_GetFormatName (const enum cloudx1_SampleFormat fmt)
- enum cloudx1_SampleFormat cloudx1_GetFormatFromName (const char *name)
- unsigned int cloudx1_GetBytesFromFormat (const enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_GetMaxPacketFrame (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_AddAudio (struct cloudx1_CInstance *instance, const uint8_t *payload, unsigned int nFrame, enum cloudx1_SampleFormat fmt)
- unsigned int cloudx1_AddPadAudio (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetOutputCount (struct cloudx1_CInstance *instance)
- unsigned int cloudx1_GetAudio (struct cloudx1_CInstance *instance, uint8_t *payload, unsigned int max_↩nFrame, enum cloudx1_SampleFormat fmt)
- void cloudx1_AudioConvertFrom (const uint8_t *payload, float *output, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_AudioConvertTo (const float *input, uint8_t *payload, size_t nSpl, enum cloudx1_SampleFormat fmt)
- void cloudx1_StereoToMono (const float *input, float *output, size_t nFrame)
- void cloudx1_MonoToStereo (const float *input, float *output, size_t nFrame)
- void cloudx1_StereoToMono_Planar (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void cloudx1_MonoToStereo_Planar (const float *input, float *outputL, float *outputR, size_t nFrame)
- void cloudx1_AudioMonoFromLiveStereo (const uint8_t *payload, float *output)

- void cloudx1_AudioMonoToLiveStereo (const float ∗input, uint8_t ∗payload)
- struct cloudx1_CClientInstance ∗ cloudx1_NewClient (struct cloudx1_CInstance ∗instance)
- void cloudx1_DeleteClient (struct cloudx1_CClientInstance ∗client)
- const char ∗ cloudx1_ProcessJson (struct cloudx1_CClientInstance ∗client, const char ∗json_str, int ∗need↩ _save)
- void cloudx1_FreeJsonAnswer (const char ∗json_str)
- int cloudx1_SaveState (struct cloudx1_CInstance ∗instance)
- uint64_t cloudx1_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char ∗socket_ip, unsigned int socket_port)
- uint64_t cloudx1_Webserver_tcp2 (unsigned int listenport, unsigned int listenport_secure, const char ∗socket_ip, unsigned int socket_port, const struct cloudx1_CParameters ∗parameters)
- uint64_t cloudx1_Webserver (unsigned int listenport, unsigned int listenport_secure, struct cloudx1_CInstance ∗instance)
- int cloudx1_Webserver_Stop (uint64_t id, int timeout_ms)
- int cloudx1_Webserver_Status (uint64_t id)
- void cloudx1_Webserver_SetAppHealth (struct cloudx1_CInstance ∗instance, int httpcode, const char ∗contenttype, const char ∗content)
- void cloudx1_Webserver_GetAppHealth (struct cloudx1_CInstance ∗instance, int ∗httpcode, char ∗∗contenttype, char ∗∗content)
- void cloudx1_Webserver_FreeString (char ∗str)

11.7.1 Macro Definition Documentation

11.7.1.1 CLOUDX1_DECL

```
#define CLOUDX1_DECL __declspec(dllimport)
```

Definition at line 14 of file sound4.x1.cloud.h.

11.7.1.2 CLOUDX1_HAS_CLOUDBUS

```
#define CLOUDX1_HAS_CLOUDBUS 0
```

Definition at line 19 of file sound4.x1.cloud.h.

11.7.1.3 CLOUDX1_HAS_WEBSERVER

```
#define CLOUDX1_HAS_WEBSERVER 1
```

Definition at line 18 of file sound4.x1.cloud.h.

11.8 sound4.x1.cloud.h

```
00001
00004 #pragma once
00005
00006 #include <stdint.h>
00007 #ifdef __unix__
00008     #include <sys/types.h>
00009     #define CLOUDX1_DECL        __attribute__ ((visibility ("default")))
00010 #elif defined(_WIN32) || defined(WIN32)
00011     #ifdef BUILDING_CLOUDX1_DLL
00012         #define CLOUDX1_DECL    __declspec(dllexport)
00013     #else
00014         #define CLOUDX1_DECL    __declspec(dllimport)
00015     #endif
00016 #endif
00017
00018 #define CLOUDX1_HAS_WEBSERVER 1
00019 #define CLOUDX1_HAS_CLOUDBUS 0
00020
00021 #ifdef __cplusplus
00022 extern "C" {
00023 #endif // __cplusplus
00024
00070 // ------------------------------------------------------------
00089 // ------------------------------------------------------------
00143 // ------------------------------------------------------------
00144
00145 // ------------------------------------------------------------
00157 #define CLOUDX1_AUDIOFRAME_COUNT      64
00163 #define CLOUDX1_CHANNEL_COUNT          2
00169 #define CLOUDX1_AUDIO_INPUT_COUNT      1
00175 #define CLOUDX1_AUDIO_OUTPUT_COUNT     1
00181 #define CLOUDX1_SAMPLE_RATE            48000
00182
00188 struct SOUND4_ProcessInfo {
00189     size_t struct_size;
00190     const char *name;
00191     const char *prefix;
00192     const char *shortname;
00193     const char *version;
00194 };
00195
00203 CLOUDX1_DECL const struct SOUND4_ProcessInfo* SOUND4_GetProcessInfo();
00204
00210 CLOUDX1_DECL const char *cloudx1_GetVersion();
00211
00224 CLOUDX1_DECL unsigned int cloudx1_GetChunkSizeInFrames();
00225
00234 CLOUDX1_DECL unsigned int* cloudx1_GetPossibleChunkSizeInFrames();
00235
00243 CLOUDX1_DECL unsigned int cloudx1_GetChannelCount();
00244
00253 CLOUDX1_DECL unsigned int cloudx1_GetAudioInputCount();
00254
00263 CLOUDX1_DECL unsigned int cloudx1_GetAudioOutputCount();
00264
00272 CLOUDX1_DECL unsigned int cloudx1_GetSampleRate();
00273
00277 enum cloudx1_LogSeverity {
00278     none = 0,
00279     fatal = 1,
00280     error = 2,
00281     warning = 3,
00282     info = 4,
00283     verbose = 5,
00284     verbose2 = 6,
00285     verbose3 = 7,
00286     verbose4 = 8,
00287     verbose5 = 9,
00288 };
00289
00296 typedef void (*cloudx1_loggerfn) (enum cloudx1_LogSeverity, const char *);
00297
00307 CLOUDX1_DECL void cloudx1_SetLoggerCallback( cloudx1_loggerfn logger );
00308
00316 CLOUDX1_DECL void cloudx1_SetLogSeverity( enum cloudx1_LogSeverity severity );
00317
00320 // ------------------------------------------------------------
00326 // ------------------------------------------------------------
00330 struct cloudx1_CInstance;
00331
00335 struct cloudx1_CParameters;
00336
00344 CLOUDX1_DECL struct cloudx1_CParameters *cloudx1_NewParameters();
00345
00349 CLOUDX1_DECL void cloudx1_FreeParameters(struct cloudx1_CParameters *params);
```

```
CLOUDX1_DECL void cloudx1_SetParameter(struct cloudx1_CParameters *params, const char *name, const char *value);

CLOUDX1_DECL const char *cloudx1_GetParameter(struct cloudx1_CParameters *params, const char *name);

CLOUDX1_DECL void cloudx1_FreeParameterValue(const char *value);

CLOUDX1_DECL 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
    );

CLOUDX1_DECL 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
    );

CLOUDX1_DECL 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
    );

CLOUDX1_DECL unsigned int cloudx1_GetProcessChunkFrames(struct cloudx1_CInstance *instance);

CLOUDX1_DECL int cloudx1_TerminateProcess(struct cloudx1_CInstance *instance);

CLOUDX1_DECL void cloudx1_ExitProcess(struct cloudx1_CInstance *instance);

#if CLOUDX1_HAS_CLOUDBUS
// ---------------------------------------------------------------
struct cloudx1_CBus;

CLOUDX1_DECL struct cloudx1_CBus* cloudx1_NewBus();

CLOUDX1_DECL void cloudx1_FreeBus(struct cloudx1_CBus *bus);

CLOUDX1_DECL void cloudx1_SetInstanceBus(struct cloudx1_CParameters *params, struct cloudx1_CBus *bus);


#endif // CLOUDX1_HAS_CLOUDBUS

// ---------------------------------------------------------------
// ---------------------------------------------------------------

// OS dependent filesystem path char type
#ifdef _WIN32
    typedef wchar_t fs_char;
#else // !_WIN32
    typedef char fs_char;
#endif // !_WIN32

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);

CLOUDX1_DECL 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
    );

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

CLOUDX1_DECL void cloudx1_PresetManager_InformChange(struct cloudx1_CInstance *instance, const fs_char *relative_path, enum cloudx1_PresetChange_Kind change_kind);

// ---------------------------------------------------------------
// ---------------------------------------------------------------

CLOUDX1_DECL void cloudx1_SetMetadata(struct cloudx1_CInstance *instance, const char* key, const char* value);

CLOUDX1_DECL void cloudx1_SetMetadataMulti(struct cloudx1_CInstance *instance, const char** keyvalue);

CLOUDX1_DECL const char** cloudx1_GetMetadataInfos(struct cloudx1_CInstance *instance);

CLOUDX1_DECL void cloudx1_FreeMetadataInfos(struct cloudx1_CInstance *instance, const char** infos);

// ---------------------------------------------------------------
// ---------------------------------------------------------------

CLOUDX1_DECL void cloudx1_StartUpdateThread(struct cloudx1_CInstance *instance, unsigned int port);

CLOUDX1_DECL void cloudx1_StopUpdateThread(struct cloudx1_CInstance *instance);

CLOUDX1_DECL int cloudx1_WaitUpdateThreadReady(struct cloudx1_CInstance *instance, int milliseconds);

// ---------------------------------------------------------------
// ---------------------------------------------------------------
CLOUDX1_DECL void cloudx1_ProcessAudio(struct cloudx1_CInstance *instance, const float *input, float *output );

CLOUDX1_DECL void cloudx1_ProcessAudio_Planar(struct cloudx1_CInstance *instance
    , float const * const *input
    , float * const *output);

CLOUDX1_DECL float *cloudx1_GetBufferIn(struct cloudx1_CInstance *instance);

CLOUDX1_DECL float *cloudx1_GetBufferOut(struct cloudx1_CInstance *instance);

CLOUDX1_DECL unsigned int cloudx1_GetEstimatedDelay(struct cloudx1_CInstance *instance);

// ---------------------------------------------------------------
// ---------------------------------------------------------------
    #ifdef _MSC_VER
        #define __BYTE_ORDER__  REG_DWORD
        #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
        #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
    #endif

enum cloudx1_SampleFormat {
    INVALID_FORMAT=0,
    S16_LE,
    S16_BE,
    S24_LE,
    S24_BE,
    S32_LE,
    S32_BE,
    F32_LE,
    F32_BE,

    // Native definitions, only links to the physical ones
    #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
        S16_NATIVE=S16_LE,
        S24_NATIVE=S24_LE,
        S32_NATIVE=S32_LE,
        F32_NATIVE=F32_LE,
    #else
        S16_NATIVE=S16_BE,
        S24_NATIVE=S24_BE,
        S32_NATIVE=S32_BE,
        F32_NATIVE=F32_BE,
    #endif
};

CLOUDX1_DECL const char *cloudx1_GetFormatName(const enum cloudx1_SampleFormat fmt);

CLOUDX1_DECL enum cloudx1_SampleFormat cloudx1_GetFormatFromName(const char *name);

```

```
CLOUDX1_DECL unsigned int cloudx1_GetBytesFromFormat(const enum cloudx1_SampleFormat fmt);


CLOUDX1_DECL unsigned int cloudx1_GetMaxPacketFrame(struct cloudx1_CInstance *instance);

CLOUDX1_DECL unsigned int cloudx1_AddAudio(struct cloudx1_CInstance *instance, const uint8_t *payload,
    unsigned int nFrame, enum cloudx1_SampleFormat fmt );

CLOUDX1_DECL unsigned int cloudx1_AddPadAudio(struct cloudx1_CInstance *instance);

CLOUDX1_DECL unsigned int cloudx1_GetOutputCount(struct cloudx1_CInstance *instance);

CLOUDX1_DECL unsigned int cloudx1_GetAudio(struct cloudx1_CInstance *instance, uint8_t *payload,
    unsigned int max_nFrame, enum cloudx1_SampleFormat fmt );

// ------------------------------------------------------------
// ------------------------------------------------------------
CLOUDX1_DECL void cloudx1_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
    cloudx1_SampleFormat fmt);

CLOUDX1_DECL void cloudx1_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
    cloudx1_SampleFormat fmt);

CLOUDX1_DECL void cloudx1_StereoToMono(const float *input, float *output, size_t nFrame);

CLOUDX1_DECL void cloudx1_MonoToStereo(const float *input, float *output, size_t nFrame);

CLOUDX1_DECL void cloudx1_StereoToMono_Planar(const float *inputL, const float *inputR, float *output,
    size_t nFrame);

CLOUDX1_DECL void cloudx1_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
    size_t nFrame);

CLOUDX1_DECL void cloudx1_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);

CLOUDX1_DECL void cloudx1_AudioMonoToLiveStereo(const float *input, uint8_t *payload);

// ------------------------------------------------------------
// ------------------------------------------------------------
struct cloudx1_CClientInstance;

CLOUDX1_DECL struct cloudx1_CClientInstance *cloudx1_NewClient(struct cloudx1_CInstance *instance);

CLOUDX1_DECL void cloudx1_DeleteClient(struct cloudx1_CClientInstance *client);

CLOUDX1_DECL const char *cloudx1_ProcessJson(struct cloudx1_CClientInstance *client, const char
    *json_str, int *need_save);

CLOUDX1_DECL void cloudx1_FreeJsonAnswer(const char *json_str);

CLOUDX1_DECL int cloudx1_SaveState(struct cloudx1_CInstance *instance);

#if CLOUDX1_HAS_WEBSERVER
// ------------------------------------------------------------
// ------------------------------------------------------------
CLOUDX1_DECL uint64_t cloudx1_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
    const char *socket_ip, unsigned int socket_port);

// ------------------------------------------------------------
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);

CLOUDX1_DECL uint64_t cloudx1_Webserver(unsigned int listenport, unsigned int listenport_secure,
    struct cloudx1_CInstance *instance);

CLOUDX1_DECL int cloudx1_Webserver_Stop(uint64_t id, int timeout_ms);

CLOUDX1_DECL int cloudx1_Webserver_Status(uint64_t id);

CLOUDX1_DECL void cloudx1_Webserver_SetAppHealth(struct cloudx1_CInstance *instance, int httpcode,
    const char *contenttype, const char *content);

CLOUDX1_DECL void cloudx1_Webserver_GetAppHealth(struct cloudx1_CInstance *instance, int* httpcode,
    char** contenttype, char** content);

CLOUDX1_DECL void cloudx1_Webserver_FreeString(char *str);

#define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)

#define SOUND4_WEBSERVER_HTTP_OK               (1«0)
#define SOUND4_WEBSERVER_HTTPS_OK              (1«1)

#endif // CLOUDX1_HAS_WEBSERVER

#ifdef __cplusplus
} // extern "C"
#endif // __cplusplus
```

11.9 sound4.x1.cloud.hpp File Reference

C++ wrapper for library.

```
#include <string>
#include <exception>
#include <thread>
#include <functional>
#include <memory>
#include <array>
#include <assert.h>
#include <stdexcept>
#include <filesystem>
#include <string.h>
#include "sound4.x1.cloud.h"
#include <stringapiset.h>
#include <wchar.h>
```

Classes

- struct sound4::cloudx1::helper::SampleFormat< T >

  Helper for Sample format types.
- struct sound4::cloudx1::helper::SampleFormat< int16_t >
- struct sound4::cloudx1::helper::SampleFormat< int32_t >
- struct sound4::cloudx1::helper::SampleFormat< float >
- class sound4::cloudx1::CPresetLoader
- class sound4::cloudx1::CInstance
- class sound4::cloudx1::CInstance::CClient

Namespaces

- sound4
- sound4::cloudx1
- sound4::cloudx1::helper

Macros

- #define fs_strdup wcsdup

  helper for OS dependant file name copy

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

Variables

- const size_t sound4::cloudx1::ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
- const size_t sound4::cloudx1::InputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_INPUT_COUNT
- const size_t sound4::cloudx1::OutputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_OUTPUT_COUNT
- const size_t sound4::cloudx1::InputSampleSize =ChunkFrames∗InputChannels
- const size_t sound4::cloudx1::OutputSampleSize =ChunkFrames∗OutputChannels
- const size_t sound4::cloudx1::SampleRate =CLOUDX1_SAMPLE_RATE

11.9.1 Macro Definition Documentation

11.9.1.1 fs_strdup

```
#define fs_strdup wcsdup
```

Definition at line 25 of file sound4.x1.cloud.hpp.

11.10 sound4.x1.cloud.hpp

```
00001
00004 #pragma once
00005
00006 #include <string>
00007 #include <exception>
00008 #include <thread>
00009 #include <functional>
00010 #include <memory>
00011 #include <array>
00012 #include <assert.h>
00013 #include <stdexcept>
00014 #include <filesystem>
00015 #include <string.h>
00016
00017 #include "sound4.x1.cloud.h"
00018
00019
00020 // Filesystem path char type
00021 #ifdef _WIN32
00022     #include <stringapiset.h>
00023     #include <wchar.h>
00025     #define fs_strdup wcsdup
00026 #else // !_WIN32
00028     #define fs_strdup strdup
00029 #endif // !_WIN32
00030
00031 // bridge C callbacks to CPresetLoader
00032 extern "C" {
00033     static char *cloudx1_custom_reader(const fs_char *filename, void* userdata);
00034     static void cloudx1_custom_reader_free(char *content, void* userdata);
00035     static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata);
00036     static int cloudx1_custom_exists(const fs_char *filename, void* userdata);
00037     static fs_char** cloudx1_custom_getall(void* userdata);
00038     static void cloudx1_custom_getall_free(fs_char** all, void* userdata);
00039     static int cloudx1_custom_remove(const fs_char *filename, void* userdata);
00040     static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00041 };
00042
00046 namespace sound4 {
00050 namespace cloudx1 {
00051     static constexpr const char *process_name = "SOUND4 x1.CLOUD";
00052     static constexpr const char *process_shortname = "x1";
00053     using SampleFormat = cloudx1_SampleFormat;
00057     const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
00061     const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
00065     const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
00069     const size_t InputSampleSize=ChunkFrames*InputChannels;
00073     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
```

```
    const size_t SampleRate=CLOUDX1_SAMPLE_RATE;

    namespace helper {
        #ifdef _WIN32
        static inline std::string WStringToUTF8(const std::wstring& wstr) {
            if (wstr.empty()) return {};
            int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
            std::string str(size, 0);
            WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
            return str;
        }
        static inline std::wstring UTF8ToWString(const std::string& str) {
            if (str.empty()) return std::wstring();
            int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
            std::wstring wstr(size, 0);
            MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
            return wstr;
        }
        #endif // _WIN32

            template<typename T>
        struct SampleFormat {
        };
            template<>
        struct SampleFormat<int16_t> {
            const cloudx1_SampleFormat format = S16_NATIVE;
        };
            template<>
        struct SampleFormat<int32_t> {
            const cloudx1_SampleFormat format = S32_NATIVE;
        };
            template<>
        struct SampleFormat<float> {
            const cloudx1_SampleFormat format = F32_NATIVE;
        };


        static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, cloudx1_SampleFormat fmt) {
            cloudx1_AudioConvertFrom(payload, output, nSpl, fmt);
        }
        static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, cloudx1_SampleFormat fmt) {
            cloudx1_AudioConvertTo(input, payload, nSpl, fmt);
        }
        static inline void StereoToMono(const float *input, float *output, size_t nSpl) {
            cloudx1_StereoToMono(input, output, nSpl);
        }
        static inline void MonoToStereo(const float *input, float *output, size_t nSpl) {
            cloudx1_MonoToStereo(input, output, nSpl);
        }
        static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t nSpl) {
            cloudx1_StereoToMono_Planar(inputL, inputR, output, nSpl);
        }
        static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nSpl) {
            cloudx1_MonoToStereo_Planar(input, outputL, outputR, nSpl);
        }
        static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
            cloudx1_AudioMonoFromLiveStereo(payload, output);
        }
        static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
            cloudx1_AudioMonoToLiveStereo(input, payload);
        }

    }; // namespace helper

    static inline std::string GetVersion()
        {
            return std::string(cloudx1_GetVersion());
        }

    static inline unsigned int GetChunkSizeInFrames()
        {
            return cloudx1_GetChunkSizeInFrames();
        }

    static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
        {
            std::vector<unsigned int> list;
            for (unsigned int* src=cloudx1_GetPossibleChunkSizeInFrames();*src;src++) {
                list.push_back(*src);
            }
            return list;
        }
```

```
    static inline unsigned int GetChannelCount()
        {
            return cloudx1_GetChannelCount();
        }

    static inline unsigned int GetAudioInputCount()
        {
            return cloudx1_GetAudioInputCount();
        }

    static inline unsigned int GetAudioOutputCount()
        {
            return cloudx1_GetAudioOutputCount();
        }

    static inline unsigned int GetSampleRate()
        {
            return cloudx1_GetSampleRate();
        }

    static inline void SanityCheck(bool a_bCheckFrames = true)
    {
        // Do some sanity checks
        if (cloudx1_GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
            throw std::runtime_error("Bad library sampling rate");
        }
        if (a_bCheckFrames && cloudx1_GetChunkSizeInFrames()!=ChunkFrames) {
            throw std::runtime_error("Bad library frame size");
        }
        if (cloudx1_GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
            throw std::runtime_error("Bad library channel count");
        }
        if (cloudx1_GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
            throw std::runtime_error("Bad library input count");
        }
        if (cloudx1_GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
            throw std::runtime_error("Bad library output count");
        }
    }

    static inline std::string GetFormatName(const SampleFormat fmt)
        {
            return std::string(cloudx1_GetFormatName(fmt));
        }

    static inline SampleFormat GetFormatFromName(const std::string& name)
        {
            return cloudx1_GetFormatFromName(name.c_str());
        }

    static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
        {
            return cloudx1_GetBytesFromFormat(fmt);
        }

    using LogSeverity = cloudx1_LogSeverity;
    using log_cb_t = std::function<void(LogSeverity,const std::string&)>;

    static log_cb_t _log_cb;
    extern "C" {
        static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
            _log_cb(severity, std::string(c_msg));
        }
    }
    static inline void SetLogSeverity(LogSeverity severity)
        {
            cloudx1_SetLogSeverity(severity);
        }
    static inline void SetLoggerCallback(log_cb_t cb)
        {
            _log_cb=cb;
            cloudx1_SetLoggerCallback(&_log_cb_c);
        }

#if CLOUDX1_HAS_CLOUDBUS
    class CBus {
        public:
            CBus()
                : m_bus(cloudx1_NewBus()) {}
            ~CBus()
                {
                    cloudx1_FreeBus(m_bus);
                    m_bus = nullptr;
                }
            cloudx1_CBus *Get() const { return m_bus; }
        private:
```

```
            cloudx1_CBus *m_bus;
    };
#endif // CLOUDX1_HAS_CLOUDBUS

    class CPresetLoader {
    public:
        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;
    };
    using PresetChange_Kind = cloudx1_PresetChange_Kind;

    class CInstance {
    public:
        CInstance()
            {
                params = cloudx1_NewParameters();
                if (!params) throw std::bad_alloc();
            }
        // non-copyable
        CInstance( const CInstance& ) = delete;
        CInstance& operator=( const CInstance& ) = delete;
        ~CInstance()
            {
                Stop();
                if (params) cloudx1_FreeParameters(params);
            }
       bool IsOk() const { return instance!=nullptr; }
        void SetParam(const std::string& name, const std::string& value)
            {
                assert(params);
                cloudx1_SetParameter(params, name.c_str(), value.c_str());
            }
        std::string GetParam(const std::string& name)
            {
                assert(params);
                auto c_value=cloudx1_GetParameter(params, name.c_str());
                std::string ret(c_value);
                cloudx1_FreeParameterValue(c_value);
                return ret;
            }

        void SetPresetManager(CPresetLoader *preset_manager)
            {
                if (!preset_manager) return;
                assert(params);
                cloudx1_SetPresetManager(params,
                    cloudx1_custom_reader,
                    cloudx1_custom_reader_free,
                    cloudx1_custom_writer,
                    cloudx1_custom_exists,
                    cloudx1_custom_getall,
                    cloudx1_custom_getall_free,
                    cloudx1_custom_remove,
                    cloudx1_custom_rename,
                    preset_manager->IsReadOnly(),
                    preset_manager
                );
            }

#if CLOUDX1_HAS_CLOUDBUS
            void SetBus(const CBus& bus) {
                assert(params);
                cloudx1_SetInstanceBus(params,bus.Get());
            }
#endif // CLOUDX1_HAS_CLOUDBUS

        bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
    AWS_Access_ID, const std::string& AWS_Access_Secret, const std::filesystem::path& save_path, int
    json_port = 0, unsigned int frames_per_chunk=64)
            {
                assert(!instance);
                std::string l_save_path;
                if (!save_path.empty()) {
                    l_save_path = save_path.u8string();
                }
                instance = cloudx1_InitProcess3(LoginKey.c_str(), RadioName.c_str(),
    AWS_Access_ID.c_str(), AWS_Access_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
                if (!instance) return false;
                if (!init_metadata.empty()) {
```

```
                    for (auto&& [key,value]: init_metadata) {
                        cloudx1_SetMetadata(instance, key.c_str(), value.c_str());
                    }
                    init_metadata.clear();
                }
                update_thread = std::thread([this, json_port](){
                    OnUpdateThreadStart();
                    cloudx1_StartUpdateThread(instance, json_port);
                    OnUpdateThreadStop();
                });
                int timeout=5000;
                #ifdef DEBUG
                    timeout=60*1000; // For debugger pauses
                #endif
                if (cloudx1_WaitUpdateThreadReady(instance, timeout)<0) {
                    return false;
                }
                return true;
            }

        unsigned int GetChunkFrames() {
            if (instance) {
                return cloudx1_GetProcessChunkFrames(instance);
            }
            return 0;
        }

        void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind change_kind)
            {
                cloudx1_PresetManager_InformChange(instance, relative_path.c_str(), (cloudx1_PresetChange_Kind) change_kind);
            }

        void SetMetadata(const std::string& key, const char* value)
            {
                if (instance) {
                    cloudx1_SetMetadata(instance, key.c_str(), value);
                } else {
                    init_metadata.push_back( {key, value} );
                }
            }

        void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
            {
                if (instance) {
                    typedef const char* pchar;
                    pchar* keyvalue=new pchar[2*list.size()+1];
                    size_t n=0;
                    for (auto&& [key,value]: list) {
                        keyvalue[2*n+0]=key.c_str();
                        keyvalue[2*n+1]=value;
                        n++;
                    }
                    keyvalue[2*n+0]=nullptr;

                    cloudx1_SetMetadataMulti(instance, keyvalue);

                    delete[] keyvalue;
                } else {
                    for (auto&& [key,value]: list) {
                        if (value) {
                            init_metadata.push_back( {key, value} );
                        }
                    }
                }
            }

        std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
            {
                std::vector< std::tuple<std::string,std::string> > values;
                const char** c_values = cloudx1_GetMetadataInfos(instance);
                if (c_values) {
                    for (const char** c_value=c_values; *c_value; ) {
                        std::string key(*c_value);
                        c_value++;
                        if (*c_value) {
                            std::string descr(*c_value);
                            c_value++;
                            values.push_back( {key,descr} );
                        }
                    }
                    cloudx1_FreeMetadataInfos(instance, c_values);
                }
                return values;
            }

```

```
        // Can be overloaded, called at start of UpdateThread in the thread
        virtual void OnUpdateThreadStart() {}
        // Can be overloaded to call at end of UpdateThread in the thread
        virtual void OnUpdateThreadStop() {}

#if CLOUDX1_HAS_WEBSERVER
        bool StartWebServer(int http_port, int https_port=0)
            {
                assert(instance);
                assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
                webserver = cloudx1_Webserver(http_port,https_port,instance);
                if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
                    return false;
                }
                int web_status=cloudx1_Webserver_Status(webserver);
                if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
                    return false;
                }
                if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
                    return false;
                }
                return true;
            }
        void StopWebServer()
            {
                if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
                    cloudx1_Webserver_Stop(webserver, 1000);
                    webserver = SOUND4_INVALID_WEBSERVER_ID;
                }
            }

        void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string& content)
            {
                assert(instance);
                cloudx1_Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(), content.c_str());
            }
        void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
            {
                assert(instance);
                char* c_contenttype=nullptr;
                char* c_content=nullptr;
                cloudx1_Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
                contenttype=c_contenttype;
                content=c_content;
                cloudx1_Webserver_FreeString(c_contenttype);
                cloudx1_Webserver_FreeString(c_content);
        }
#endif // CLOUDX1_HAS_WEBSERVER

        int Terminate()
            {
                if (!instance) return 0;
                return cloudx1_TerminateProcess(instance);
            }

        void Stop()
            {
                if (!instance) return;
#if CLOUDX1_HAS_WEBSERVER
                StopWebServer();
#endif // CLOUDX1_HAS_WEBSERVER
                if (update_thread.joinable()) {
                    cloudx1_StopUpdateThread(instance); // this will exit the update thread
                    update_thread.join();
                }
                cloudx1_ExitProcess(instance);
                instance=nullptr;
            }

        unsigned int GetEstimatedDelay()
            {
                assert(instance);
                return cloudx1_GetEstimatedDelay(instance);
            }

        std::array<float,InputSampleSize>& GetBufferIn()
            {
                assert(instance);
                float *buf=cloudx1_GetBufferIn(instance);
                return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);

            }
        std::array<float,OutputSampleSize>& GetBufferOut()
            {
                assert(instance);
```

```
                float *buf= cloudx1_GetBufferOut(instance);
                return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
            }
        void ProcessAudio()
            {
                assert(instance);
                cloudx1_ProcessAudio(instance, cloudx1_GetBufferIn(instance), cloudx1_GetBufferOut(instance));
            }
        // void ProcessAudio_Planar(const std::array< const std::array<const float,ChunkFrames>, InputChannels>& input, std::array< std::array<float,ChunkFrames>, OutputChannels>& output)
        void ProcessAudio_Planar(float const * const *input, float * const *output)
            {
                assert(instance);
                cloudx1_ProcessAudio_Planar(instance, input, output);
            }

            template<typename T>
        std::vector<T> ProcessAnyAudio(const std::vector<T> input)
            {
                assert(instance);
                std::vector<T> output;
                unsigned int out_offset=0;
                unsigned int in_offset=0;
                unsigned int todo = input.size();
                while (todo>0) {
                    unsigned int left = AddAudio(&input[in_offset], todo);
                    unsigned int out_avail = cloudx1_GetOutputCount(instance);
                    output.resize(out_offset + out_avail);
                    GetAudio(&output[out_offset], out_avail);
                    out_offset+=out_avail;
                    in_offset += todo-left;
                    todo=left;
                }
                return output;
            }

        class CClient {
            public:
                CClient(cloudx1_CInstance* instance)
                    {
                        assert(instance);
                        client=cloudx1_NewClient(instance);
                        if (!client) throw std::bad_alloc();
                    }
                // non-copyable
                CClient( const CInstance& ) = delete;
                CClient& operator=( const CInstance& ) = delete;
                ~CClient()
                    {
                        if (client) {
                            cloudx1_DeleteClient(client);
                            client=nullptr;
                        }
                    }
                std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
                    {
                        assert(client);
                        int need_save=0;
                        const char *canswer = cloudx1_ProcessJson(client, request.c_str(), &need_save);
                        if (!canswer) return {};
                        std::string answer(canswer);
                        cloudx1_FreeJsonAnswer (canswer);
                        if (NeedSave) {
                            *NeedSave=(need_save!=0);
                        }
                        return answer;
                    }
            private:
                cloudx1_CClientInstance *client = nullptr;
        };
        std::shared_ptr<CClient> NewClient()
            {
                assert(instance);
                return std::make_shared<CClient>(instance);
            }

    protected:
            template<typename T>
        unsigned int AddAudio(const T* payload, unsigned int nFrame)
            {
                assert(instance);
                return cloudx1_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame, helper::SampleFormat<T>::format);
            }
            template<typename T>
```

```
        unsigned int GetAudio(T* payload, unsigned int max_nFrame)
            {
                assert(instance);
                return cloudx1_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame, helper::SampleFormat<T>::format);
            }

    private:
        cloudx1_CParameters* params = nullptr;
        cloudx1_CInstance* instance = nullptr;
        std::thread update_thread;
        std::vector< std::pair<std::string,std::string> > init_metadata;
#if CLOUDX1_HAS_WEBSERVER
        uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
#endif // CLOUDX1_HAS_WEBSERVER
    };

}; // namespace cloudx1
}; // namespace sound4

// bridge C callbacks to CPresetLoader
extern "C" {
    static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        std::string content=preset_loader->Read(filename);
        return strdup(content.c_str());
    }
    static void cloudx1_custom_reader_free(char *content, void* userdata) {
        if (!content) return;
        free(content);
    }
    static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        auto res=preset_loader->Write(filename,content);
        return res?0:-1;
    }
    static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        auto res=preset_loader->Exists(filename);
        return res?0:-1;
    }
    static fs_char** cloudx1_custom_getall(void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        auto res=preset_loader->GetAll();
        fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
        for (size_t n=0;n<res.size();n++) {
            all[n]=fs_strdup(res[n].c_str());
        }
        all[res.size()]=nullptr;
        return all;
    }
    static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
        if (!all) return;
        for (fs_char** one=all;*one;one++) {
            free(*one);
        }
        free(all);
    }
    static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        auto res=preset_loader->Remove(filename);
        return res?0:-1;
    }
    static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
        sound4::cloudx1::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::CPresetLoader*>(userdata);
        auto res=preset_loader->Rename(from,to);
        return res?0:-1;
    }

};
```

11.11 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.
- struct sound4::cloudx1::dyn::helper::SampleFormat< int16_t >
- struct sound4::cloudx1::dyn::helper::SampleFormat< int32_t >
- struct sound4::cloudx1::dyn::helper::SampleFormat< float >
- class sound4::cloudx1::dyn::CPresetLoader
- class sound4::cloudx1::dyn::CInstance
- class sound4::cloudx1::dyn::CInstance::CClient

Namespaces

- sound4
- sound4::cloudx1
- dyn
- sound4::cloudx1::dyn
- sound4::cloudx1::dyn::helper

Macros

- #define fs_strdup wcsdup

  helper for OS dependant file name copy

Typedefs

- using sound4::cloudx1::dyn::helper::DynLib_t = HMODULE
- using sound4::cloudx1::dyn::helper::DynFunc_t = FARPROC
- 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

Variables

- const size_t sound4::cloudx1::dyn::ChunkFrames =CLOUDX1_AUDIOFRAME_COUNT
- const size_t sound4::cloudx1::dyn::InputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_INPUT_COUNT
- const size_t sound4::cloudx1::dyn::OutputChannels =CLOUDX1_CHANNEL_COUNT∗CLOUDX1_AUDIO_OUTPUT_COUNT
- 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 =CLOUDX1_SAMPLE_RATE

11.11.1 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.

11.11.2 Macro Definition Documentation

11.11.2.1 fs_strdup

```
#define fs_strdup wcsdup
```

Definition at line 36 of file sound4.x1.cloud_dyn.hpp.

11.12 sound4.x1.cloud_dyn.hpp

```
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 #ifdef __unix__
00021     #include <dlfcn.h>
00022 #elif defined(_WIN32)
00023     #include <libloaderapi.h>
00024 #else
00025     #error "Unsupported OS"
00026 #endif
00027
00028 // Needed to have function declarations
00029 #include "sound4.x1.cloud.h"
00030
00031
00032 #ifdef _WIN32
00033     #include <stringapiset.h>
00034     #include <wchar.h>
00036     #define fs_strdup wcsdup
00037 #else // !_WIN32
00039     #define fs_strdup strdup
00040 #endif // !_WIN32
00041
00042 // bridge C callbacks to CPresetLoader
00043 extern "C" {
```

```
    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);
};

namespace sound4 {
namespace cloudx1 {
namespace dyn {

    namespace helper {
        // Storage definition, depending on OS
        #ifdef __unix__
            using DynLib_t = void *;
            using DynFunc_t = void *;
        #else
            using DynLib_t = HMODULE;
            using DynFunc_t = FARPROC;
        #endif

            template <typename T>
        class DynFuncHolder {
        private:
            DynFunc_t m_ptr = NULL;
        public:
            DynFuncHolder() = default;
            DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
            operator bool() const { return (m_ptr!=NULL); }
            bool IsOk() const { return m_ptr!=NULL; }
            operator T* () const { return reinterpret_cast<T*>(m_ptr); }
        };

        class CDynLoader {
            private:
                DynLib_t m_lib {};
            public:
                CDynLoader() = default;
                CDynLoader(CDynLoader&&) = default;
                // No copy allowed
                CDynLoader(const CDynLoader&) = delete;
                CDynLoader& operator=(CDynLoader const&) = delete;
                operator bool() const {
                    return (m_lib!=NULL);
                }
                bool IsOk() const {
                    return (m_lib!=NULL);
                }
                #ifdef __unix__
                    void Close() {
                        if (IsOk()) {
                            #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
                            dlclose(m_lib);
                            #endif // __SANITIZE_ADDRESS__
                            m_lib=NULL;
                        }
                    }
                    bool Load(const std::filesystem::path& dynlib)
                    {
                        Close();
                        m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
                        return IsOk();
                    }
                        template <typename T>
                    DynFuncHolder<T> GetSymbol(const std::string& name) {
                        auto ptr=dlsym(m_lib, name.c_str());
                        if (!ptr) {
                            throw std::runtime_error("Missing function in library");
                        }
                        return DynFuncHolder<T>(ptr);
                    }
                        template <typename T>
                    DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
                        auto ptr=dlsym(m_lib, name.c_str());
                        return DynFuncHolder<T>(ptr);
                    }
                    static std::filesystem::path GetThisLibraryPath(void) {
                        static Dl_info info;
                        if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
                        auto rp=realpath(info.dli_fname, NULL);
                        std::filesystem::path p(rp);
                        free(rp);
                        return p;
                    }
```

```
                #else
                    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_
                    {
                        Close();
                        m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
                        return IsOk();
                    }
                    void Close() {
                        if (IsOk()) {
                            // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use FreeLibraryAndExitThread but...
                            // So define SOUND4_CALL_FREELIBRARY before including this file if you really want to unload the library dynamically
                            #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
                                FreeLibraryAndExitThread(m_lib, 0);
                            #elif defined(SOUND4_CALL_FREELIBRARY)
                                FreeLibrary(m_lib);
                            #endif // SOUND4_CALL_FREELIBRARY
                            m_lib=NULL;
                        }
                    }
                        template <typename T>
                    DynFuncHolder<T> GetSymbol(const std::string& name) {
                        auto ptr=GetProcAddress(m_lib, name.c_str());
                        if (!ptr) {
                            throw std::runtime_error("Missing function in library");
                        }
                        return DynFuncHolder<T>(ptr);
                    }
                        template <typename T>
                    DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
                        auto ptr=GetProcAddress(m_lib, name.c_str());
                        return DynFuncHolder<T>(ptr);
                    }
                    static std::filesystem::path GetThisLibraryPath(void) {
                        static wchar_t path[MAX_PATH]={};
                        HMODULE hm = NULL;
                        if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
                                GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
                                (LPCWSTR) &GetThisLibraryPath, &hm))
                        {
                            if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
                                return std::filesystem::path(path);
                            }
                        }
                        return {};
                    }
                #endif
                virtual ~CDynLoader() {
                    Close();
                }
        }; // class CDynLoader
    }; // namespace helper

    class CDynLib {
    private:
        helper::CDynLoader m_lib;
        bool m_bOK = false;
    public:
        CDynLib() = default;

        // All those wrapped functions have the same signature as there C cloudx1_XXX
        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              ;
#if CLOUDX1_HAS_WEBSERVER
        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       ;
#endif // CLOUDX1_HAS_WEBSERVER

        // added 2023-03-22
        helper::DynFuncHolder< decltype(cloudx1_StereoToMono_Planar    ) > StereoToMono_Planar    ;
        helper::DynFuncHolder< decltype(cloudx1_MonoToStereo_Planar    ) > MonoToStereo_Planar    ;

        // added 2023-05-22
        helper::DynFuncHolder< decltype(cloudx1_SetMetadata            ) > SetMetadata            ;
        helper::DynFuncHolder< decltype(cloudx1_GetMetadataInfos       ) > GetMetadataInfos       ;
        helper::DynFuncHolder< decltype(cloudx1_FreeMetadataInfos      ) > FreeMetadataInfos      ;

        // added 2023-06-20
        helper::DynFuncHolder< decltype(cloudx1_SetPresetManager                ) > SetPresetManager
    ;
        helper::DynFuncHolder< decltype(cloudx1_PresetManager_InformChange      ) >
    PresetManager_InformChange      ;

        // added 2023-09-21
        helper::DynFuncHolder< decltype(cloudx1_GetPossibleChunkSizeInFrames    ) >
    GetPossibleChunkSizeInFrames   ;
        helper::DynFuncHolder< decltype(cloudx1_GetProcessChunkFrames           ) >
    GetProcessChunkFrames   ;
        helper::DynFuncHolder< decltype(cloudx1_InitProcess3                    ) > InitProcess3
    ;

#if CLOUDX1_HAS_CLOUDBUS
        // added 2023-06-26
        helper::DynFuncHolder< decltype(cloudx1_NewBus                 ) > NewBus      ;
        helper::DynFuncHolder< decltype(cloudx1_FreeBus                ) > FreeBus      ;
        helper::DynFuncHolder< decltype(cloudx1_SetInstanceBus         ) > SetInstanceBus      ;
#endif // CLOUDX1_HAS_CLOUDBUS

        // added 2024-02-19
        helper::DynFuncHolder< decltype(cloudx1_SetMetadataMulti       ) > SetMetadataMulti         ;

#if CLOUDX1_HAS_WEBSERVER
        // added 2024-05-23
        helper::DynFuncHolder< decltype(cloudx1_Webserver_SetAppHealth    ) > Webserver_SetAppHealth
    ;
        helper::DynFuncHolder< decltype(cloudx1_Webserver_GetAppHealth    ) > Webserver_GetAppHealth
    ;
        helper::DynFuncHolder< decltype(cloudx1_Webserver_FreeString      ) > Webserver_FreeString
    ;
#endif // CLOUDX1_HAS_WEBSERVER

        bool Load(const std::filesystem::path& path = {}) {
            std::string filename;
            #ifdef __unix__
                filename="libsound4.x1.cloud.so";
            #else
                filename="sound4.x1.cloud.dll";
            #endif
            m_lib.Close();
            // If a path is given, use it directly and do not try other path
            if (!path.empty() && !m_lib.Load(path / filename)) {
                return false;
            } else if (!m_lib.IsOk()) {
                auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
                #ifndef _WIN32
                // Linux: if in a bin directory, try ../lib first
                if (!m_lib.IsOk() && thisdir.filename()=="bin") {
```

```
                    auto libdir = thisdir.parent_path() / "lib";
                    m_lib.Load(libdir / filename);
                }
                #endif // !_WIN32
                if (!m_lib.IsOk()) {
                    // Search in the same directory this code is in
                    m_lib.Load(thisdir / filename);
                }
                if (!m_lib.IsOk()) {
                    // Try current path
                    m_lib.Load(std::filesystem::current_path() / filename);
                }
                if (!m_lib.IsOk()) {
                    return false;
                }
            }
            // Load all C functions
            try {
                GetVersion              = m_lib.GetSymbol< decltype(cloudx1_GetVersion              )
>("cloudx1_GetVersion"             );
                GetChunkSizeInFrames    = m_lib.GetSymbol< decltype(cloudx1_GetChunkSizeInFrames    )
>("cloudx1_GetChunkSizeInFrames"   );
                GetChannelCount         = m_lib.GetSymbol< decltype(cloudx1_GetChannelCount         )
>("cloudx1_GetChannelCount"        );
                GetAudioInputCount      = m_lib.GetSymbol< decltype(cloudx1_GetAudioInputCount      )
>("cloudx1_GetAudioInputCount"     );
                GetAudioOutputCount     = m_lib.GetSymbol< decltype(cloudx1_GetAudioOutputCount     )
>("cloudx1_GetAudioOutputCount"    );
                GetSampleRate           = m_lib.GetSymbol< decltype(cloudx1_GetSampleRate           )
>("cloudx1_GetSampleRate"          );
                SetLoggerCallback       = m_lib.GetSymbol< decltype(cloudx1_SetLoggerCallback       )
>("cloudx1_SetLoggerCallback"      );
                SetLogSeverity          = m_lib.GetSymbol< decltype(cloudx1_SetLogSeverity          )
>("cloudx1_SetLogSeverity"         );
                NewParameters           = m_lib.GetSymbol< decltype(cloudx1_NewParameters           )
>("cloudx1_NewParameters"          );
                FreeParameters          = m_lib.GetSymbol< decltype(cloudx1_FreeParameters          )
>("cloudx1_FreeParameters"         );
                SetParameter            = m_lib.GetSymbol< decltype(cloudx1_SetParameter            )
>("cloudx1_SetParameter"           );
                GetParameter            = m_lib.GetSymbol< decltype(cloudx1_GetParameter            )
>("cloudx1_GetParameter"           );
                FreeParameterValue      = m_lib.GetSymbol< decltype(cloudx1_FreeParameterValue      )
>("cloudx1_FreeParameterValue"     );
                InitProcess             = m_lib.GetSymbol< decltype(cloudx1_InitProcess             )
>("cloudx1_InitProcess"            );
                InitProcess2            = m_lib.GetSymbol< decltype(cloudx1_InitProcess2            )
>("cloudx1_InitProcess2"           );
                TerminateProcess        = m_lib.GetSymbol< decltype(cloudx1_TerminateProcess        )
>("cloudx1_TerminateProcess"       );
                ExitProcess             = m_lib.GetSymbol< decltype(cloudx1_ExitProcess             )
>("cloudx1_ExitProcess"            );
                StartUpdateThread       = m_lib.GetSymbol< decltype(cloudx1_StartUpdateThread       )
>("cloudx1_StartUpdateThread"      );
                StopUpdateThread        = m_lib.GetSymbol< decltype(cloudx1_StopUpdateThread        )
>("cloudx1_StopUpdateThread"       );
                WaitUpdateThreadReady   = m_lib.GetSymbol< decltype(cloudx1_WaitUpdateThreadReady   )
>("cloudx1_WaitUpdateThreadReady"  );
                ProcessAudio            = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio            )
>("cloudx1_ProcessAudio"           );
                ProcessAudio_Planar     = m_lib.GetSymbol< decltype(cloudx1_ProcessAudio_Planar     )
>("cloudx1_ProcessAudio_Planar"    );
                GetBufferIn             = m_lib.GetSymbol< decltype(cloudx1_GetBufferIn             )
>("cloudx1_GetBufferIn"            );
                GetBufferOut            = m_lib.GetSymbol< decltype(cloudx1_GetBufferOut            )
>("cloudx1_GetBufferOut"           );
                GetEstimatedDelay       = m_lib.GetSymbol< decltype(cloudx1_GetEstimatedDelay       )
>("cloudx1_GetEstimatedDelay"      );
                GetFormatName           = m_lib.GetSymbol< decltype(cloudx1_GetFormatName           )
>("cloudx1_GetFormatName"          );
                GetFormatFromName       = m_lib.GetSymbol< decltype(cloudx1_GetFormatFromName       )
>("cloudx1_GetFormatFromName"      );
                GetBytesFromFormat      = m_lib.GetSymbol< decltype(cloudx1_GetBytesFromFormat      )
>("cloudx1_GetBytesFromFormat"     );
                GetMaxPacketFrame       = m_lib.GetSymbol< decltype(cloudx1_GetMaxPacketFrame       )
>("cloudx1_GetMaxPacketFrame"      );
                AddAudio                = m_lib.GetSymbol< decltype(cloudx1_AddAudio                )
>("cloudx1_AddAudio"               );
                AddPadAudio             = m_lib.GetSymbol< decltype(cloudx1_AddPadAudio             )
>("cloudx1_AddPadAudio"            );
                GetOutputCount          = m_lib.GetSymbol< decltype(cloudx1_GetOutputCount          )
>("cloudx1_GetOutputCount"         );
                GetAudio                = m_lib.GetSymbol< decltype(cloudx1_GetAudio                )
>("cloudx1_GetAudio"               );
                AudioConvertFrom        = m_lib.GetSymbol< decltype(cloudx1_AudioConvertFrom        )
>("cloudx1_AudioConvertFrom"       );
                AudioConvertTo          = m_lib.GetSymbol< decltype(cloudx1_AudioConvertTo          )
```

```
>("cloudx1_AudioConvertTo"          );
                StereoToMono            = m_lib.GetSymbol< decltype(cloudx1_StereoToMono           )
>("cloudx1_StereoToMono"            );
                MonoToStereo            = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo           )
>("cloudx1_MonoToStereo"            );
                AudioMonoFromLiveStereo = m_lib.GetSymbol< decltype(cloudx1_AudioMonoFromLiveStereo)
>("cloudx1_AudioMonoFromLiveStereo");
                AudioMonoToLiveStereo   = m_lib.GetSymbol< decltype(cloudx1_AudioMonoToLiveStereo  )
>("cloudx1_AudioMonoToLiveStereo"  );
                NewClient               = m_lib.GetSymbol< decltype(cloudx1_NewClient              )
>("cloudx1_NewClient"               );
                DeleteClient            = m_lib.GetSymbol< decltype(cloudx1_DeleteClient           )
>("cloudx1_DeleteClient"            );
                ProcessJson             = m_lib.GetSymbol< decltype(cloudx1_ProcessJson            )
>("cloudx1_ProcessJson"             );
                FreeJsonAnswer          = m_lib.GetSymbol< decltype(cloudx1_FreeJsonAnswer         )
>("cloudx1_FreeJsonAnswer"          );
                SaveState               = m_lib.GetSymbol< decltype(cloudx1_SaveState              )
>("cloudx1_SaveState"               );
#if CLOUDX1_HAS_WEBSERVER
                Webserver_tcp           = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp          )
>("cloudx1_Webserver_tcp"           );
                Webserver_tcp2          = m_lib.GetSymbol< decltype(cloudx1_Webserver_tcp2         )
>("cloudx1_Webserver_tcp2"          );
                Webserver               = m_lib.GetSymbol< decltype(cloudx1_Webserver              )
>("cloudx1_Webserver"               );
                Webserver_Stop          = m_lib.GetSymbol< decltype(cloudx1_Webserver_Stop         )
>("cloudx1_Webserver_Stop"          );
                Webserver_Status        = m_lib.GetSymbol< decltype(cloudx1_Webserver_Status       )
>("cloudx1_Webserver_Status"        );
#endif // CLOUDX1_HAS_WEBSERVER

                // added 2023-03-22
                StereoToMono_Planar     = m_lib.GetSymbol< decltype(cloudx1_StereoToMono_Planar    )
>("cloudx1_StereoToMono_Planar"     );
                MonoToStereo_Planar     = m_lib.GetSymbol< decltype(cloudx1_MonoToStereo_Planar    )
>("cloudx1_MonoToStereo_Planar"     );

            } catch (std::runtime_error& ) {
                return false;
            }
            // C functions allowed to be missing
            try {
                // added 2023-05-22
                SetMetadata             = m_lib.GetSymbol< decltype(cloudx1_SetMetadata            )
>("cloudx1_SetMetadata"             );
                GetMetadataInfos        = m_lib.GetSymbol< decltype(cloudx1_GetMetadataInfos       )
>("cloudx1_GetMetadataInfos"        );
                FreeMetadataInfos       = m_lib.GetSymbol< decltype(cloudx1_FreeMetadataInfos      )
>("cloudx1_FreeMetadataInfos"       );
                // added 2023-06-20
                SetPresetManager            = m_lib.GetSymbol< decltype(cloudx1_SetPresetManager
        ) >("cloudx1_SetPresetManager"              );
                PresetManager_InformChange  = m_lib.GetSymbol<
decltype(cloudx1_PresetManager_InformChange     ) >("cloudx1_PresetManager_InformChange"    );

                // added 2023-09-21
                GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
decltype(cloudx1_GetPossibleChunkSizeInFrames  ) >("cloudx1_GetPossibleChunkSizeInFrames"  );
                GetProcessChunkFrames        = m_lib.GetSymbol< decltype(cloudx1_GetProcessChunkFrames
        ) >("cloudx1_GetProcessChunkFrames"         );
                InitProcess3                 = m_lib.GetSymbol< decltype(cloudx1_InitProcess3
        ) >("cloudx1_InitProcess3"                  );

#if CLOUDX1_HAS_CLOUDBUS
                // added 2023-06-26
                NewBus                  = m_lib.GetSymbol< decltype(cloudx1_NewBus                 )
>("cloudx1_NewBus"                  );
                FreeBus                 = m_lib.GetSymbol< decltype(cloudx1_FreeBus                )
>("cloudx1_FreeBus"                 );
                SetInstanceBus          = m_lib.GetSymbol< decltype(cloudx1_SetInstanceBus         )
>("cloudx1_SetInstanceBus"          );
#endif // CLOUDX1_HAS_CLOUDBUS

                // added 2024-02-19
                SetMetadataMulti        = m_lib.GetSymbol< decltype(cloudx1_SetMetadataMulti       )
>("cloudx1_SetMetadataMulti"        );

#if CLOUDX1_HAS_WEBSERVER
                // added 2024-05-23
                Webserver_SetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_SetAppHealth
      ) >("Webserver_SetAppHealth"      );
                Webserver_GetAppHealth      = m_lib.GetSymbol< decltype(cloudx1_Webserver_GetAppHealth
      ) >("Webserver_GetAppHealth"      );
                Webserver_FreeString        = m_lib.GetSymbol< decltype(cloudx1_Webserver_FreeString
      ) >("Webserver_FreeString"        );
#endif // CLOUDX1_HAS_WEBSERVER
```

```
            }  catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }

            m_bOK=true;
            return true;
        }
        bool IsOk() const {
            return m_bOK;
        }

    };



    static constexpr const char *process_name = "SOUND4 x1.CLOUD";
    static constexpr const char *process_shortname = "x1";
    using SampleFormat = cloudx1_SampleFormat;
    const size_t ChunkFrames=CLOUDX1_AUDIOFRAME_COUNT;
    const size_t InputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_INPUT_COUNT;
    const size_t OutputChannels=CLOUDX1_CHANNEL_COUNT*CLOUDX1_AUDIO_OUTPUT_COUNT;
    const size_t InputSampleSize=ChunkFrames*InputChannels;
    const size_t OutputSampleSize=ChunkFrames*OutputChannels;
    const size_t SampleRate=CLOUDX1_SAMPLE_RATE;

    namespace helper {
        #ifdef _WIN32
        static inline std::string WStringToUTF8(const std::wstring& wstr) {
            if (wstr.empty()) return {};
            int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
            std::string str(size, 0);
            WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
            return str;
        }
        static inline std::wstring UTF8ToWString(const std::string& str) {
            if (str.empty()) return std::wstring();
            int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
            std::wstring wstr(size, 0);
            MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
            return wstr;
        }
        #endif // _WIN32

            template<typename T>
        struct SampleFormat {
        };
            template<>
        struct SampleFormat<int16_t> {
            const cloudx1_SampleFormat format = S16_NATIVE;
        };
            template<>
        struct SampleFormat<int32_t> {
            const cloudx1_SampleFormat format = S32_NATIVE;
        };
            template<>
        struct SampleFormat<float> {
            const cloudx1_SampleFormat format = F32_NATIVE;
        };


        static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output, size_t nSpl, cloudx1_SampleFormat fmt) {
            dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
        }
        static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload, size_t nSpl, cloudx1_SampleFormat fmt) {
            dynlib.AudioConvertTo(input, payload, nSpl, fmt);
        }
        static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t nSpl) {
            dynlib.StereoToMono(input, output, nSpl);
        }
        static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t nSpl) {
            dynlib.MonoToStereo(input, output, nSpl);
        }
        static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float *inputR, float *output, size_t nSpl) {
            dynlib.StereoToMono_Planar(inputL, inputR, output, nSpl);
        }
        static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL, float *outputR, size_t nSpl) {
            dynlib.MonoToStereo_Planar(input, outputL, outputR, nSpl);
        }
        static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
```

```
*output) {
            dynlib.AudioMonoFromLiveStereo(payload, output);
        }
        static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
*payload) {
            dynlib.AudioMonoToLiveStereo(input, payload);
        }

    }; // namespace helper

    static inline std::string GetVersion(CDynLib& dynlib)
        {
            return std::string(dynlib.GetVersion());
        }

    static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
        {
            return dynlib.GetChunkSizeInFrames();
        }

    static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
        {
            std::vector<unsigned int> list;
            if (dynlib.GetPossibleChunkSizeInFrames) {
                for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
                    list.push_back(*src);
                }
            } else {
                list.push_back(64);
            }
            return list;
        }

    static inline unsigned int GetChannelCount(CDynLib& dynlib)
        {
            return dynlib.GetChannelCount();
        }

    static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
        {
            return dynlib.GetAudioInputCount();
        }

    static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
        {
            return dynlib.GetAudioOutputCount();
        }

    static inline unsigned int GetSampleRate(CDynLib& dynlib)
        {
            return dynlib.GetSampleRate();
        }

    static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
    {
        // Do some sanity checks
        if (dynlib.GetSampleRate()!=CLOUDX1_SAMPLE_RATE) {
            throw std::runtime_error("Bad library sampling rate");
        }
        if (a_bCheckFrames && dynlib.GetChunkSizeInFrames()!=ChunkFrames) {
            throw std::runtime_error("Bad library frame size");
        }
        if (dynlib.GetChannelCount()!=CLOUDX1_CHANNEL_COUNT) {
            throw std::runtime_error("Bad library channel count");
        }
        if (dynlib.GetAudioInputCount()!=CLOUDX1_AUDIO_INPUT_COUNT) {
            throw std::runtime_error("Bad library input count");
        }
        if (dynlib.GetAudioOutputCount()!=CLOUDX1_AUDIO_OUTPUT_COUNT) {
            throw std::runtime_error("Bad library output count");
        }
    }

    static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
        {
            return std::string(dynlib.GetFormatName(fmt));
        }

    static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
        {
            return dynlib.GetFormatFromName(name.c_str());
        }

    static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
        {
            return dynlib.GetBytesFromFormat(fmt);
        }
```

```
    using LogSeverity = cloudx1_LogSeverity;
    using log_cb_t = std::function<void(LogSeverity,const std::string&)>;

    static log_cb_t _log_cb;
    extern "C" {
        static inline void _log_cb_c(cloudx1_LogSeverity severity, const char *c_msg) {
            _log_cb(severity, std::string(c_msg));
        }
    }
    static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
        {
            dynlib.SetLogSeverity(severity);
        }
    static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
        {
            _log_cb=cb;
            dynlib.SetLoggerCallback(&_log_cb_c);
        }

#if CLOUDX1_HAS_CLOUDBUS
    class CBus {
        public:
            CBus(CDynLib& dynlib)
                : m_dynlib(dynlib)
                {
                    if (m_dynlib.NewBus) {
                        m_bus=m_dynlib.NewBus();
                    }
                }
            ~CBus()
                {
                    if (m_bus && m_dynlib.FreeBus) {
                        m_dynlib.FreeBus(m_bus);
                    }
                    m_bus = nullptr;
                }
            cloudx1_CBus *Get() const { return m_bus; }
        private:
            CDynLib& m_dynlib;
            cloudx1_CBus *m_bus;
    };
#endif // CLOUDX1_HAS_CLOUDBUS

    class CPresetLoader {
    public:
        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;
    };
    using PresetChange_Kind = cloudx1_PresetChange_Kind;

    class CInstance {
    public:
        CInstance(CDynLib& dynlib)
        : m_dynlib(dynlib)
            {
                params = m_dynlib.NewParameters();
                if (!params) throw std::bad_alloc();
            }
        // non-copyable
        CInstance( const CInstance& ) = delete;
        CInstance& operator=( const CInstance& ) = delete;
        ~CInstance()
            {
                Stop();
                if (params) m_dynlib.FreeParameters(params);
            }
       bool IsOk() const { return instance!=nullptr; }
        void SetParam(const std::string& name, const std::string& value)
            {
                assert(params);
                m_dynlib.SetParameter(params, name.c_str(), value.c_str());
            }
        std::string GetParam(const std::string& name)
            {
                assert(params);
                auto c_value=m_dynlib.GetParameter(params, name.c_str());
                std::string ret(c_value);
                m_dynlib.FreeParameterValue(c_value);
```

```
                return ret;
            }
#if CLOUDX1_HAS_CLOUDBUS
            void SetBus(const CBus& bus) {
                assert(params);
                if (m_dynlib.SetInstanceBus) {
                    m_dynlib.SetInstanceBus(params,bus.Get());
                }
            }
#endif // CLOUDX1_HAS_CLOUDBUS

        void SetPresetManager(CPresetLoader *preset_manager)
            {
                if (!preset_manager) return;
                if (!m_dynlib.SetPresetManager) return;
                assert(params);
                m_dynlib.SetPresetManager(params,
                    cloudx1_custom_reader,
                    cloudx1_custom_reader_free,
                    cloudx1_custom_writer,
                    cloudx1_custom_exists,
                    cloudx1_custom_getall,
                    cloudx1_custom_getall_free,
                    cloudx1_custom_remove,
                    cloudx1_custom_rename,
                    preset_manager->IsReadOnly(),
                    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)
            {
                assert(!instance);
                std::string l_save_path;
                if (!save_path.empty()) {
                    l_save_path = save_path.u8string();
                }
                if (m_dynlib.InitProcess3) {
                    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);
                } else if (frames_per_chunk!=64) {
                    return false;
                } else {
                    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);
                }
                if (!instance) return false;
                if (!init_metadata.empty()) {
                    for (auto&& [key,value]: init_metadata) {
                        m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
                    }
                    init_metadata.clear();
                }
                update_thread = std::thread([this, json_port](){
                    OnUpdateThreadStart();
                    m_dynlib.StartUpdateThread(instance, json_port);
                    OnUpdateThreadStop();
                });
                int timeout=5000;
                #ifdef DEBUG
                    timeout=60*1000; // For debugger pauses
                #endif
                if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
                    return false;
                }
                return true;
            }

        unsigned int GetChunkFrames() {
            if (instance && m_dynlib.GetProcessChunkFrames) {
                return m_dynlib.GetProcessChunkFrames(instance);
            }
            return 64;
        }

        void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind change_kind)
            {
                if (m_dynlib.PresetManager_InformChange) {
                    m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(), (cloudx1_PresetChange_Kind) change_kind);
                }
            }

        void SetMetadata(const std::string& key, const char* value)
```

```
            {
                if (m_dynlib.SetMetadata) {
                    if (instance) {
                        m_dynlib.SetMetadata(instance, key.c_str(), value);
                    } else {
                        init_metadata.push_back( {key, value} );
                    }
                }
            }


        void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
            {
                if (instance) {
                    if (m_dynlib.SetMetadataMulti) {
                        typedef const char* pchar;
                        pchar* keyvalue=new pchar[2*list.size()+1];
                        size_t n=0;
                        for (auto&& [key,value]: list) {
                            keyvalue[2*n+0]=key.c_str();
                            keyvalue[2*n+1]=value;
                            n++;
                        }
                        keyvalue[2*n+0]=nullptr;

                        cloudx1_SetMetadataMulti(instance, keyvalue);

                        delete[] keyvalue;
                    } else {
                        for (auto&& [key,value]: list) {
                            SetMetadata(key, value);
                        }
                    }
                } else {
                    for (auto&& [key,value]: list) {
                        if (value) {
                            init_metadata.push_back( {key, value} );
                        }
                    }
                }
            }

        std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
            {
                std::vector< std::tuple<std::string,std::string> > values;
                if (m_dynlib.GetMetadataInfos) {
                    const char** c_values = m_dynlib.GetMetadataInfos(instance);
                    if (c_values) {
                        for (const char** c_value=c_values; *c_value; ) {
                            std::string key(*c_value);
                            c_value++;
                            if (*c_value) {
                                std::string descr(*c_value);
                                c_value++;
                                values.push_back( {key,descr} );
                            }
                        }
                        m_dynlib.FreeMetadataInfos(instance, c_values);
                    }
                }
                return values;
            }

        // Can be overloaded, called at start of UpdateThread in the thread
        virtual void OnUpdateThreadStart() {}
        // Can be overloaded to call at end of UpdateThread in the thread
        virtual void OnUpdateThreadStop() {}

#if CLOUDX1_HAS_WEBSERVER
        bool StartWebServer(int http_port, int https_port=0)
            {
                assert(instance);
                assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
                webserver = m_dynlib.Webserver(http_port,https_port,instance);
                if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
                    return false;
                }
                int web_status=m_dynlib.Webserver_Status(webserver);
                if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
                    return false;
                }
                if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
                    return false;
                }
                return true;
            }
        void StopWebServer()
```

```
            {
                if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
                    m_dynlib.Webserver_Stop(webserver, 1000);
                    webserver = SOUND4_INVALID_WEBSERVER_ID;
                }
            }

        void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string& content)
            {
                assert(instance);
                if (!m_dynlib.Webserver_SetAppHealth) return;
                m_dynlib.Webserver_SetAppHealth(instance, httpcode, contenttype.c_str(), content.c_str());
            }
        void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
            {
                assert(instance);
                if (!m_dynlib.Webserver_GetAppHealth) return;
                char* c_contenttype=nullptr;
                char* c_content=nullptr;
                m_dynlib.Webserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
                contenttype=c_contenttype;
                content=c_content;
                m_dynlib.Webserver_FreeString(c_contenttype);
                m_dynlib.Webserver_FreeString(c_content);
            }
#endif // CLOUDX1_HAS_WEBSERVER

        int Terminate()
            {
                if (!instance) return 0;
                return m_dynlib.TerminateProcess(instance);
            }

        void Stop()
            {
                if (!instance) return;
#if CLOUDX1_HAS_WEBSERVER
                StopWebServer();
#endif // CLOUDX1_HAS_WEBSERVER
                if (update_thread.joinable()) {
                    m_dynlib.StopUpdateThread(instance); // this will exit the update thread
                    update_thread.join();
                }
                m_dynlib.ExitProcess(instance);
                instance=nullptr;
            }

        unsigned int GetEstimatedDelay()
            {
                assert(instance);
                return m_dynlib.GetEstimatedDelay(instance);
            }

        std::array<float,InputSampleSize>& GetBufferIn()
            {
                assert(instance);
                float *buf=m_dynlib.GetBufferIn(instance);
                return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);

            }
        std::array<float,OutputSampleSize>& GetBufferOut()
            {
                assert(instance);
                float *buf= m_dynlib.GetBufferOut(instance);
                return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
            }
        void ProcessAudio()
            {
                assert(instance);
                m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance), m_dynlib.GetBufferOut(instance));
            }
        // void ProcessAudio_Planar(const std::array< const std::array<const float,ChunkFrames>, InputChannels>& input, std::array< std::array<float,ChunkFrames>, OutputChannels>& output)
        void ProcessAudio_Planar(float const * const *input, float * const *output)
            {
                assert(instance);
                m_dynlib.ProcessAudio_Planar(instance, input, output);
            }

            template<typename T>
        std::vector<T> ProcessAnyAudio(const std::vector<T> input)
            {
                assert(instance);
                std::vector<T> output;
```

```
                unsigned int out_offset=0;
                unsigned int in_offset=0;
                unsigned int todo = input.size();
                while (todo>0) {
                    unsigned int left = AddAudio(&input[in_offset], todo);
                    unsigned int out_avail = m_dynlib.GetOutputCount(instance);
                    output.resize(out_offset + out_avail);
                    GetAudio(&output[out_offset], out_avail);
                    out_offset+=out_avail;
                    in_offset += todo-left;
                    todo=left;
                }
                return output;
            }

        class CClient {
            public:
                CClient(cloudx1_CInstance* instance, CDynLib& dynlib)
                : m_dynlib(dynlib)
                    {
                        assert(instance);
                        client=m_dynlib.NewClient(instance);
                        if (!client) throw std::bad_alloc();
                    }
                // non-copyable
                CClient( const CInstance& ) = delete;
                CClient& operator=( const CInstance& ) = delete;
                ~CClient()
                    {
                        if (client) {
                            m_dynlib.DeleteClient(client);
                            client=nullptr;
                        }
                    }
                std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
                    {
                        assert(client);
                        int need_save=0;
                        const char *canswer = m_dynlib.ProcessJson(client, request.c_str(), &need_save);
                        if (!canswer) return {};
                        std::string answer(canswer);
                        m_dynlib.FreeJsonAnswer (canswer);
                        if (NeedSave) {
                            *NeedSave=(need_save!=0);
                        }
                        return answer;
                    }
            private:
                CDynLib& m_dynlib;
                cloudx1_CClientInstance *client = nullptr;
        };
        std::shared_ptr<CClient> NewClient()
            {
                assert(instance);
                return std::make_shared<CClient>(instance, m_dynlib);
            }

    protected:
            template<typename T>
        unsigned int AddAudio(const T* payload, unsigned int nFrame)
            {
                assert(instance);
                return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame, helper::SampleFormat<T>::format);
            }
            template<typename T>
        unsigned int GetAudio(T* payload, unsigned int max_nFrame)
            {
                assert(instance);
                return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame, helper::SampleFormat<T>::format);
            }

    private:
        CDynLib& m_dynlib;
        cloudx1_CParameters* params = nullptr;
        cloudx1_CInstance* instance = nullptr;
        std::thread update_thread;
        std::vector< std::pair<std::string,std::string> > init_metadata;
#if CLOUDX1_HAS_WEBSERVER
        uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
#endif // CLOUDX1_HAS_WEBSERVER
    };

}; // namespace dyn
}; // namespace cloudx1
```

```
}; // namespace sound4


// bridge C callbacks to CPresetLoader
extern "C" {
    static char *cloudx1_custom_reader(const fs_char *filename, void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        std::string content=preset_loader->Read(filename);
        return strdup(content.c_str());
    }
    static void cloudx1_custom_reader_free(char *content, void* userdata) {
        if (!content) return;
        free(content);
    }
    static int cloudx1_custom_writer(const fs_char *filename, const char *content, void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        auto res=preset_loader->Write(filename,content);
        return res?0:-1;
    }
    static int cloudx1_custom_exists(const fs_char *filename, void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        auto res=preset_loader->Exists(filename);
        return res?0:-1;
    }
    static fs_char** cloudx1_custom_getall(void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        auto res=preset_loader->GetAll();
        fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
        for (size_t n=0;n<res.size();n++) {
            all[n]=fs_strdup(res[n].c_str());
        }
        all[res.size()]=nullptr;
        return all;
    }
    static void cloudx1_custom_getall_free(fs_char** all, void* userdata) {
        if (!all) return;
        for (fs_char** one=all;*one;one++) {
            free(*one);
        }
        free(all);
    }
    static int cloudx1_custom_remove(const fs_char *filename, void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        auto res=preset_loader->Remove(filename);
        return res?0:-1;
    }
    static int cloudx1_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
        sound4::cloudx1::dyn::CPresetLoader *preset_loader=reinterpret_cast<sound4::cloudx1::dyn::CPresetLoader*>(userdata);
        auto res=preset_loader->Rename(from,to);
        return res?0:-1;
    }

};
```

11.13 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]

11.13.1 Function Documentation

11.13.1.1 GetEnv()

```
std::string GetEnv (
            const std::string & name )
```

Definition at line 60 of file sound4cl-pipe.cpp.

11.13.1.2 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 178 of file sound4cl-pipe.cpp.

11.13.1.3 ProcessLoop()

```
void ProcessLoop (
            sound4::CInstance & instance,
            sound4::CProcessor & processor )
```

[Processing Loop]

[Processing Loop]

Definition at line 69 of file sound4cl-pipe.cpp.

11.14 sound4cl-pipe.cpp

```
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 // ------------------------------------------------------------
00060 std::string GetEnv(const std::string& name)
00061 {
00062     char *data=getenv(name.c_str());
00063     if (!data) return {};
00064     return std::string(data);
00065 }
00066
00067
00068 // Reads from stdin, write to stdout
00069 void ProcessLoop(sound4::CInstance &instance, sound4::CProcessor &processor)
00070 {
00071 bool stopthread=false;
00072
00073     #if defined(_WIN32) || defined(WIN32)
00074         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00075         _setmode(_fileno(stdin), _O_BINARY);
00076         _setmode(_fileno(stdout), _O_BINARY);
00077     #endif
00078
00080     size_t input_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
00081     size_t output_sample_count = instance.GetChunkFrames() * processor.GetChannelCount();
00082     while (!stopthread)
00083     {
00084         // Read a chunk from stdin
00085         if (std::fread(instance.GetBufferIn(), sizeof(float), input_sample_count, stdin) !=
      input_sample_count) {
00086             std::cerr « "ProcessLoop: input stopped"« std::endl;
00087             stopthread=true;
00088             break;
00089         }
00090         if (stopthread) break;
00091         // Process it
00092         instance.ProcessAudio();
00093         // Write it to stdout
00094         if (fwrite(instance.GetBufferOut(), sizeof(float), output_sample_count, stdout) !=
      output_sample_count) {
00095             std::cerr « "Failed to write audio output" « std::endl;
00096             stopthread=true;
00097         }
00098     }
00100 }
00101
00103 class MyPresetManager : public sound4::CPresetLoader {
00104 public:
00105     MyPresetManager() = default;
00106     virtual ~MyPresetManager() = default;
00107
```

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

int main(int argc, const char **argv)
{
    int listen_port = 8080;
    auto store_path_str=GetEnv("S4STORE_PATH");
    std::string libname =  "libsound4.x1.cloud.so";


    // Get credentials from environment
    auto LoginKey=GetEnv("S4LOGINKEY");
    if (LoginKey.empty()) {
        LoginKey=GetEnv("S4USERID");
    }
    auto RadioName=GetEnv("RADIO_NAME");
    auto KeyId=GetEnv("ACCESS_KEY_ID");
    auto AccessKey=GetEnv("ACCESS_KEY_SECRET");

    const char *progname=argv[0];
```

```
    for (int n=1;n<argc;n++) {
        std::string arg = argv[n];
        if (arg=="-p" || arg=="--port") {
            if (argc<n+1) {
                std::cerr « "--port needs a number"« std::endl;
                return 1;
            }
            n++;
            listen_port=atoi(argv[n]);
        } else if (arg=="-l" || arg=="--lib") {
            if (argc<n+1) {
                std::cerr « "--port needs a number"« std::endl;
                return 1;
            }
            n++;
            libname=argv[n];
        } else if (arg=="-h" || arg=="--help") {
            std::cerr « "Usage: " « progname « " [-p port] [-l libname.so]"«std::endl;;
            std::cerr « "  Reads audio from stdin and return processed audio in stdout" « std::endl;
            std::cerr « "   -l, --lib   : library to load ["«libname«"]" « std::endl;
            std::cerr « "   -p, --port  : port for web server ["«listen_port«"]" « std::endl;
            std::cerr « std::endl;
            std::cerr « "Environment:"« std::endl;;
            std::cerr « " - S4STORE_PATH            : path to store config [default: (no storage)]"« std::endl;;
            std::cerr « " - S4LOGINKEY              : SOUND4 Login key (or S4USERID)"« std::endl;;
            std::cerr « " - RADIO_NAME              : SOUND4 Radio Name"« std::endl;;
            std::cerr « " - ACCESS_KEY_ID           : SOUND4 Access Key ID"« std::endl;;
            std::cerr « " - ACCESS_KEY_SECRET       : SOUND4 Access Key Secret"« std::endl;
            std::cerr « std::endl;
            std::cerr « "Example:"« std::endl;
            std::cerr « "  ffmpeg -re -i file.wav -ar 48000 -ac 2 -f f32le - | "«progname«" | ffplay -f f32le -ar 48000 -ac 2 -i -" « std::endl;;
            #if defined(_WIN32) || defined(WIN32)
            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;;
            #else
            std::cerr « "  ffmpeg -f pulse -i default -ar 48000 -ac 2 -f f32le - | "«progname«" | ffplay -f f32le -ar 48000 -ac 2 -i -" « std::endl;;
            #endif
            return 0;
        } else {
            std::cerr « "unknown argument '"«arg«"'"« std::endl;
            return 1;
        }
    }


    sound4::CProcessor processor;
    if (!processor.Load(libname)) {
        std::cerr « "Failed to load processing library" « std::endl;
        return 1;
    }

    if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
        std::cerr « "Missing credentials in environment" « std::endl;
        return 1;
    }
    std::filesystem::path store_path;
    if (!store_path_str.empty()) {
        store_path = std::filesystem::path(store_path_str);
        if (!std::filesystem::exists(store_path) || !std::filesystem::is_directory(store_path)) {
            std::cerr « "Storage folder '"«store_path.c_str()«"' is not a valid directory" « std::endl;
            return 1;
        }
    }

    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");

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

```

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

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

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

    ProcessLoop(instance, processor);

    std::cerr « "Stopping now" « std::endl;
    instance.Stop();

    std::cerr « "End" « std::endl;
    return 0;
}
```

11.15 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.
- struct sound4::helper::SampleFormat< int16_t >
- struct sound4::helper::SampleFormat< int32_t >
- struct sound4::helper::SampleFormat< float >
- class sound4::helper::DynFuncHolder< T >

  Helper to wrap a function pointer.
- class sound4::helper::CDynLoader

  Helper to load dynamic library.
- class sound4::CProcessor

  Dynamic library interface.
- class sound4::CBus

  Shared Bus.
- class sound4::CPresetLoader

  Custom preset handler helper.
- class sound4::CInstance

  Instance handling class.
- class sound4::CInstance::CClient

Namespaces

- sound4
- sound4::helper

Typedefs

- using sound4::helper::DynLib_t = HMODULE
- using sound4::helper::DynFunc_t = FARPROC
- using sound4::LogSeverity = sound4cl_LogSeverity
- using sound4::log_cb_t = std::function< void(LogSeverity, const std::string &)>
- using sound4::PresetChange_Kind = sound4cl_PresetChange_Kind

Variables

- const LogSeverity sound4::none = (LogSeverity) sound4cl_none
- const LogSeverity sound4::fatal = (LogSeverity) sound4cl_fatal
- const LogSeverity sound4::error = (LogSeverity) sound4cl_error
- const LogSeverity sound4::warning = (LogSeverity) sound4cl_warning
- const LogSeverity sound4::info = (LogSeverity) sound4cl_info
- const LogSeverity sound4::verbose = (LogSeverity) sound4cl_verbose
- const LogSeverity sound4::verbose2 = (LogSeverity) sound4cl_verbose2
- const LogSeverity sound4::verbose3 = (LogSeverity) sound4cl_verbose3
- const LogSeverity sound4::verbose4 = (LogSeverity) sound4cl_verbose4
- const LogSeverity sound4::verbose5 = (LogSeverity) sound4cl_verbose5

11.15.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file sound4cl.hpp.

11.16 sound4cl.hpp

```
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 #ifdef __unix__
00022     #include <dlfcn.h>
00023 #elif defined(_WIN32)
00024     #include <libloaderapi.h>
00025 #else
00026     #error "Unsupported OS"
00027 #endif
00028
00029 #include "sound4cl_cdef.h"
00030
00031     // bridge C callbacks to CPresetLoader
00032     extern "C" {
```

```
        static char *sound4cl_custom_reader(const fs_char *filename, void* userdata);
        static void sound4cl_custom_reader_free(char *content, void* userdata);
        static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata);
        static int sound4cl_custom_exists(const fs_char *filename, void* userdata);
        static fs_char** sound4cl_custom_getall(void* userdata);
        static void sound4cl_custom_getall_free(fs_char** all, void* userdata);
        static int sound4cl_custom_remove(const fs_char *filename, void* userdata);
        static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
    };

namespace sound4 {

    namespace helper {
        #ifdef _WIN32
        static inline std::string WStringToUTF8(const std::wstring& wstr) {
            if (wstr.empty()) return {};
            int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
            std::string str(size, 0);
            WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
            return str;
        }
        static inline std::wstring UTF8ToWString(const std::string& str) {
            if (str.empty()) return std::wstring();
            int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
            std::wstring wstr(size, 0);
            MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
            return wstr;
        }
        #endif // _WIN32

            template<typename T>
        struct SampleFormat {
        };
            template<>
        struct SampleFormat<int16_t> {
            const sound4cl_SampleFormat format = S16_NATIVE;
        };
            template<>
        struct SampleFormat<int32_t> {
            const sound4cl_SampleFormat format = S32_NATIVE;
        };
            template<>
        struct SampleFormat<float> {
            const sound4cl_SampleFormat format = F32_NATIVE;
        };

        // Storage definition, depending on OS
        #ifdef __unix__
            using DynLib_t = void *;
            using DynFunc_t = void *;
        #else
            using DynLib_t = HMODULE;
            using DynFunc_t = FARPROC;
        #endif

            template <typename T>
        class DynFuncHolder {
        private:
            DynFunc_t m_ptr = NULL;
        public:
            DynFuncHolder() = default;
            DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
            operator bool() const { return (m_ptr!=NULL); }
            bool IsOk() const { return m_ptr!=NULL; }
            operator T* () const { return reinterpret_cast<T*>(m_ptr); }
        };

        class CDynLoader {
            private:
                DynLib_t m_lib {};
            public:
                CDynLoader() = default;
                CDynLoader(CDynLoader&&) = default;
                // No copy allowed
                CDynLoader(const CDynLoader&) = delete;
                CDynLoader& operator=(CDynLoader const&) = delete;
                operator bool() const {
                    return (m_lib!=NULL);
                }
                bool IsOk() const {
                    return (m_lib!=NULL);
                }
                #ifdef __unix__
                    void Close() {
                        if (IsOk()) {
```

```
                        #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
                        dlclose(m_lib);
                        #endif // __SANITIZE_ADDRESS__
                        m_lib=NULL;
                    }
                }
                bool Load(const std::filesystem::path& dynlib)
                {
                    Close();
                    m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
                    return IsOk();
                }
                    template <typename T>
                DynFuncHolder<T> GetSymbol(const std::string& name) {
                    auto ptr=dlsym(m_lib, name.c_str());
                    if (!ptr) {
                        throw std::runtime_error("Missing function in library");
                    }
                    return DynFuncHolder<T>(ptr);
                }
                    template <typename T>
                DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
                    auto ptr=dlsym(m_lib, name.c_str());
                    return DynFuncHolder<T>(ptr);
                }
                static std::filesystem::path GetThisLibraryPath(void) {
                    static Dl_info info;
                    if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
                    auto rp=realpath(info.dli_fname, NULL);
                    std::filesystem::path p(rp);
                    free(rp);
                    return p;
                }
            #else
                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_
                {
                    Close();
                    m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
                    return IsOk();
                }
                void Close() {
                    if (IsOk()) {
                        // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
FreeLibraryAndExitThread but...
                        // So define SOUND4_CALL_FREELIBRARY before including this file if you
really want to unload the library dynamically
                        #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
                            FreeLibraryAndExitThread(m_lib, 0);
                        #elif defined(SOUND4_CALL_FREELIBRARY)
                            FreeLibrary(m_lib);
                        #endif // SOUND4_CALL_FREELIBRARY
                        m_lib=NULL;
                    }
                }
                    template <typename T>
                DynFuncHolder<T> GetSymbol(const std::string& name) {
                    auto ptr=GetProcAddress(m_lib, name.c_str());
                    if (!ptr) {
                        throw std::runtime_error("Missing function in library");
                    }
                    return DynFuncHolder<T>(ptr);
                }
                    template <typename T>
                DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
                    auto ptr=GetProcAddress(m_lib, name.c_str());
                    return DynFuncHolder<T>(ptr);
                }
                static std::filesystem::path GetThisLibraryPath(void) {
                    static wchar_t path[MAX_PATH]={};
                    HMODULE hm = NULL;
                    if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
                            GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
                            (LPCWSTR) &GetThisLibraryPath, &hm))
                    {
                        if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
                            return std::filesystem::path(path);
                        }
                    }
                    return {};
                }
            #endif
            virtual ~CDynLoader() {
                Close();
            }
        }; // class CDynLoader
    }; // namespace helper
```

```
    using LogSeverity = sound4cl_LogSeverity;
    const LogSeverity none     = (LogSeverity) sound4cl_none;
    const LogSeverity fatal    = (LogSeverity) sound4cl_fatal;
    const LogSeverity error    = (LogSeverity) sound4cl_error;
    const LogSeverity warning  = (LogSeverity) sound4cl_warning;
    const LogSeverity info     = (LogSeverity) sound4cl_info;
    const LogSeverity verbose  = (LogSeverity) sound4cl_verbose;
    const LogSeverity verbose2 = (LogSeverity) sound4cl_verbose2;
    const LogSeverity verbose3 = (LogSeverity) sound4cl_verbose3;
    const LogSeverity verbose4 = (LogSeverity) sound4cl_verbose4;
    const LogSeverity verbose5 = (LogSeverity) sound4cl_verbose5;

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

    static log_cb_t _log_cb;
    extern "C" {
        static inline void _log_cb_c(LogSeverity severity, const char *c_msg) {
            _log_cb(severity, std::string(c_msg));
        }
    }

    class CProcessor {
    private:
        helper::CDynLoader m_lib;
        bool m_bOK = false;
        std::string m_prefix; // The prefix of functions, depends on the library (cloud_, impact_...)
    protected:
        friend class CBus;
        friend class CInstance;

        // All those wrapped functions have the same signature as there C sound4cl_XXX
        helper::DynFuncHolder< decltype(sound4cl_GetVersion              ) > fnGetVersion              ;
        helper::DynFuncHolder< decltype(sound4cl_GetChunkSizeInFrames    ) > fnGetChunkSizeInFrames    ;
        helper::DynFuncHolder< decltype(sound4cl_GetChannelCount         ) > fnGetChannelCount         ;
        helper::DynFuncHolder< decltype(sound4cl_GetAudioInputCount      ) > fnGetAudioInputCount      ;
        helper::DynFuncHolder< decltype(sound4cl_GetAudioOutputCount     ) > fnGetAudioOutputCount     ;
        helper::DynFuncHolder< decltype(sound4cl_GetSampleRate           ) > fnGetSampleRate           ;
        helper::DynFuncHolder< decltype(sound4cl_SetLoggerCallback       ) > fnSetLoggerCallback       ;
        helper::DynFuncHolder< decltype(sound4cl_SetLogSeverity          ) > fnSetLogSeverity          ;
        helper::DynFuncHolder< decltype(sound4cl_NewParameters           ) > fnNewParameters           ;
        helper::DynFuncHolder< decltype(sound4cl_FreeParameters          ) > fnFreeParameters          ;
        helper::DynFuncHolder< decltype(sound4cl_SetParameter            ) > fnSetParameter            ;
        helper::DynFuncHolder< decltype(sound4cl_GetParameter            ) > fnGetParameter            ;
        helper::DynFuncHolder< decltype(sound4cl_FreeParameterValue      ) > fnFreeParameterValue      ;
        helper::DynFuncHolder< decltype(sound4cl_InitProcess             ) > fnInitProcess             ;
        helper::DynFuncHolder< decltype(sound4cl_InitProcess2            ) > fnInitProcess2            ;
        helper::DynFuncHolder< decltype(sound4cl_TerminateProcess        ) > fnTerminateProcess        ;
        helper::DynFuncHolder< decltype(sound4cl_ExitProcess             ) > fnExitProcess             ;
        helper::DynFuncHolder< decltype(sound4cl_StartUpdateThread       ) > fnStartUpdateThread       ;
        helper::DynFuncHolder< decltype(sound4cl_StopUpdateThread        ) > fnStopUpdateThread        ;
        helper::DynFuncHolder< decltype(sound4cl_WaitUpdateThreadReady   ) > fnWaitUpdateThreadReady   ;
        helper::DynFuncHolder< decltype(sound4cl_ProcessAudio            ) > fnProcessAudio            ;
        helper::DynFuncHolder< decltype(sound4cl_ProcessAudio_Planar     ) > fnProcessAudio_Planar     ;
        helper::DynFuncHolder< decltype(sound4cl_GetBufferIn             ) > fnGetBufferIn             ;
        helper::DynFuncHolder< decltype(sound4cl_GetBufferOut            ) > fnGetBufferOut            ;
        helper::DynFuncHolder< decltype(sound4cl_GetEstimatedDelay       ) > fnGetEstimatedDelay       ;
        helper::DynFuncHolder< decltype(sound4cl_GetFormatName           ) > fnGetFormatName           ;
        helper::DynFuncHolder< decltype(sound4cl_GetFormatFromName       ) > fnGetFormatFromName       ;
        helper::DynFuncHolder< decltype(sound4cl_GetBytesFromFormat      ) > fnGetBytesFromFormat      ;
        helper::DynFuncHolder< decltype(sound4cl_GetMaxPacketFrame       ) > fnGetMaxPacketFrame       ;
        helper::DynFuncHolder< decltype(sound4cl_AddAudio                ) > fnAddAudio                ;
        helper::DynFuncHolder< decltype(sound4cl_AddPadAudio             ) > fnAddPadAudio             ;
        helper::DynFuncHolder< decltype(sound4cl_GetOutputCount          ) > fnGetOutputCount          ;
        helper::DynFuncHolder< decltype(sound4cl_GetAudio                ) > fnGetAudio                ;
        helper::DynFuncHolder< decltype(sound4cl_AudioConvertFrom        ) > fnAudioConvertFrom        ;
        helper::DynFuncHolder< decltype(sound4cl_AudioConvertTo          ) > fnAudioConvertTo          ;
        helper::DynFuncHolder< decltype(sound4cl_StereoToMono            ) > fnStereoToMono            ;
        helper::DynFuncHolder< decltype(sound4cl_MonoToStereo            ) > fnMonoToStereo            ;
        helper::DynFuncHolder< decltype(sound4cl_AudioMonoFromLiveStereo) > fnAudioMonoFromLiveStereo;
        helper::DynFuncHolder< decltype(sound4cl_AudioMonoToLiveStereo   ) > fnAudioMonoToLiveStereo   ;
        helper::DynFuncHolder< decltype(sound4cl_NewClient               ) > fnNewClient               ;
        helper::DynFuncHolder< decltype(sound4cl_DeleteClient            ) > fnDeleteClient            ;
        helper::DynFuncHolder< decltype(sound4cl_ProcessJson             ) > fnProcessJson             ;
        helper::DynFuncHolder< decltype(sound4cl_FreeJsonAnswer          ) > fnFreeJsonAnswer          ;
        helper::DynFuncHolder< decltype(sound4cl_SaveState               ) > fnSaveState               ;

        // only for processes supporting webserver
        helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp           ) > fnWebserver_tcp           ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver_tcp2          ) > fnWebserver_tcp2          ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver               ) > fnWebserver               ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver_Stop          ) > fnWebserver_Stop          ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver_Status        ) > fnWebserver_Status        ;

        // added 2023-03-22
        helper::DynFuncHolder< decltype(sound4cl_StereoToMono_Planar     ) > fnStereoToMono_Planar     ;
```

```
        helper::DynFuncHolder< decltype(sound4cl_MonoToStereo_Planar    ) > fnMonoToStereo_Planar    ;

        // added 2023-05-22
        helper::DynFuncHolder< decltype(sound4cl_SetMetadata            ) > fnSetMetadata            ;
        helper::DynFuncHolder< decltype(sound4cl_GetMetadataInfos       ) > fnGetMetadataInfos       ;
        helper::DynFuncHolder< decltype(sound4cl_FreeMetadataInfos      ) > fnFreeMetadataInfos      ;

        // added 2023-06-20
        helper::DynFuncHolder< decltype(sound4cl_SetPresetManager                ) > fnSetPresetManager      ;
        helper::DynFuncHolder< decltype(sound4cl_PresetManager_InformChange      ) > fnPresetManager_InformChange      ;

        // added 2023-09-21
        helper::DynFuncHolder< decltype(sound4cl_GetPossibleChunkSizeInFrames    ) > fnGetPossibleChunkSizeInFrames   ;
        helper::DynFuncHolder< decltype(sound4cl_GetProcessChunkFrames           ) > fnGetProcessChunkFrames   ;
        helper::DynFuncHolder< decltype(sound4cl_InitProcess3                    ) > fnInitProcess3 ;

        // added 2023-06-26, only for processes supporting bus
        helper::DynFuncHolder< decltype(sound4cl_NewBus                  ) > fnNewBus      ;
        helper::DynFuncHolder< decltype(sound4cl_FreeBus                 ) > fnFreeBus      ;
        helper::DynFuncHolder< decltype(sound4cl_SetInstanceBus          ) > fnSetInstanceBus      ;

        // added 2024-02-19
        helper::DynFuncHolder< decltype(sound4cl_SetMetadataMulti        ) > fnSetMetadataMulti ;

        // added 2024-05-23
        helper::DynFuncHolder< decltype(sound4cl_Webserver_SetAppHealth    ) > fnWebserver_SetAppHealth      ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver_GetAppHealth    ) > fnWebserver_GetAppHealth      ;
        helper::DynFuncHolder< decltype(sound4cl_Webserver_FreeString      ) > fnWebserver_FreeString ;

            template <typename T>
        helper::DynFuncHolder<T> GetPrefixSymbol(const std::string& name) {
            std::string prefix_name = m_prefix+name;
            return m_lib.GetSymbol<T>(prefix_name);
        }

    public:
        CProcessor() = default;

        bool Load(const std::filesystem::path& filepath) {
            m_lib.Close();
            // If a path is given, use it directly and do not try other path
            if (filepath.has_parent_path() && !m_lib.Load(filepath)) {
                return false;
            } else if (!m_lib.IsOk()) {
                auto thisdir=helper::CDynLoader::GetThisLibraryPath().parent_path();
                auto filename = filepath.filename();
                #ifndef _WIN32
                // Linux: if in a bin directory, try ../lib first
                if (!m_lib.IsOk() && thisdir.filename()=="bin") {
                    auto libdir = thisdir.parent_path() / "lib";
                    m_lib.Load(libdir / filename);
                }
                #endif // !_WIN32
                if (!m_lib.IsOk()) {
                    // Search in the same directory this code is in
                    m_lib.Load(thisdir / filename);
                }
                if (!m_lib.IsOk()) {
                    // Try current path
                    m_lib.Load(std::filesystem::current_path() / filename);
                }
                if (!m_lib.IsOk()) {
                    return false;
                }
            }
            // Get information from the library itself
            try {
                auto GetInfo = m_lib.GetSymbol< decltype(sound4cl_SOUND4_GetProcessInfo             ) >("SOUND4_GetProcessInfo"   );
                auto info = GetInfo();
                m_prefix = info->prefix;
            } catch (std::runtime_error& ) {
                // Compatibility: search in known libraries without this function
                #ifdef _WIN32
                    std::string filename = helper::WStringToUTF8(filepath.stem());
                #else // !_WIN32
                    std::string filename = filepath.stem();
                    // remove lib prefix
```

```
                    filename=filename.substr(3);
                #endif // ! _WIN32
                if (filename=="sound4.x1.cloud") {
                    m_prefix="cloudx1_";
                } else if (filename=="sound4.impact.cl") {
                    m_prefix="impact_";
                } else if (filename=="sound4.bigvoice.cl") {
                    m_prefix="bigvoice_";
                } else {
                    return false;
                }
            }

            // Load all C functions
            try {
                fnGetVersion              = GetPrefixSymbol< decltype(sound4cl_GetVersion
    ) >("GetVersion"             );
                fnGetChunkSizeInFrames    = GetPrefixSymbol< decltype(sound4cl_GetChunkSizeInFrames
    ) >("GetChunkSizeInFrames"   );
                fnGetChannelCount         = GetPrefixSymbol< decltype(sound4cl_GetChannelCount
    ) >("GetChannelCount"        );
                fnGetAudioInputCount      = GetPrefixSymbol< decltype(sound4cl_GetAudioInputCount
    ) >("GetAudioInputCount"     );
                fnGetAudioOutputCount     = GetPrefixSymbol< decltype(sound4cl_GetAudioOutputCount
    ) >("GetAudioOutputCount"    );
                fnGetSampleRate           = GetPrefixSymbol< decltype(sound4cl_GetSampleRate
    ) >("GetSampleRate"          );
                fnSetLoggerCallback       = GetPrefixSymbol< decltype(sound4cl_SetLoggerCallback
    ) >("SetLoggerCallback"      );
                fnSetLogSeverity          = GetPrefixSymbol< decltype(sound4cl_SetLogSeverity
    ) >("SetLogSeverity"         );
                fnNewParameters           = GetPrefixSymbol< decltype(sound4cl_NewParameters
    ) >("NewParameters"          );
                fnFreeParameters          = GetPrefixSymbol< decltype(sound4cl_FreeParameters
    ) >("FreeParameters"         );
                fnSetParameter            = GetPrefixSymbol< decltype(sound4cl_SetParameter
    ) >("SetParameter"           );
                fnGetParameter            = GetPrefixSymbol< decltype(sound4cl_GetParameter
    ) >("GetParameter"           );
                fnFreeParameterValue      = GetPrefixSymbol< decltype(sound4cl_FreeParameterValue
    ) >("FreeParameterValue"     );
                fnInitProcess             = GetPrefixSymbol< decltype(sound4cl_InitProcess
    ) >("InitProcess"            );
                fnInitProcess2            = GetPrefixSymbol< decltype(sound4cl_InitProcess2
    ) >("InitProcess2"           );
                fnTerminateProcess        = GetPrefixSymbol< decltype(sound4cl_TerminateProcess
    ) >("TerminateProcess"       );
                fnExitProcess             = GetPrefixSymbol< decltype(sound4cl_ExitProcess
    ) >("ExitProcess"            );
                fnStartUpdateThread       = GetPrefixSymbol< decltype(sound4cl_StartUpdateThread
    ) >("StartUpdateThread"      );
                fnStopUpdateThread        = GetPrefixSymbol< decltype(sound4cl_StopUpdateThread
    ) >("StopUpdateThread"       );
                fnWaitUpdateThreadReady   = GetPrefixSymbol< decltype(sound4cl_WaitUpdateThreadReady
    ) >("WaitUpdateThreadReady"  );
                fnProcessAudio            = GetPrefixSymbol< decltype(sound4cl_ProcessAudio
    ) >("ProcessAudio"           );
                fnProcessAudio_Planar     = GetPrefixSymbol< decltype(sound4cl_ProcessAudio_Planar
    ) >("ProcessAudio_Planar"    );
                fnGetBufferIn             = GetPrefixSymbol< decltype(sound4cl_GetBufferIn
    ) >("GetBufferIn"            );
                fnGetBufferOut            = GetPrefixSymbol< decltype(sound4cl_GetBufferOut
    ) >("GetBufferOut"           );
                fnGetEstimatedDelay       = GetPrefixSymbol< decltype(sound4cl_GetEstimatedDelay
    ) >("GetEstimatedDelay"      );
                fnGetFormatName           = GetPrefixSymbol< decltype(sound4cl_GetFormatName
    ) >("GetFormatName"          );
                fnGetFormatFromName       = GetPrefixSymbol< decltype(sound4cl_GetFormatFromName
    ) >("GetFormatFromName"      );
                fnGetBytesFromFormat      = GetPrefixSymbol< decltype(sound4cl_GetBytesFromFormat
    ) >("GetBytesFromFormat"     );
                fnGetMaxPacketFrame       = GetPrefixSymbol< decltype(sound4cl_GetMaxPacketFrame
    ) >("GetMaxPacketFrame"      );
                fnAddAudio                = GetPrefixSymbol< decltype(sound4cl_AddAudio
    ) >("AddAudio"               );
                fnAddPadAudio             = GetPrefixSymbol< decltype(sound4cl_AddPadAudio
    ) >("AddPadAudio"            );
                fnGetOutputCount          = GetPrefixSymbol< decltype(sound4cl_GetOutputCount
    ) >("GetOutputCount"         );
                fnGetAudio                = GetPrefixSymbol< decltype(sound4cl_GetAudio
    ) >("GetAudio"               );
                fnAudioConvertFrom        = GetPrefixSymbol< decltype(sound4cl_AudioConvertFrom
    ) >("AudioConvertFrom"       );
                fnAudioConvertTo          = GetPrefixSymbol< decltype(sound4cl_AudioConvertTo
    ) >("AudioConvertTo"         );
                fnStereoToMono            = GetPrefixSymbol< decltype(sound4cl_StereoToMono
    ) >("StereoToMono"           );
```

```
                fnMonoToStereo            = GetPrefixSymbol< decltype(sound4cl_MonoToStereo ) >("MonoToStereo"           );
                fnAudioMonoFromLiveStereo = GetPrefixSymbol< decltype(sound4cl_AudioMonoFromLiveStereo) >("AudioMonoFromLiveStereo");
                fnAudioMonoToLiveStereo   = GetPrefixSymbol< decltype(sound4cl_AudioMonoToLiveStereo ) >("AudioMonoToLiveStereo"  );
                fnNewClient               = GetPrefixSymbol< decltype(sound4cl_NewClient ) >("NewClient"              );
                fnDeleteClient            = GetPrefixSymbol< decltype(sound4cl_DeleteClient ) >("DeleteClient"           );
                fnProcessJson             = GetPrefixSymbol< decltype(sound4cl_ProcessJson ) >("ProcessJson"            );
                fnFreeJsonAnswer          = GetPrefixSymbol< decltype(sound4cl_FreeJsonAnswer ) >("FreeJsonAnswer"         );
                fnSaveState               = GetPrefixSymbol< decltype(sound4cl_SaveState ) >("SaveState"              );
            } catch (std::runtime_error& ) {
                return false;
            }
            // Webserver : not for all projects
            try {
                fnWebserver_tcp           = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp ) >("Webserver_tcp"          );
                fnWebserver_tcp2          = GetPrefixSymbol< decltype(sound4cl_Webserver_tcp2 ) >("Webserver_tcp2"         );
                fnWebserver               = GetPrefixSymbol< decltype(sound4cl_Webserver ) >("Webserver"              );
                fnWebserver_Stop          = GetPrefixSymbol< decltype(sound4cl_Webserver_Stop ) >("Webserver_Stop"         );
                fnWebserver_Status        = GetPrefixSymbol< decltype(sound4cl_Webserver_Status ) >("Webserver_Status"       );
            } catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }
            // Stereo/Mono conversion : must be present
            try {
                // added 2023-03-22
                fnStereoToMono_Planar     = GetPrefixSymbol< decltype(sound4cl_StereoToMono_Planar ) >("StereoToMono_Planar"    );
                fnMonoToStereo_Planar     = GetPrefixSymbol< decltype(sound4cl_MonoToStereo_Planar ) >("MonoToStereo_Planar"    );

            } catch (std::runtime_error& ) {
                return false;
            }
            // C functions allowed to be missing
            try {
                // added 2023-05-22
                fnSetMetadata             = GetPrefixSymbol< decltype(sound4cl_SetMetadata ) >("SetMetadata"            );
                fnGetMetadataInfos        = GetPrefixSymbol< decltype(sound4cl_GetMetadataInfos ) >("GetMetadataInfos"       );
                fnFreeMetadataInfos       = GetPrefixSymbol< decltype(sound4cl_FreeMetadataInfos ) >("FreeMetadataInfos"      );
                // added 2023-06-20
                fnSetPresetManager            = GetPrefixSymbol< decltype(sound4cl_SetPresetManager ) >("SetPresetManager"             );
                fnPresetManager_InformChange  = GetPrefixSymbol< decltype(sound4cl_PresetManager_InformChange     ) >("PresetManager_InformChange"    );

                // added 2023-09-21
                fnGetPossibleChunkSizeInFrames = GetPrefixSymbol< decltype(sound4cl_GetPossibleChunkSizeInFrames  ) >("GetPossibleChunkSizeInFrames"  );
                fnGetProcessChunkFrames        = GetPrefixSymbol< decltype(sound4cl_GetProcessChunkFrames         ) >("GetProcessChunkFrames"         );
                fnInitProcess3                 = GetPrefixSymbol< decltype(sound4cl_InitProcess3 ) >("InitProcess3"                 );
            }  catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }

            // bus: not for all projects
            try {
                // added 2023-06-26
                fnNewBus                  = GetPrefixSymbol< decltype(sound4cl_NewBus ) >("NewBus"                 );
                fnFreeBus                 = GetPrefixSymbol< decltype(sound4cl_FreeBus ) >("FreeBus"                 );
                fnSetInstanceBus          = GetPrefixSymbol< decltype(sound4cl_SetInstanceBus ) >("SetInstanceBus"         );
            }  catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }

            // Other optional
            try {
                // added 2024-02-19
```

```
                fnSetMetadataMulti        = GetPrefixSymbol< decltype(sound4cl_SetMetadataMulti ) >("SetMetadataMulti"       );

            }  catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }

            // Other optional, web server only
            try {
                // added 2024-05-23
                fnWebserver_SetAppHealth    = GetPrefixSymbol< decltype(sound4cl_Webserver_SetAppHealth    ) >("Webserver_SetAppHealth"     );
                fnWebserver_GetAppHealth    = GetPrefixSymbol< decltype(sound4cl_Webserver_GetAppHealth    ) >("Webserver_GetAppHealth"     );
                fnWebserver_FreeString      = GetPrefixSymbol< decltype(sound4cl_Webserver_FreeString      ) >("Webserver_FreeString"       );
            }  catch (std::runtime_error& ) {
                // Ignored, handler will take care
            }

            m_bOK=true;
            return true;
        }
        bool IsOk() const {
            return m_bOK;
        }

        void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, sound4cl_SampleFormat fmt)
            { fnAudioConvertFrom(payload, output, nSpl, fmt); }
        void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, sound4cl_SampleFormat fmt)
            { fnAudioConvertTo(input, payload, nSpl, fmt); }
        void StereoToMono(const float *input, float *output, size_t nSpl)
            { fnStereoToMono(input, output, nSpl); }
        void MonoToStereo(const float *input, float *output, size_t nSpl)
            { fnMonoToStereo(input, output, nSpl); }
        void StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t nSpl)
            { fnStereoToMono_Planar(inputL, inputR, output, nSpl); }
        void MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nSpl)
            { fnMonoToStereo_Planar(input, outputL, outputR, nSpl); }
        void AudioMonoFromLiveStereo(const uint8_t *payload, float *output)
            { fnAudioMonoFromLiveStereo(payload, output); }
        void AudioMonoToLiveStereo(const float *input, uint8_t *payload)
            { fnAudioMonoToLiveStereo(input, payload); }

        std::string GetVersion()
            { return std::string(fnGetVersion()); }

        unsigned int GetChunkSizeInFrames()
            { return fnGetChunkSizeInFrames(); }

        std::vector<unsigned int> GetPossibleChunkSizeInFrames()
            {
                std::vector<unsigned int> list;
                if (fnGetPossibleChunkSizeInFrames) {
                    for (unsigned int* src=fnGetPossibleChunkSizeInFrames();*src;src++) {
                        list.push_back(*src);
                    }
                } else {
                    list.push_back(fnGetChunkSizeInFrames());
                }
                return list;
            }

        unsigned int GetChannelCount()
            { return fnGetChannelCount(); }

        unsigned int GetAudioInputCount()
            { return fnGetAudioInputCount(); }

        unsigned int GetAudioOutputCount()
            { return fnGetAudioOutputCount(); }

        unsigned int GetSampleRate()
            { return fnGetSampleRate(); }

        std::string GetFormatName( const sound4cl_SampleFormat fmt)
            { return std::string(fnGetFormatName(fmt)); }

        sound4cl_SampleFormat GetFormatFromName( const std::string& name)
            { return fnGetFormatFromName(name.c_str()); }

        unsigned int GetBytesFromFormat( const sound4cl_SampleFormat fmt)
            { return fnGetBytesFromFormat(fmt); }

        void SetLogSeverity(LogSeverity severity)
```

```
            { fnSetLogSeverity(severity); }

        void SetLoggerCallback(log_cb_t cb)
            { _log_cb=cb; fnSetLoggerCallback(&_log_cb_c); }
    };

    class CBus {
        public:
            CBus(CProcessor& dynlib)
                : m_dynlib(dynlib)
                {
                    if (m_dynlib.fnNewBus) {
                        m_bus=m_dynlib.fnNewBus();
                    }
                }
            ~CBus()
                {
                    if (m_bus && m_dynlib.fnFreeBus) {
                        m_dynlib.fnFreeBus(m_bus);
                    }
                    m_bus = nullptr;
                }
            sound4cl_CBus *Get() const { return m_bus; }
        private:
            CProcessor& m_dynlib;
            sound4cl_CBus *m_bus;
    };

    class CPresetLoader {
    public:
        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;
    };
    using PresetChange_Kind = sound4cl_PresetChange_Kind;

    class CInstance {
    public:
        CInstance(CProcessor& dynlib)
        : m_dynlib(dynlib)
            {
                params = m_dynlib.fnNewParameters();
                if (!params) throw std::bad_alloc();
            }
        // non-copyable
        CInstance( const CInstance& ) = delete;
        CInstance& operator=( const CInstance& ) = delete;
        ~CInstance()
            {
                Stop();
                if (params) m_dynlib.fnFreeParameters(params);
            }
       bool IsOk() const { return instance!=nullptr; }
        void SetParam(const std::string& name, const std::string& value)
            {
                assert(params);
                m_dynlib.fnSetParameter(params, name.c_str(), value.c_str());
            }
        std::string GetParam(const std::string& name)
            {
                assert(params);
                auto c_value=m_dynlib.fnGetParameter(params, name.c_str());
                std::string ret(c_value);
                m_dynlib.fnFreeParameterValue(c_value);
                return ret;
            }

        void SetBus(const CBus& bus) {
            assert(params);
            if (m_dynlib.fnSetInstanceBus) {
                m_dynlib.fnSetInstanceBus(params,bus.Get());
            }
        }

        void SetPresetManager(CPresetLoader *preset_manager)
            {
                if (!preset_manager) return;
                if (!m_dynlib.fnSetPresetManager) return;
                assert(params);
                m_dynlib.fnSetPresetManager(params,
```

```
                    sound4cl_custom_reader,
                    sound4cl_custom_reader_free,
                    sound4cl_custom_writer,
                    sound4cl_custom_exists,
                    sound4cl_custom_getall,
                    sound4cl_custom_getall_free,
                    sound4cl_custom_remove,
                    sound4cl_custom_rename,
                    preset_manager->IsReadOnly(),
                    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)
            {
                assert(!instance);
                std::string l_save_path;
                if (!save_path.empty()) {
                    l_save_path = save_path.u8string();
                }
                if (frames_per_chunk==0) {
                    frames_per_chunk=m_dynlib.GetChunkSizeInFrames();
                }
                if (m_dynlib.fnInitProcess3) {
                    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);
                } else if (frames_per_chunk!=m_dynlib.GetChunkSizeInFrames()) {
                    return false;
                } else {
                    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);
                }
                if (!instance) return false;
                if (!init_metadata.empty()) {
                    for (auto&& [key,value]: init_metadata) {
                        m_dynlib.fnSetMetadata(instance, key.c_str(), value.c_str());
                    }
                    init_metadata.clear();
                }
                update_thread = std::thread([this, json_port](){
                    OnUpdateThreadStart();
                    m_dynlib.fnStartUpdateThread(instance, json_port);
                    OnUpdateThreadStop();
                });
                int timeout=5000;
                #ifdef DEBUG
                    timeout=60*1000; // For debugger pauses
                #endif
                if (m_dynlib.fnWaitUpdateThreadReady(instance, timeout)<0) {
                    return false;
                }
                return true;
            }

        unsigned int GetChunkFrames() {
            if (instance && m_dynlib.fnGetProcessChunkFrames) {
                return m_dynlib.fnGetProcessChunkFrames(instance);
            }
            return m_dynlib.fnGetChunkSizeInFrames();
        }

        void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind change_kind)
            {
                if (m_dynlib.fnPresetManager_InformChange) {
                    m_dynlib.fnPresetManager_InformChange(instance, relative_path.c_str(), (sound4cl_PresetChange_Kind) change_kind);
                }
            }

        void SetMetadata(const std::string& key, const char* value)
            {
                if (m_dynlib.fnSetMetadata) {
                    if (instance) {
                        m_dynlib.fnSetMetadata(instance, key.c_str(), value);
                    } else {
                        init_metadata.push_back( {key, value} );
                    }
                }
            }


        void SetMetadataMulti(const std::unordered_map<std::string, const char*>& list)
            {
                if (instance) {
```

```
                    if (m_dynlib.fnSetMetadataMulti) {
                        typedef const char* pchar;
                        pchar* keyvalue=new pchar[2*list.size()+1];
                        size_t n=0;
                        for (auto&& [key,value]: list) {
                            keyvalue[2*n+0]=key.c_str();
                            keyvalue[2*n+1]=value;
                            n++;
                        }
                        keyvalue[2*n+0]=nullptr;

                        sound4cl_SetMetadataMulti(instance, keyvalue);

                        delete[] keyvalue;
                    } else {
                        for (auto&& [key,value]: list) {
                            SetMetadata(key, value);
                        }
                    }
                } else {
                    for (auto&& [key,value]: list) {
                        if (value) {
                            init_metadata.push_back( {key, value} );
                        }
                    }
                }
            }

        std::vector< std::tuple<std::string,std::string> > GetMetadataInfos()
            {
                std::vector< std::tuple<std::string,std::string> > values;
                if (m_dynlib.fnGetMetadataInfos) {
                    const char** c_values = m_dynlib.fnGetMetadataInfos(instance);
                    if (c_values) {
                        for (const char** c_value=c_values; *c_value; ) {
                            std::string key(*c_value);
                            c_value++;
                            if (*c_value) {
                                std::string descr(*c_value);
                                c_value++;
                                values.push_back( {key,descr} );
                            }
                        }
                        m_dynlib.fnFreeMetadataInfos(instance, c_values);
                    }
                }
                return values;
            }

        // Can be overloaded, called at start of UpdateThread in the thread
        virtual void OnUpdateThreadStart() {}
        // Can be overloaded to call at end of UpdateThread in the thread
        virtual void OnUpdateThreadStop() {}

        bool StartWebServer(int http_port, int https_port=0)
            {
                assert(instance);
                assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
                if (!m_dynlib.fnWebserver) {
                    return false;
                }
                webserver = m_dynlib.fnWebserver(http_port,https_port,instance);
                if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
                    return false;
                }
                int web_status=m_dynlib.fnWebserver_Status(webserver);
                if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
                    return false;
                }
                if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
                    return false;
                }
                return true;
            }
        void StopWebServer()
            {
                if (!m_dynlib.fnWebserver_Stop) return;
                if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
                    m_dynlib.fnWebserver_Stop(webserver, 1000);
                    webserver = SOUND4_INVALID_WEBSERVER_ID;
                }
            }

        void SetWebServerAppHealth(int httpcode, const std::string& contenttype, const std::string&
    content)
            {
                assert(instance);
```

```
                if (!m_dynlib.fnWebserver_SetAppHealth) return;
                m_dynlib.fnWebserver_SetAppHealth(instance, httpcode, contenttype.c_str(), content.c_str());
            }
        void GetWebServerAppHealth(int &httpcode, std::string& contenttype, std::string& content)
            {
                assert(instance);
                if (!m_dynlib.fnWebserver_GetAppHealth) return;
                char* c_contenttype=nullptr;
                char* c_content=nullptr;
                m_dynlib.fnWebserver_GetAppHealth(instance, &httpcode, &c_contenttype, &c_content);
                contenttype=c_contenttype;
                content=c_content;
                m_dynlib.fnWebserver_FreeString(c_contenttype);
                m_dynlib.fnWebserver_FreeString(c_content);
            }
        int Terminate()
            {
                if (!instance) return 0;
                return m_dynlib.fnTerminateProcess(instance);
            }

        void Stop()
            {
                if (!instance) return;
                StopWebServer();
                if (update_thread.joinable()) {
                    m_dynlib.fnStopUpdateThread(instance); // this will exit the update thread
                    update_thread.join();
                }
                m_dynlib.fnExitProcess(instance);
                instance=nullptr;
            }

        unsigned int GetEstimatedDelay()
            {
                assert(instance);
                return m_dynlib.fnGetEstimatedDelay(instance);
            }

        float* GetBufferIn()
            {
                assert(instance);
                return m_dynlib.fnGetBufferIn(instance);

            }
        float* GetBufferOut()
            {
                assert(instance);
                return m_dynlib.fnGetBufferOut(instance);
            }
        void ProcessAudio()
            {
                assert(instance);
                m_dynlib.fnProcessAudio(instance, m_dynlib.fnGetBufferIn(instance), m_dynlib.fnGetBufferOut(instance));
            }
        // void ProcessAudio_Planar(const std::array< const std::array<const float,ChunkFrames>, InputChannels>& input, std::array< std::array<float,ChunkFrames>, OutputChannels>& output)
        void ProcessAudio_Planar(float const * const *input, float * const *output)
            {
                assert(instance);
                m_dynlib.fnProcessAudio_Planar(instance, input, output);
            }

            template<typename T>
        std::vector<T> ProcessAnyAudio(const std::vector<T> input)
            {
                assert(instance);
                std::vector<T> output;
                unsigned int out_offset=0;
                unsigned int in_offset=0;
                unsigned int todo = input.size();
                while (todo>0) {
                    unsigned int left = AddAudio(&input[in_offset], todo);
                    unsigned int out_avail = m_dynlib.fnGetOutputCount(instance);
                    output.resize(out_offset + out_avail);
                    GetAudio(&output[out_offset], out_avail);
                    out_offset+=out_avail;
                    in_offset += todo-left;
                    todo=left;
                }
                return output;
            }

        class CClient {
            public:
```

```
                CClient(sound4cl_CInstance* instance, CProcessor& dynlib)
                : m_dynlib(dynlib)
                    {
                        assert(instance);
                        client=m_dynlib.fnNewClient(instance);
                        if (!client) throw std::bad_alloc();
                    }
                // non-copyable
                CClient( const CInstance& ) = delete;
                CClient& operator=( const CInstance& ) = delete;
                ~CClient()
                    {
                        if (client) {
                            m_dynlib.fnDeleteClient(client);
                            client=nullptr;
                        }
                    }
                std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
                    {
                        assert(client);
                        int need_save=0;
                        const char *canswer = m_dynlib.fnProcessJson(client, request.c_str(), &need_save);
                        if (!canswer) return {};
                        std::string answer(canswer);
                        m_dynlib.fnFreeJsonAnswer (canswer);
                        if (NeedSave) {
                            *NeedSave=(need_save!=0);
                        }
                        return answer;
                    }
            private:
                CProcessor& m_dynlib;
                sound4cl_CClientInstance *client = nullptr;
        };
        std::shared_ptr<CClient> NewClient()
            {
                assert(instance);
                return std::make_shared<CClient>(instance, m_dynlib);
            }

    protected:
            template<typename T>
        unsigned int AddAudio(const T* payload, unsigned int nFrame)
            {
                assert(instance);
                return m_dynlib.fnAddAudio(instance, reinterpret_cast<const T*>(payload), nFrame, helper::SampleFormat<T>::format);
            }
            template<typename T>
        unsigned int GetAudio(T* payload, unsigned int max_nFrame)
            {
                assert(instance);
                return m_dynlib.fnGetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame, helper::SampleFormat<T>::format);
            }

    private:
        CProcessor& m_dynlib;
        sound4cl_CParameters* params = nullptr;
        sound4cl_CInstance* instance = nullptr;
        std::thread update_thread;
        std::vector< std::pair<std::string,std::string> > init_metadata;
        uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
    };

}; // namespace sound4


// bridge C callbacks to CPresetLoader
extern "C" {
    static char *sound4cl_custom_reader(const fs_char *filename, void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
        std::string content=preset_loader->Read(filename);
        return strdup(content.c_str());
    }
    static void sound4cl_custom_reader_free(char *content, void* userdata) {
        if (!content) return;
        free(content);
    }
    static int sound4cl_custom_writer(const fs_char *filename, const char *content, void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
        auto res=preset_loader->Write(filename,content);
        return res?0:-1;
    }
    static int sound4cl_custom_exists(const fs_char *filename, void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
```

```
        auto res=preset_loader->Exists(filename);
        return res?0:-1;
    }
    static fs_char** sound4cl_custom_getall(void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
        auto res=preset_loader->GetAll();
        fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
        for (size_t n=0;n<res.size();n++) {
            all[n]=fs_strdup(res[n].c_str());
        }
        all[res.size()]=nullptr;
        return all;
    }
    static void sound4cl_custom_getall_free(fs_char** all, void* userdata) {
        if (!all) return;
        for (fs_char** one=all;*one;one++) {
            free(*one);
        }
        free(all);
    }
    static int sound4cl_custom_remove(const fs_char *filename, void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
        auto res=preset_loader->Remove(filename);
        return res?0:-1;
    }
    static int sound4cl_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
        sound4::CPresetLoader *preset_loader=reinterpret_cast<sound4::CPresetLoader*>(userdata);
        auto res=preset_loader->Rename(from,to);
        return res?0:-1;
    }

};
```

11.17 sound4cl_cdef.h File Reference

C function signature definition for all sound4.cl libraries.

```
#include <stdint.h>
#include <stringapiset.h>
#include <wchar.h>
```

Classes

- struct sound4cl_SOUND4_ProcessInfo

Macros

- #define fs_strdup wcsdup

  helper for OS dependant file name copy
- #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 void(∗ sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char ∗)
- typedef char ∗(∗ sound4cl_storage_reader) (const fs_char ∗filename, void ∗userdata)
- typedef void(∗ sound4cl_storage_reader_free) (char ∗content, void ∗userdata)
- typedef int(∗ sound4cl_storage_writer) (const fs_char ∗filename, const char ∗content, void ∗userdata)
- typedef int(∗ sound4cl_storage_exists) (const fs_char ∗filename, void ∗userdata)
- typedef fs_char ∗∗(∗ sound4cl_storage_getall) (void ∗userdata)
- typedef void(∗ sound4cl_storage_getall_free) (fs_char ∗∗all, void ∗userdata)
- typedef int(∗ sound4cl_storage_remove) (const fs_char ∗filename, void ∗userdata)
- typedef int(∗ sound4cl_storage_rename) (const fs_char ∗from, const fs_char ∗to, void ∗userdata)

Enumerations

- enum sound4cl_LogSeverity {
  sound4cl_none = 0 , sound4cl_fatal = 1 , sound4cl_error = 2 , sound4cl_warning = 3 ,
  sound4cl_info = 4 , sound4cl_verbose = 5 , sound4cl_verbose2 = 6 , sound4cl_verbose3 = 7 ,
  sound4cl_verbose4 = 8 , sound4cl_verbose5 = 9 }
- enum sound4cl_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted }
- enum sound4cl_SampleFormat {
  INVALID_FORMAT =0 , S16_LE , S16_BE , S24_LE ,
  S24_BE , S32_LE , S32_BE , F32_LE ,
  F32_BE , S16_NATIVE =S16_LE , S24_NATIVE =S24_LE , S32_NATIVE =S32_LE ,
  F32_NATIVE =F32_LE }

Functions

- const struct sound4cl_SOUND4_ProcessInfo ∗ sound4cl_SOUND4_GetProcessInfo ()
- const char ∗ sound4cl_GetVersion ()
- unsigned int sound4cl_GetChunkSizeInFrames ()
- unsigned int ∗ sound4cl_GetPossibleChunkSizeInFrames ()
- unsigned int sound4cl_GetChannelCount ()
- unsigned int sound4cl_GetAudioInputCount ()
- unsigned int sound4cl_GetAudioOutputCount ()
- unsigned int sound4cl_GetSampleRate ()
- void sound4cl_SetLoggerCallback (sound4cl_loggerfn logger)
- void sound4cl_SetLogSeverity (enum sound4cl_LogSeverity severity)
- struct sound4cl_CParameters ∗ sound4cl_NewParameters ()
- void sound4cl_FreeParameters (struct sound4cl_CParameters ∗params)
- void sound4cl_SetParameter (struct sound4cl_CParameters ∗params, const char ∗name, const char ∗value)
- const char ∗ sound4cl_GetParameter (struct sound4cl_CParameters ∗params, const char ∗name)
- void sound4cl_FreeParameterValue (const char ∗value)
- struct sound4cl_CInstance ∗ sound4cl_InitProcess (const char ∗LoginKey, const char ∗RadioName, const char ∗Access_Key_ID, const char ∗Access_Key_Secret, const char ∗save_path)
- struct sound4cl_CInstance ∗ sound4cl_InitProcess2 (const char ∗LoginKey, const char ∗RadioName, const char ∗Access_Key_ID, const char ∗Access_Key_Secret, const char ∗save_path, const struct sound4cl_↩CParameters ∗parameters)
- struct sound4cl_CInstance ∗ sound4cl_InitProcess3 (const char ∗LoginKey, const char ∗RadioName, const char ∗Access_Key_ID, const char ∗Access_Key_Secret, const char ∗save_path, const struct sound4cl_↩CParameters ∗parameters, unsigned int frames_per_chunk)
- unsigned int sound4cl_GetProcessChunkFrames (struct sound4cl_CInstance ∗instance)
- int sound4cl_TerminateProcess (struct sound4cl_CInstance ∗instance)
- void sound4cl_ExitProcess (struct sound4cl_CInstance ∗instance)
- struct sound4cl_CBus ∗ sound4cl_NewBus ()
- void sound4cl_FreeBus (struct sound4cl_CBus ∗bus)
- void sound4cl_SetInstanceBus (struct sound4cl_CParameters ∗params, struct sound4cl_CBus ∗bus)
- void sound4cl_SetPresetManager (struct sound4cl_CParameters ∗params, sound4cl_storage_reader reader, sound4cl_storage_reader_free, sound4cl_storage_writer writer, sound4cl_storage_exists exists, sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free, sound4cl_storage_remove remove, sound4cl_storage_rename rename, int IsReadOnly, void ∗userdata)
- void sound4cl_PresetManager_InformChange (struct sound4cl_CInstance ∗instance, const fs_char ∗relative_path, enum sound4cl_PresetChange_Kind change_kind)
- void sound4cl_SetMetadata (struct sound4cl_CInstance ∗instance, const char ∗key, const char ∗value)
- void sound4cl_SetMetadataMulti (struct sound4cl_CInstance ∗instance, const char ∗∗keyvalue)
- const char ∗∗ sound4cl_GetMetadataInfos (struct sound4cl_CInstance ∗instance)
- void sound4cl_FreeMetadataInfos (struct sound4cl_CInstance ∗instance, const char ∗∗infos)

- void sound4cl_StartUpdateThread (struct sound4cl_CInstance ∗instance, unsigned int port)
- void sound4cl_StopUpdateThread (struct sound4cl_CInstance ∗instance)
- int sound4cl_WaitUpdateThreadReady (struct sound4cl_CInstance ∗instance, int milliseconds)
- void sound4cl_ProcessAudio (struct sound4cl_CInstance ∗instance, const float ∗input, float ∗output)
- void sound4cl_ProcessAudio_Planar (struct sound4cl_CInstance ∗instance, float const ∗const ∗input, float ∗const ∗output)
- float ∗ sound4cl_GetBufferIn (struct sound4cl_CInstance ∗instance)
- float ∗ sound4cl_GetBufferOut (struct sound4cl_CInstance ∗instance)
- unsigned int sound4cl_GetEstimatedDelay (struct sound4cl_CInstance ∗instance)
- const char ∗ sound4cl_GetFormatName (const enum sound4cl_SampleFormat fmt)
- enum sound4cl_SampleFormat sound4cl_GetFormatFromName (const char ∗name)
- unsigned int sound4cl_GetBytesFromFormat (const enum sound4cl_SampleFormat fmt)
- unsigned int sound4cl_GetMaxPacketFrame (struct sound4cl_CInstance ∗instance)
- unsigned int sound4cl_AddAudio (struct sound4cl_CInstance ∗instance, const uint8_t ∗payload, unsigned int nFrame, enum sound4cl_SampleFormat fmt)
- unsigned int sound4cl_AddPadAudio (struct sound4cl_CInstance ∗instance)
- unsigned int sound4cl_GetOutputCount (struct sound4cl_CInstance ∗instance)
- unsigned int sound4cl_GetAudio (struct sound4cl_CInstance ∗instance, uint8_t ∗payload, unsigned int max↩_nFrame, enum sound4cl_SampleFormat fmt)
- void sound4cl_AudioConvertFrom (const uint8_t ∗payload, float ∗output, size_t nSpl, enum sound4cl_SampleFormat fmt)
- void sound4cl_AudioConvertTo (const float ∗input, uint8_t ∗payload, size_t nSpl, enum sound4cl_SampleFormat fmt)
- void sound4cl_StereoToMono (const float ∗input, float ∗output, size_t nFrame)
- void sound4cl_MonoToStereo (const float ∗input, float ∗output, size_t nFrame)
- void sound4cl_StereoToMono_Planar (const float ∗inputL, const float ∗inputR, float ∗output, size_t nFrame)
- void sound4cl_MonoToStereo_Planar (const float ∗input, float ∗outputL, float ∗outputR, size_t nFrame)
- void sound4cl_AudioMonoFromLiveStereo (const uint8_t ∗payload, float ∗output)
- void sound4cl_AudioMonoToLiveStereo (const float ∗input, uint8_t ∗payload)
- struct sound4cl_CClientInstance ∗ sound4cl_NewClient (struct sound4cl_CInstance ∗instance)
- void sound4cl_DeleteClient (struct sound4cl_CClientInstance ∗client)
- const char ∗ sound4cl_ProcessJson (struct sound4cl_CClientInstance ∗client, const char ∗json_str, int ∗need_save)
- void sound4cl_FreeJsonAnswer (const char ∗json_str)
- int sound4cl_SaveState (struct sound4cl_CInstance ∗instance)
- uint64_t sound4cl_Webserver_tcp (unsigned int listenport, unsigned int listenport_secure, const char ∗socket_ip, unsigned int socket_port)
- 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)
- uint64_t sound4cl_Webserver (unsigned int listenport, unsigned int listenport_secure, struct sound4cl_↩CInstance ∗instance)
- int sound4cl_Webserver_Stop (uint64_t id, int timeout_ms)
- int sound4cl_Webserver_Status (uint64_t id)
- void sound4cl_Webserver_SetAppHealth (struct sound4cl_CInstance ∗instance, int httpcode, const char ∗contenttype, const char ∗content)
- void sound4cl_Webserver_GetAppHealth (struct sound4cl_CInstance ∗instance, int ∗httpcode, char ∗∗contenttype, char ∗∗content)
- void sound4cl_Webserver_FreeString (char ∗str)

11.17.1 Macro Definition Documentation

11.17.1.1 fs_strdup

```
#define fs_strdup wcsdup
```

Definition at line 17 of file sound4cl_cdef.h.

11.17.1.2 SOUND4_INVALID_WEBSERVER_ID

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

Definition at line 62 of file sound4cl_cdef.h.

11.17.1.3 SOUND4_WEBSERVER_HTTP_OK

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

Definition at line 63 of file sound4cl_cdef.h.

11.17.1.4 SOUND4_WEBSERVER_HTTPS_OK

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

Definition at line 64 of file sound4cl_cdef.h.

11.17.2 Typedef Documentation

11.17.2.1 fs_char

```
typedef wchar_t fs_char
```

Definition at line 18 of file sound4cl_cdef.h.

11.17.2.2 sound4cl_loggerfn

```
typedef void(* sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *)
```

Definition at line 102 of file sound4cl_cdef.h.

11.17.2.3 sound4cl_storage_exists

```
typedef int(* sound4cl_storage_exists) (const fs_char *filename, void *userdata)
```

Definition at line 143 of file sound4cl_cdef.h.

11.17.2.4 sound4cl_storage_getall

```
typedef fs_char**(* sound4cl_storage_getall) (void *userdata)
```

Definition at line 144 of file sound4cl_cdef.h.

11.17.2.5 sound4cl_storage_getall_free

```
typedef void(* sound4cl_storage_getall_free) (fs_char **all, void *userdata)
```

Definition at line 145 of file sound4cl_cdef.h.

11.17.2.6 sound4cl_storage_reader

```
typedef char*(* sound4cl_storage_reader) (const fs_char *filename, void *userdata)
```

Definition at line 140 of file sound4cl_cdef.h.

11.17.2.7 sound4cl_storage_reader_free

```
typedef void(* sound4cl_storage_reader_free) (char *content, void *userdata)
```

Definition at line 141 of file sound4cl_cdef.h.

11.17.2.8 sound4cl_storage_remove

```
typedef int(* sound4cl_storage_remove) (const fs_char *filename, void *userdata)
```

Definition at line 146 of file sound4cl_cdef.h.

11.17.2.9 sound4cl_storage_rename

```
typedef int(* sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)
```

Definition at line 147 of file sound4cl_cdef.h.

11.17.2.10 sound4cl_storage_writer

```
typedef int(* sound4cl_storage_writer) (const fs_char *filename, const char *content, void
*userdata)
```

Definition at line 142 of file sound4cl_cdef.h.

11.17.3 Enumeration Type Documentation

11.17.3.1 sound4cl_LogSeverity

```
enum sound4cl_LogSeverity
```

Enumerator

sound4cl_none	
sound4cl_fatal	
sound4cl_error	
sound4cl_warning	
sound4cl_info	
sound4cl_verbose	
sound4cl_verbose2	
sound4cl_verbose3	
sound4cl_verbose4	
sound4cl_verbose5	

Definition at line 41 of file sound4cl_cdef.h.

11.17.3.2 sound4cl_PresetChange_Kind

```
enum sound4cl_PresetChange_Kind
```

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 57 of file sound4cl_cdef.h.

11.17.3.3 sound4cl_SampleFormat

```
enum sound4cl_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 69 of file sound4cl_cdef.h.

11.17.4 Function Documentation

11.17.4.1 sound4cl_AddAudio()

```
unsigned int sound4cl_AddAudio (
            struct sound4cl_CInstance * instance,
            const uint8_t * payload,
            unsigned int nFrame,
            enum sound4cl_SampleFormat fmt )
```

11.17.4.2 sound4cl_AddPadAudio()

```
unsigned int sound4cl_AddPadAudio (
            struct sound4cl_CInstance * instance )
```

11.17.4.3 sound4cl_AudioConvertFrom()

```
void sound4cl_AudioConvertFrom (
            const uint8_t * payload,
            float * output,
            size_t nSpl,
            enum sound4cl_SampleFormat fmt )
```

11.17.4.4 sound4cl_AudioConvertTo()

```
void sound4cl_AudioConvertTo (
            const float * input,
            uint8_t * payload,
            size_t nSpl,
            enum sound4cl_SampleFormat fmt )
```

11.17.4.5 sound4cl_AudioMonoFromLiveStereo()

```
void sound4cl_AudioMonoFromLiveStereo (
            const uint8_t * payload,
            float * output )
```

11.17.4.6 sound4cl_AudioMonoToLiveStereo()

```
void sound4cl_AudioMonoToLiveStereo (
            const float * input,
            uint8_t * payload )
```

11.17.4.7 sound4cl_DeleteClient()

```
void sound4cl_DeleteClient (
            struct sound4cl_CClientInstance * client )
```

11.17.4.8 sound4cl_ExitProcess()

```
void sound4cl_ExitProcess (
            struct sound4cl_CInstance * instance )
```

11.17.4.9 sound4cl_FreeBus()

```
void sound4cl_FreeBus (
            struct sound4cl_CBus * bus )
```

11.17.4.10 sound4cl_FreeJsonAnswer()

```
void sound4cl_FreeJsonAnswer (
            const char * json_str )
```

11.17.4.11 sound4cl_FreeMetadataInfos()

```
void sound4cl_FreeMetadataInfos (
            struct sound4cl_CInstance * instance,
            const char ** infos )
```

11.17.4.12 sound4cl_FreeParameters()

```
void sound4cl_FreeParameters (
            struct sound4cl_CParameters * params )
```

11.17.4.13 sound4cl_FreeParameterValue()

```
void sound4cl_FreeParameterValue (
            const char * value )
```

11.17.4.14 sound4cl_GetAudio()

```
unsigned int sound4cl_GetAudio (
            struct sound4cl_CInstance * instance,
            uint8_t * payload,
            unsigned int max_nFrame,
            enum sound4cl_SampleFormat fmt )
```

11.17.4.15 sound4cl_GetAudioInputCount()

```
unsigned int sound4cl_GetAudioInputCount ( )
```

11.17.4.16 sound4cl_GetAudioOutputCount()

```
unsigned int sound4cl_GetAudioOutputCount ( )
```

11.17.4.17 sound4cl_GetBufferIn()

```
float* sound4cl_GetBufferIn (
            struct sound4cl_CInstance * instance )
```

11.17.4.18 sound4cl_GetBufferOut()

```
float* sound4cl_GetBufferOut (
            struct sound4cl_CInstance * instance )
```

11.17.4.19 sound4cl_GetBytesFromFormat()

```
unsigned int sound4cl_GetBytesFromFormat (
            const enum sound4cl_SampleFormat fmt )
```

11.17.4.20 sound4cl_GetChannelCount()

```
unsigned int sound4cl_GetChannelCount ( )
```

11.17.4.21 sound4cl_GetChunkSizeInFrames()

```
unsigned int sound4cl_GetChunkSizeInFrames ( )
```

11.17.4.22 sound4cl_GetEstimatedDelay()

```
unsigned int sound4cl_GetEstimatedDelay (
            struct sound4cl_CInstance * instance )
```

11.17.4.23 sound4cl_GetFormatFromName()

```
enum sound4cl_SampleFormat sound4cl_GetFormatFromName (
            const char * name )
```

11.17.4.24 sound4cl_GetFormatName()

```
const char* sound4cl_GetFormatName (
            const enum sound4cl_SampleFormat fmt )
```

11.17.4.25 sound4cl_GetMaxPacketFrame()

```
unsigned int sound4cl_GetMaxPacketFrame (
            struct sound4cl_CInstance * instance )
```

11.17.4.26 sound4cl_GetMetadataInfos()

```
const char** sound4cl_GetMetadataInfos (
            struct sound4cl_CInstance * instance )
```

11.17.4.27 sound4cl_GetOutputCount()

```
unsigned int sound4cl_GetOutputCount (
            struct sound4cl_CInstance * instance )
```

11.17.4.28 sound4cl_GetParameter()

```
const char* sound4cl_GetParameter (
            struct sound4cl_CParameters * params,
            const char * name )
```

11.17.4.29 sound4cl_GetPossibleChunkSizeInFrames()

```
unsigned int* sound4cl_GetPossibleChunkSizeInFrames ( )
```

11.17.4.30 sound4cl_GetProcessChunkFrames()

```
unsigned int sound4cl_GetProcessChunkFrames (
            struct sound4cl_CInstance * instance )
```

11.17.4.31 sound4cl_GetSampleRate()

```
unsigned int sound4cl_GetSampleRate ( )
```

11.17.4.32 sound4cl_GetVersion()

```
const char* sound4cl_GetVersion ( )
```

11.17.4.33 sound4cl_InitProcess()

```
struct sound4cl_CInstance* sound4cl_InitProcess (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_Secret,
            const char * save_path )
```

11.17.4.34 sound4cl_InitProcess2()

```
struct sound4cl_CInstance* sound4cl_InitProcess2 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_Secret,
            const char * save_path,
            const struct sound4cl_CParameters * parameters )
```

11.17.4.35 sound4cl_InitProcess3()

```
struct sound4cl_CInstance* sound4cl_InitProcess3 (
            const char * LoginKey,
            const char * RadioName,
            const char * Access_Key_ID,
            const char * Access_Key_Secret,
            const char * save_path,
            const struct sound4cl_CParameters * parameters,
            unsigned int frames_per_chunk )
```

11.17.4.36 sound4cl_MonoToStereo()

```
void sound4cl_MonoToStereo (
            const float * input,
            float * output,
            size_t nFrame )
```

11.17.4.37 sound4cl_MonoToStereo_Planar()

```
void sound4cl_MonoToStereo_Planar (
            const float * input,
            float * outputL,
            float * outputR,
            size_t nFrame )
```

11.17.4.38 sound4cl_NewBus()

```
struct sound4cl_CBus* sound4cl_NewBus ( )
```

11.17.4.39 sound4cl_NewClient()

```
struct sound4cl_CClientInstance* sound4cl_NewClient (
            struct sound4cl_CInstance * instance )
```

11.17.4.40 sound4cl_NewParameters()

```
struct sound4cl_CParameters* sound4cl_NewParameters ( )
```

11.17.4.41 sound4cl_PresetManager_InformChange()

```
void sound4cl_PresetManager_InformChange (
            struct sound4cl_CInstance * instance,
            const fs_char * relative_path,
            enum sound4cl_PresetChange_Kind change_kind )
```

11.17.4.42 sound4cl_ProcessAudio()

```
void sound4cl_ProcessAudio (
            struct sound4cl_CInstance * instance,
            const float * input,
            float * output )
```

11.17.4.43 sound4cl_ProcessAudio_Planar()

```
void sound4cl_ProcessAudio_Planar (
            struct sound4cl_CInstance * instance,
            float const *const * input,
            float *const * output )
```

11.17.4.44 sound4cl_ProcessJson()

```
const char* sound4cl_ProcessJson (
            struct sound4cl_CClientInstance * client,
            const char * json_str,
            int * need_save )
```

11.17.4.45 sound4cl_SaveState()

```
int sound4cl_SaveState (
            struct sound4cl_CInstance * instance )
```

11.17.4.46 sound4cl_SetInstanceBus()

```
void sound4cl_SetInstanceBus (
            struct sound4cl_CParameters * params,
            struct sound4cl_CBus * bus )
```

11.17.4.47 sound4cl_SetLoggerCallback()

```
void sound4cl_SetLoggerCallback (
            sound4cl_loggerfn logger )
```

11.17.4.48 sound4cl_SetLogSeverity()

```
void sound4cl_SetLogSeverity (
            enum sound4cl_LogSeverity severity )
```

11.17.4.49 sound4cl_SetMetadata()

```
void sound4cl_SetMetadata (
            struct sound4cl_CInstance * instance,
            const char * key,
            const char * value )
```

11.17.4.50 sound4cl_SetMetadataMulti()

```
void sound4cl_SetMetadataMulti (
            struct sound4cl_CInstance * instance,
            const char ** keyvalue )
```

11.17.4.51 sound4cl_SetParameter()

```
void sound4cl_SetParameter (
            struct sound4cl_CParameters * params,
            const char * name,
            const char * value )
```

11.17.4.52 sound4cl_SetPresetManager()

```
void sound4cl_SetPresetManager (
            struct sound4cl_CParameters * params,
            sound4cl_storage_reader reader,
            sound4cl_storage_reader_free ,
            sound4cl_storage_writer writer,
            sound4cl_storage_exists exists,
            sound4cl_storage_getall getall,
            sound4cl_storage_getall_free getall_free,
            sound4cl_storage_remove remove,
            sound4cl_storage_rename rename,
            int IsReadOnly,
            void * userdata )
```

11.17.4.53 sound4cl_SOUND4_GetProcessInfo()

```
const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo ( )
```

11.17.4.54 sound4cl_StartUpdateThread()

```
void sound4cl_StartUpdateThread (
            struct sound4cl_CInstance * instance,
            unsigned int port )
```

11.17.4.55 sound4cl_StereoToMono()

```
void sound4cl_StereoToMono (
            const float * input,
            float * output,
            size_t nFrame )
```

11.17.4.56 sound4cl_StereoToMono_Planar()

```
void sound4cl_StereoToMono_Planar (
            const float * inputL,
            const float * inputR,
            float * output,
            size_t nFrame )
```

11.17.4.57 sound4cl_StopUpdateThread()

```
void sound4cl_StopUpdateThread (
            struct sound4cl_CInstance * instance )
```

11.17.4.58 sound4cl_TerminateProcess()

```
int sound4cl_TerminateProcess (
            struct sound4cl_CInstance * instance )
```

11.17.4.59 sound4cl_WaitUpdateThreadReady()

```
int sound4cl_WaitUpdateThreadReady (
            struct sound4cl_CInstance * instance,
            int milliseconds )
```

11.17.4.60 sound4cl_Webserver()

```
uint64_t sound4cl_Webserver (
            unsigned int listenport,
            unsigned int listenport_secure,
            struct sound4cl_CInstance * instance )
```

11.17.4.61 sound4cl_Webserver_FreeString()

```
void sound4cl_Webserver_FreeString (
            char * str )
```

11.17.4.62 sound4cl_Webserver_GetAppHealth()

```
void sound4cl_Webserver_GetAppHealth (
            struct sound4cl_CInstance * instance,
            int * httpcode,
            char ** contenttype,
            char ** content )
```

11.17.4.63 sound4cl_Webserver_SetAppHealth()

```
void sound4cl_Webserver_SetAppHealth (
            struct sound4cl_CInstance * instance,
            int httpcode,
            const char * contenttype,
            const char * content )
```

11.17.4.64 sound4cl_Webserver_Status()

```
int sound4cl_Webserver_Status (
            uint64_t id )
```

11.17.4.65 sound4cl_Webserver_Stop()

```
int sound4cl_Webserver_Stop (
            uint64_t id,
            int timeout_ms )
```

11.17.4.66 sound4cl_Webserver_tcp()

```
uint64_t sound4cl_Webserver_tcp (
            unsigned int listenport,
            unsigned int listenport_secure,
            const char * socket_ip,
            unsigned int socket_port )
```

11.17.4.67 sound4cl_Webserver_tcp2()

```
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 )
```

11.18 sound4cl_cdef.h

```
00001
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #ifdef __unix__
00010     #include <sys/types.h>
00011 #endif
00012
00013 #ifdef _WIN32
00014     #include <stringapiset.h>
00015     #include <wchar.h>
00017     #define fs_strdup wcsdup
00018     typedef wchar_t fs_char;
00019     #ifdef _MSC_VER
00020         #define __BYTE_ORDER__  REG_DWORD
00021         #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00022         #define __ORDER_BIG_ENDIAN__    REG_DWORD_BIG_ENDIAN
00023     #endif
00024 #else // !_WIN32
00026     #define fs_strdup strdup
00027     typedef char fs_char;
00028 #endif // !_WIN32
00029
00030 #ifdef __cplusplus
00031 extern "C" {
00032 #endif // __cplusplus
00033
00034 struct sound4cl_SOUND4_ProcessInfo {
00035     size_t struct_size;
00036     const char *name;
00037     const char *prefix;
00038     const char *shortname;
00039     const char *version;
00040 };
00041 enum sound4cl_LogSeverity {
00042     sound4cl_none = 0,
```

```
    sound4cl_fatal = 1,
    sound4cl_error = 2,
    sound4cl_warning = 3,
    sound4cl_info = 4,
    sound4cl_verbose = 5,
    sound4cl_verbose2 = 6,
    sound4cl_verbose3 = 7,
    sound4cl_verbose4 = 8,
    sound4cl_verbose5 = 9,
};
struct sound4cl_CInstance;
struct sound4cl_CParameters;
struct sound4cl_CBus;
struct sound4cl_CClientInstance;
enum sound4cl_PresetChange_Kind {
    change_kind_created = 1,
    change_kind_modified   ,
    change_kind_deleted    ,
};
#define SOUND4_INVALID_WEBSERVER_ID            ((uint64_t)-1)
#define SOUND4_WEBSERVER_HTTP_OK               (1«0)
#define SOUND4_WEBSERVER_HTTPS_OK              (1«1)

enum sound4cl_SampleFormat {
    INVALID_FORMAT=0,
    S16_LE,
    S16_BE,
    S24_LE,
    S24_BE,
    S32_LE,
    S32_BE,
    F32_LE,
    F32_BE,

    // Native definitions, only links to the physical ones
    #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
        S16_NATIVE=S16_LE,
        S24_NATIVE=S24_LE,
        S32_NATIVE=S32_LE,
        F32_NATIVE=F32_LE,
    #else
        S16_NATIVE=S16_BE,
        S24_NATIVE=S24_BE,
        S32_NATIVE=S32_BE,
        F32_NATIVE=F32_BE,
    #endif
};

const struct sound4cl_SOUND4_ProcessInfo* sound4cl_SOUND4_GetProcessInfo();
const char *sound4cl_GetVersion();
unsigned int sound4cl_GetChunkSizeInFrames();
unsigned int* sound4cl_GetPossibleChunkSizeInFrames();
unsigned int sound4cl_GetChannelCount();
unsigned int sound4cl_GetAudioInputCount();
unsigned int sound4cl_GetAudioOutputCount();
unsigned int sound4cl_GetSampleRate();
typedef void (*sound4cl_loggerfn) (enum sound4cl_LogSeverity, const char *);
void sound4cl_SetLoggerCallback( sound4cl_loggerfn logger );
void sound4cl_SetLogSeverity( enum sound4cl_LogSeverity severity );
struct sound4cl_CParameters *sound4cl_NewParameters();
void sound4cl_FreeParameters(struct sound4cl_CParameters *params);
void sound4cl_SetParameter(struct sound4cl_CParameters *params, const char *name, const char *value);
const char *sound4cl_GetParameter(struct sound4cl_CParameters *params, const char *name);
void sound4cl_FreeParameterValue(const char *value);
struct sound4cl_CInstance *sound4cl_InitProcess(
        const char *LoginKey,
        const char *RadioName,
        const char *Access_Key_ID,
        const char *Access_Key_Secret,
        const char *save_path
    );
struct sound4cl_CInstance *sound4cl_InitProcess2(
        const char *LoginKey,
        const char *RadioName,
        const char *Access_Key_ID,
        const char *Access_Key_Secret,
        const char *save_path,
        const struct sound4cl_CParameters *parameters
    );
struct sound4cl_CInstance *sound4cl_InitProcess3(
        const char *LoginKey,
        const char *RadioName,
        const char *Access_Key_ID,
        const char *Access_Key_Secret,
        const char *save_path,
        const struct sound4cl_CParameters *parameters,
        unsigned int frames_per_chunk
```

```
    );
unsigned int sound4cl_GetProcessChunkFrames(struct sound4cl_CInstance *instance);
int sound4cl_TerminateProcess(struct sound4cl_CInstance *instance);
void sound4cl_ExitProcess(struct sound4cl_CInstance *instance);
struct sound4cl_CBus* sound4cl_NewBus();
void sound4cl_FreeBus(struct sound4cl_CBus *bus);
void sound4cl_SetInstanceBus(struct sound4cl_CParameters *params, struct sound4cl_CBus *bus);
typedef char * (*sound4cl_storage_reader) (const fs_char *filename, void* userdata);
typedef void (*sound4cl_storage_reader_free) (char *content, void* userdata);
typedef int (*sound4cl_storage_writer) (const fs_char *filename, const char *content, void* userdata);
typedef int (*sound4cl_storage_exists) (const fs_char *filename, void* userdata);
typedef fs_char** (*sound4cl_storage_getall) (void* userdata);
typedef void (*sound4cl_storage_getall_free) (fs_char**all, void* userdata);
typedef int (*sound4cl_storage_remove) (const fs_char *filename, void* userdata);
typedef int (*sound4cl_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
void sound4cl_SetPresetManager(struct sound4cl_CParameters *params
    , sound4cl_storage_reader reader, sound4cl_storage_reader_free
    , sound4cl_storage_writer writer
    , sound4cl_storage_exists exists
    , sound4cl_storage_getall getall, sound4cl_storage_getall_free getall_free
    , sound4cl_storage_remove remove
    , sound4cl_storage_rename rename
    , int IsReadOnly
    , void* userdata
    );
void sound4cl_PresetManager_InformChange(struct sound4cl_CInstance *instance, const fs_char
    *relative_path, enum sound4cl_PresetChange_Kind change_kind);
void sound4cl_SetMetadata(struct sound4cl_CInstance *instance, const char* key, const char* value);
void sound4cl_SetMetadataMulti(struct sound4cl_CInstance *instance, const char** keyvalue);
const char** sound4cl_GetMetadataInfos(struct sound4cl_CInstance *instance);
void sound4cl_FreeMetadataInfos(struct sound4cl_CInstance *instance, const char** infos);
void sound4cl_StartUpdateThread(struct sound4cl_CInstance *instance, unsigned int port);
void sound4cl_StopUpdateThread(struct sound4cl_CInstance *instance);
int sound4cl_WaitUpdateThreadReady(struct sound4cl_CInstance *instance, int milliseconds);
void sound4cl_ProcessAudio(struct sound4cl_CInstance *instance, const float *input, float *output );
void sound4cl_ProcessAudio_Planar(struct sound4cl_CInstance *instance
    , float const * const *input
    , float * const *output);
float *sound4cl_GetBufferIn(struct sound4cl_CInstance *instance);
float *sound4cl_GetBufferOut(struct sound4cl_CInstance *instance);
unsigned int sound4cl_GetEstimatedDelay(struct sound4cl_CInstance *instance);
const char *sound4cl_GetFormatName(const enum sound4cl_SampleFormat fmt);
enum sound4cl_SampleFormat sound4cl_GetFormatFromName(const char *name);
unsigned int sound4cl_GetBytesFromFormat(const enum sound4cl_SampleFormat fmt);
unsigned int sound4cl_GetMaxPacketFrame(struct sound4cl_CInstance *instance);
unsigned int sound4cl_AddAudio(struct sound4cl_CInstance *instance, const uint8_t *payload, unsigned
    int nFrame, enum sound4cl_SampleFormat fmt );
unsigned int sound4cl_AddPadAudio(struct sound4cl_CInstance *instance);
unsigned int sound4cl_GetOutputCount(struct sound4cl_CInstance *instance);
unsigned int sound4cl_GetAudio(struct sound4cl_CInstance *instance, uint8_t *payload, unsigned int
    max_nFrame, enum sound4cl_SampleFormat fmt );
void sound4cl_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
    sound4cl_SampleFormat fmt);
void sound4cl_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
    sound4cl_SampleFormat fmt);
void sound4cl_StereoToMono(const float *input, float *output, size_t nFrame);
void sound4cl_MonoToStereo(const float *input, float *output, size_t nFrame);
void sound4cl_StereoToMono_Planar(const float *inputL, const float *inputR, float *output, size_t
    nFrame);
void sound4cl_MonoToStereo_Planar(const float *input, float *outputL, float *outputR, size_t nFrame);
void sound4cl_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
void sound4cl_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
struct sound4cl_CClientInstance *sound4cl_NewClient(struct sound4cl_CInstance *instance);
void sound4cl_DeleteClient(struct sound4cl_CClientInstance *client);
const char *sound4cl_ProcessJson(struct sound4cl_CClientInstance *client, const char *json_str, int
    *need_save);
void sound4cl_FreeJsonAnswer(const char *json_str);
int sound4cl_SaveState(struct sound4cl_CInstance *instance);
uint64_t sound4cl_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure, const char
    *socket_ip, unsigned int socket_port);
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);
uint64_t sound4cl_Webserver(unsigned int listenport, unsigned int listenport_secure, struct
    sound4cl_CInstance *instance);
int sound4cl_Webserver_Stop(uint64_t id, int timeout_ms);
int sound4cl_Webserver_Status(uint64_t id);
void sound4cl_Webserver_SetAppHealth(struct sound4cl_CInstance *instance, int httpcode, const char*
    contenttype, const char* content);
void sound4cl_Webserver_GetAppHealth(struct sound4cl_CInstance *instance, int* httpcode, char**
    contenttype, char** content);
void sound4cl_Webserver_FreeString(char* str);

#ifdef __cplusplus
} // extern "C"
#endif // __cplusplus
```

Index

~CBus
sound4::CBus, 85
~CClient
sound4::CInstance::CClient, 87
sound4::cloudx1::CInstance::CClient, 89
sound4::cloudx1::dyn::CInstance::CClient, 91
~CDynLoader
sound4::cloudx1::dyn::helper::CDynLoader, 106
sound4::helper::CDynLoader, 109
~CInstance
sound4::CInstance, 113
sound4::cloudx1::CInstance, 124
sound4::cloudx1::dyn::CInstance, 135
~CPresetLoader
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 150
~MyPresetManager
MyPresetManager, 176

AddAudio
sound4::CInstance, 113
sound4::cloudx1::CInstance, 124
sound4::cloudx1::dyn::CDynLib, 94
sound4::cloudx1::dyn::CInstance, 135
AddPadAudio
sound4::cloudx1::dyn::CDynLib, 95
Audio convert helpers, 48
cloudx1_AudioConvertFrom, 48
cloudx1_AudioConvertTo, 49
cloudx1_AudioMonoFromLiveStereo, 49
cloudx1_AudioMonoToLiveStereo, 49
cloudx1_MonoToStereo, 50
cloudx1_MonoToStereo_Planar, 50
cloudx1_StereoToMono, 51
cloudx1_StereoToMono_Planar, 51
Audio managing (any audio), 43
cloudx1_AddAudio, 45
cloudx1_AddPadAudio, 45
cloudx1_GetAudio, 45
cloudx1_GetBytesFromFormat, 46
cloudx1_GetFormatFromName, 46
cloudx1_GetFormatName, 47
cloudx1_GetMaxPacketFrame, 47
cloudx1_GetOutputCount, 47
cloudx1_SampleFormat, 44
F32_BE, 44
F32_LE, 44
F32_NATIVE, 44
INVALID_FORMAT, 44
S16_BE, 44
S16_LE, 44
S16_NATIVE, 44
S24_BE, 44
S24_LE, 44
S24_NATIVE, 44
S32_BE, 44
S32_LE, 44
S32_NATIVE, 44
AudioConvertFrom
sound4::cloudx1::dyn::CDynLib, 95
sound4::CProcessor, 154
AudioConvertTo
sound4::cloudx1::dyn::CDynLib, 95
sound4::CProcessor, 154
AudioMonoFromLiveStereo
sound4::cloudx1::dyn::CDynLib, 95
sound4::CProcessor, 155
AudioMonoToLiveStereo
sound4::cloudx1::dyn::CDynLib, 95
sound4::CProcessor, 155

CBus
sound4::CBus, 85
sound4::CProcessor, 161
CClient
sound4::CInstance::CClient, 87
sound4::cloudx1::CInstance::CClient, 89
sound4::cloudx1::dyn::CInstance::CClient, 90, 91
CDynLib
sound4::cloudx1::dyn::CDynLib, 94
CDynLoader
sound4::cloudx1::dyn::helper::CDynLoader, 106
sound4::helper::CDynLoader, 109
change_kind_created
Custom preset managagement, 36
sound4cl_cdef.h, 260
change_kind_deleted
Custom preset managagement, 36
sound4cl_cdef.h, 260
change_kind_modified
Custom preset managagement, 36
sound4cl_cdef.h, 260
CHANNEL_COUNT
sound4.x1.cloud-pipe.cpp, 200
CHUNK_SIZE
sound4.x1.cloud-pipe.cpp, 201
ChunkFrames
sound4::cloudx1, 78
sound4::cloudx1::dyn, 81

CInstance
sound4::CInstance, 113
sound4::cloudx1::CInstance, 124
sound4::cloudx1::dyn::CInstance, 134, 135
sound4::CProcessor, 161
Client's JSon config interfacing, 52
cloudx1_DeleteClient, 52
cloudx1_FreeJsonAnswer, 52
cloudx1_NewClient, 53
cloudx1_ProcessJson, 53
cloudx1_SaveState, 54
Close
sound4::cloudx1::dyn::helper::CDynLoader, 106
sound4::helper::CDynLoader, 109
cloudx1_AddAudio
Audio managing (any audio), 45
cloudx1_AddPadAudio
Audio managing (any audio), 45
CLOUDX1_AUDIO_INPUT_COUNT
Global setup, 21
CLOUDX1_AUDIO_OUTPUT_COUNT
Global setup, 21
cloudx1_AudioConvertFrom
Audio convert helpers, 48
cloudx1_AudioConvertTo
Audio convert helpers, 49
CLOUDX1_AUDIOFRAME_COUNT
Global setup, 21
cloudx1_AudioMonoFromLiveStereo
Audio convert helpers, 49
cloudx1_AudioMonoToLiveStereo
Audio convert helpers, 49
cloudx1_CClientInstance, 144
CLOUDX1_CHANNEL_COUNT
Global setup, 21
cloudx1_CInstance, 144
cloudx1_CParameters, 145
CLOUDX1_DECL
sound4.x1.cloud.h, 208
cloudx1_DeleteClient
Client's JSon config interfacing, 52
cloudx1_ExitProcess
Process creation/destruction, 27
cloudx1_FreeJsonAnswer
Client's JSon config interfacing, 52
cloudx1_FreeMetadataInfos
Metadata Management, 37
cloudx1_FreeParameters
Process creation/destruction, 27
cloudx1_FreeParameterValue
Process creation/destruction, 27
cloudx1_GetAudio
Audio managing (any audio), 45
cloudx1_GetAudioInputCount
Global setup, 23
cloudx1_GetAudioOutputCount
Global setup, 23
cloudx1_GetBufferIn
Direct processing, 41
cloudx1_GetBufferOut
Direct processing, 41
cloudx1_GetBytesFromFormat
Audio managing (any audio), 46
cloudx1_GetChannelCount
Global setup, 23
cloudx1_GetChunkSizeInFrames
Global setup, 24
cloudx1_GetEstimatedDelay
Direct processing, 42
cloudx1_GetFormatFromName
Audio managing (any audio), 46
cloudx1_GetFormatName
Audio managing (any audio), 47
cloudx1_GetMaxPacketFrame
Audio managing (any audio), 47
cloudx1_GetMetadataInfos
Metadata Management, 38
cloudx1_GetOutputCount
Audio managing (any audio), 47
cloudx1_GetParameter
Process creation/destruction, 27
cloudx1_GetPossibleChunkSizeInFrames
Global setup, 24
cloudx1_GetProcessChunkFrames
Process creation/destruction, 28
cloudx1_GetSampleRate
Global setup, 24
cloudx1_GetVersion
Global setup, 25
CLOUDX1_HAS_CLOUDBUS
sound4.x1.cloud.h, 208
CLOUDX1_HAS_WEBSERVER
sound4.x1.cloud.h, 208
cloudx1_InitProcess
Process creation/destruction, 28
cloudx1_InitProcess2
Process creation/destruction, 28
cloudx1_InitProcess3
Process creation/destruction, 29
cloudx1_loggerfn
Global setup, 22
cloudx1_LogSeverity
Global setup, 22
cloudx1_MonoToStereo
Audio convert helpers, 50
cloudx1_MonoToStereo_Planar
Audio convert helpers, 50
cloudx1_NewClient
Client's JSon config interfacing, 53
cloudx1_NewParameters
Process creation/destruction, 30
cloudx1_PresetChange_Kind
Custom preset managagement, 35
cloudx1_PresetManager_InformChange
Custom preset managagement, 36
cloudx1_ProcessAudio

Direct processing, 42
cloudx1_ProcessAudio_Planar
Direct processing, 43
cloudx1_ProcessJson
Client's JSon config interfacing, 53
CLOUDX1_SAMPLE_RATE
Global setup, 22
cloudx1_SampleFormat
Audio managing (any audio), 44
cloudx1_SaveState
Client's JSon config interfacing, 54
cloudx1_SetLoggerCallback
Global setup, 25
cloudx1_SetLogSeverity
Global setup, 25
cloudx1_SetMetadata
Metadata Management, 38
cloudx1_SetMetadataMulti
Metadata Management, 39
cloudx1_SetParameter
Process creation/destruction, 30
cloudx1_SetPresetManager
Custom preset managagement, 36
cloudx1_StartUpdateThread
Update Management, 39
cloudx1_StereoToMono
Audio convert helpers, 51
cloudx1_StereoToMono_Planar
Audio convert helpers, 51
cloudx1_StopUpdateThread
Update Management, 40
cloudx1_storage_exists
Custom preset managagement, 32
cloudx1_storage_getall
Custom preset managagement, 33
cloudx1_storage_getall_free
Custom preset managagement, 33
cloudx1_storage_reader
Custom preset managagement, 33
cloudx1_storage_reader_free
Custom preset managagement, 34
cloudx1_storage_remove
Custom preset managagement, 34
cloudx1_storage_rename
Custom preset managagement, 34
cloudx1_storage_writer
Custom preset managagement, 35
cloudx1_TerminateProcess
Process creation/destruction, 31
cloudx1_WaitUpdateThreadReady
Update Management, 40
cloudx1_Webserver
Web server, 62
cloudx1_Webserver_FreeString
Web server, 63
cloudx1_Webserver_GetAppHealth
Web server, 63
cloudx1_Webserver_SetAppHealth
Web server, 63
cloudx1_Webserver_Status
Web server, 64
cloudx1_Webserver_Stop
Web server, 64
cloudx1_Webserver_tcp
Web server, 65
cloudx1_Webserver_tcp2
Web server, 65
CPresetLoader
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 150
CProcessor
sound4::CProcessor, 154
Create
sound4::CInstance, 113
sound4::cloudx1::CInstance, 124
sound4::cloudx1::dyn::CInstance, 135
Custom preset managagement, 31
change_kind_created, 36
change_kind_deleted, 36
change_kind_modified, 36
cloudx1_PresetChange_Kind, 35
cloudx1_PresetManager_InformChange, 36
cloudx1_SetPresetManager, 36
cloudx1_storage_exists, 32
cloudx1_storage_getall, 33
cloudx1_storage_getall_free, 33
cloudx1_storage_reader, 33
cloudx1_storage_reader_free, 34
cloudx1_storage_remove, 34
cloudx1_storage_rename, 34
cloudx1_storage_writer, 35
fs_char, 35

DeleteClient
sound4::cloudx1::dyn::CDynLib, 95
Direct processing, 40
cloudx1_GetBufferIn, 41
cloudx1_GetBufferOut, 41
cloudx1_GetEstimatedDelay, 42
cloudx1_ProcessAudio, 42
cloudx1_ProcessAudio_Planar, 43
dyn, 73
DynFunc_t
sound4::cloudx1::dyn::helper, 82
sound4::helper, 84
DynFuncHolder
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 173
sound4::helper::DynFuncHolder< T >, 174
DynLib_t
sound4::cloudx1::dyn::helper, 82
sound4::helper, 84

Environment variables and parameters, 19
error
Global setup, 23

sound4, 75
Exists
MyPresetManager, 177
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 150
ExitProcess
sound4::cloudx1::dyn::CDynLib, 96

F32_BE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
F32_LE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
F32_NATIVE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
fatal
Global setup, 23
sound4, 75
fnAddAudio
sound4::CProcessor, 161
fnAddPadAudio
sound4::CProcessor, 161
fnAudioConvertFrom
sound4::CProcessor, 161
fnAudioConvertTo
sound4::CProcessor, 162
fnAudioMonoFromLiveStereo
sound4::CProcessor, 162
fnAudioMonoToLiveStereo
sound4::CProcessor, 162
fnDeleteClient
sound4::CProcessor, 162
fnExitProcess
sound4::CProcessor, 162
fnFreeBus
sound4::CProcessor, 162
fnFreeJsonAnswer
sound4::CProcessor, 163
fnFreeMetadataInfos
sound4::CProcessor, 163
fnFreeParameters
sound4::CProcessor, 163
fnFreeParameterValue
sound4::CProcessor, 163
fnGetAudio
sound4::CProcessor, 163
fnGetAudioInputCount
sound4::CProcessor, 163
fnGetAudioOutputCount
sound4::CProcessor, 164
fnGetBufferIn
sound4::CProcessor, 164
fnGetBufferOut
sound4::CProcessor, 164
fnGetBytesFromFormat
sound4::CProcessor, 164
fnGetChannelCount
sound4::CProcessor, 164
fnGetChunkSizeInFrames
sound4::CProcessor, 164
fnGetEstimatedDelay
sound4::CProcessor, 165
fnGetFormatFromName
sound4::CProcessor, 165
fnGetFormatName
sound4::CProcessor, 165
fnGetMaxPacketFrame
sound4::CProcessor, 165
fnGetMetadataInfos
sound4::CProcessor, 165
fnGetOutputCount
sound4::CProcessor, 165
fnGetParameter
sound4::CProcessor, 166
fnGetPossibleChunkSizeInFrames
sound4::CProcessor, 166
fnGetProcessChunkFrames
sound4::CProcessor, 166
fnGetSampleRate
sound4::CProcessor, 166
fnGetVersion
sound4::CProcessor, 166
fnInitProcess
sound4::CProcessor, 166
fnInitProcess2
sound4::CProcessor, 167
fnInitProcess3
sound4::CProcessor, 167
fnMonoToStereo
sound4::CProcessor, 167
fnMonoToStereo_Planar
sound4::CProcessor, 167
fnNewBus
sound4::CProcessor, 167
fnNewClient
sound4::CProcessor, 167
fnNewParameters
sound4::CProcessor, 168
fnPresetManager_InformChange
sound4::CProcessor, 168
fnProcessAudio
sound4::CProcessor, 168
fnProcessAudio_Planar
sound4::CProcessor, 168
fnProcessJson
sound4::CProcessor, 168
fnSaveState
sound4::CProcessor, 168
fnSetInstanceBus
sound4::CProcessor, 169
fnSetLoggerCallback
sound4::CProcessor, 169
fnSetLogSeverity
sound4::CProcessor, 169

fnSetMetadata
sound4::CProcessor, 169
fnSetMetadataMulti
sound4::CProcessor, 169
fnSetParameter
sound4::CProcessor, 169
fnSetPresetManager
sound4::CProcessor, 170
fnStartUpdateThread
sound4::CProcessor, 170
fnStereoToMono
sound4::CProcessor, 170
fnStereoToMono_Planar
sound4::CProcessor, 170
fnStopUpdateThread
sound4::CProcessor, 170
fnTerminateProcess
sound4::CProcessor, 170
fnWaitUpdateThreadReady
sound4::CProcessor, 171
fnWebserver
sound4::CProcessor, 171
fnWebserver_FreeString
sound4::CProcessor, 171
fnWebserver_GetAppHealth
sound4::CProcessor, 171
fnWebserver_SetAppHealth
sound4::CProcessor, 171
fnWebserver_Status
sound4::CProcessor, 171
fnWebserver_Stop
sound4::CProcessor, 172
fnWebserver_tcp
sound4::CProcessor, 172
fnWebserver_tcp2
sound4::CProcessor, 172
format
sound4::cloudx1::dyn::helper::SampleFormat< float >, 182
sound4::cloudx1::dyn::helper::SampleFormat< int16_t >, 184
sound4::cloudx1::dyn::helper::SampleFormat< int32_t >, 186
sound4::cloudx1::helper::SampleFormat< float >, 183
sound4::cloudx1::helper::SampleFormat< int16_t >, 184
sound4::cloudx1::helper::SampleFormat< int32_t >, 186
sound4::helper::SampleFormat< float >, 183
sound4::helper::SampleFormat< int16_t >, 185
sound4::helper::SampleFormat< int32_t >, 187
FreeJsonAnswer
sound4::cloudx1::dyn::CDynLib, 96
FreeMetadataInfos
sound4::cloudx1::dyn::CDynLib, 96
FreeParameters
sound4::cloudx1::dyn::CDynLib, 96
FreeParameterValue
sound4::cloudx1::dyn::CDynLib, 96
fs_char
Custom preset managagement, 35
sound4cl_cdef.h, 258
fs_strdup
sound4.x1.cloud.hpp, 214
sound4.x1.cloud_dyn.hpp, 223
sound4cl_cdef.h, 257

Get
sound4::CBus, 86
GetAll
MyPresetManager, 177
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 150
GetAudio
sound4::CInstance, 114
sound4::cloudx1::CInstance, 125
sound4::cloudx1::dyn::CDynLib, 96
sound4::cloudx1::dyn::CInstance, 136
GetAudioInputCount
sound4::cloudx1::dyn::CDynLib, 97
sound4::CProcessor, 155
GetAudioOutputCount
sound4::cloudx1::dyn::CDynLib, 97
sound4::CProcessor, 155
GetBufferIn
sound4::CInstance, 114
sound4::cloudx1::CInstance, 125
sound4::cloudx1::dyn::CDynLib, 97
sound4::cloudx1::dyn::CInstance, 136
GetBufferOut
sound4::CInstance, 114
sound4::cloudx1::CInstance, 125
sound4::cloudx1::dyn::CDynLib, 97
sound4::cloudx1::dyn::CInstance, 136
GetBytesFromFormat
sound4::cloudx1::dyn::CDynLib, 97
sound4::CProcessor, 156
GetChannelCount
sound4::cloudx1::dyn::CDynLib, 97
sound4::CProcessor, 156
GetChunkFrames
sound4::CInstance, 115
sound4::cloudx1::CInstance, 126
sound4::cloudx1::dyn::CInstance, 137
GetChunkSizeInFrames
sound4::cloudx1::dyn::CDynLib, 98
sound4::CProcessor, 156
GetEnv
sound4.x1.cloud-pipe-cxx.cpp, 191
sound4.x1.cloud-pipe-dyn.cpp, 195
sound4cl-pipe.cpp, 237
GetEstimatedDelay
sound4::CInstance, 115
sound4::cloudx1::CInstance, 126
sound4::cloudx1::dyn::CDynLib, 98

sound4::cloudx1::dyn::CInstance, 137
GetFormatFromName
sound4::cloudx1::dyn::CDynLib, 98
sound4::CProcessor, 157
GetFormatName
sound4::cloudx1::dyn::CDynLib, 98
sound4::CProcessor, 157
GetMaxPacketFrame
sound4::cloudx1::dyn::CDynLib, 98
GetMetadataInfos
sound4::CInstance, 115
sound4::cloudx1::CInstance, 126
sound4::cloudx1::dyn::CDynLib, 98
sound4::cloudx1::dyn::CInstance, 137
GetOutputCount
sound4::cloudx1::dyn::CDynLib, 99
GetParam
sound4::CInstance, 116
sound4::cloudx1::CInstance, 127
sound4::cloudx1::dyn::CInstance, 138
GetParameter
sound4::cloudx1::dyn::CDynLib, 99
GetPossibleChunkSizeInFrames
sound4::cloudx1::dyn::CDynLib, 99
sound4::CProcessor, 158
GetPrefixSymbol
sound4::CProcessor, 158
GetProcessChunkFrames
sound4::cloudx1::dyn::CDynLib, 99
GetSampleRate
sound4::cloudx1::dyn::CDynLib, 99
sound4::CProcessor, 158
GetSymbol
sound4::cloudx1::dyn::helper::CDynLoader, 107
sound4::helper::CDynLoader, 110
GetSymbol_safe
sound4::cloudx1::dyn::helper::CDynLoader, 107
sound4::helper::CDynLoader, 110
GetThisLibraryPath
sound4::cloudx1::dyn::helper::CDynLoader, 107
sound4::helper::CDynLoader, 110
GetVersion
sound4::cloudx1::dyn::CDynLib, 99
sound4::CProcessor, 158
GetWebServerAppHealth
sound4::CInstance, 116
sound4::cloudx1::CInstance, 127
sound4::cloudx1::dyn::CInstance, 138
Global setup, 20
CLOUDX1_AUDIO_INPUT_COUNT, 21
CLOUDX1_AUDIO_OUTPUT_COUNT, 21
CLOUDX1_AUDIOFRAME_COUNT, 21
CLOUDX1_CHANNEL_COUNT, 21
cloudx1_GetAudioInputCount, 23
cloudx1_GetAudioOutputCount, 23
cloudx1_GetChannelCount, 23
cloudx1_GetChunkSizeInFrames, 24
cloudx1_GetPossibleChunkSizeInFrames, 24
cloudx1_GetSampleRate, 24
cloudx1_GetVersion, 25
cloudx1_loggerfn, 22
cloudx1_LogSeverity, 22
CLOUDX1_SAMPLE_RATE, 22
cloudx1_SetLoggerCallback, 25
cloudx1_SetLogSeverity, 25
error, 23
fatal, 23
info, 23
none, 23
SOUND4_GetProcessInfo, 26
verbose, 23
verbose2, 23
verbose3, 23
verbose4, 23
verbose5, 23
warning, 23

info
Global setup, 23
sound4, 75
InitProcess
sound4::cloudx1::dyn::CDynLib, 100
InitProcess2
sound4::cloudx1::dyn::CDynLib, 100
InitProcess3
sound4::cloudx1::dyn::CDynLib, 100
InputChannels
sound4::cloudx1, 78
sound4::cloudx1::dyn, 81
InputSampleSize
sound4::cloudx1, 78
sound4::cloudx1::dyn, 81
INVALID_FORMAT
Audio managing (any audio), 44
sound4cl_cdef.h, 261
IsOk
sound4::CInstance, 116
sound4::cloudx1::CInstance, 127
sound4::cloudx1::dyn::CDynLib, 94
sound4::cloudx1::dyn::CInstance, 138
sound4::cloudx1::dyn::helper::CDynLoader, 107
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 173
sound4::CProcessor, 159
sound4::helper::CDynLoader, 110
sound4::helper::DynFuncHolder< T >, 175
IsReadOnly
MyPresetManager, 177, 178
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 151

JSON syntax, 54
JSON syntax for internal setup, 57

Load
sound4::cloudx1::dyn::CDynLib, 94

sound4::cloudx1::dyn::helper::CDynLoader, 107
sound4::CProcessor, 159
sound4::helper::CDynLoader, 110
log_cb_t
sound4, 74
sound4::cloudx1, 77
sound4::cloudx1::dyn, 80
LogSeverity
sound4, 74
sound4::cloudx1, 77
sound4::cloudx1::dyn, 80

main
sound4.x1.cloud-pipe-cxx.cpp, 191
sound4.x1.cloud-pipe-dyn.cpp, 195
sound4.x1.cloud-pipe.cpp, 201
sound4cl-pipe.cpp, 237
Metadata Management, 37
cloudx1_FreeMetadataInfos, 37
cloudx1_GetMetadataInfos, 38
cloudx1_SetMetadata, 38
cloudx1_SetMetadataMulti, 39
MonoToStereo
sound4::cloudx1::dyn::CDynLib, 100
sound4::CProcessor, 159
MonoToStereo_Planar
sound4::cloudx1::dyn::CDynLib, 100
sound4::CProcessor, 160
MyLogger
sound4.x1.cloud-pipe-cxx.cpp, 192
sound4.x1.cloud-pipe-dyn.cpp, 196
sound4.x1.cloud-pipe.cpp, 201
MyPresetManager, 175
~MyPresetManager, 176
Exists, 177
GetAll, 177
IsReadOnly, 177, 178
MyPresetManager, 176
Read, 178
Remove, 178
Rename, 179
SetupPath, 179
Write, 179, 180

name
SOUND4_ProcessInfo, 188
sound4cl_SOUND4_ProcessInfo, 189
NewClient
sound4::CInstance, 116
sound4::cloudx1::CInstance, 127
sound4::cloudx1::dyn::CDynLib, 100
sound4::cloudx1::dyn::CInstance, 138
NewParameters
sound4::cloudx1::dyn::CDynLib, 101
none
Global setup, 23
sound4, 75

OnUpdateThreadStart
sound4::CInstance, 117
sound4::cloudx1::CInstance, 128
sound4::cloudx1::dyn::CInstance, 139
OnUpdateThreadStop
sound4::CInstance, 117
sound4::cloudx1::CInstance, 128
sound4::cloudx1::dyn::CInstance, 139
operator bool
sound4::cloudx1::dyn::helper::CDynLoader, 108
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 173
sound4::helper::CDynLoader, 111
sound4::helper::DynFuncHolder< T >, 175
operator T*
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 173
sound4::helper::DynFuncHolder< T >, 175
operator=
sound4::CInstance, 117
sound4::CInstance::CClient, 87
sound4::cloudx1::CInstance, 128
sound4::cloudx1::CInstance::CClient, 89
sound4::cloudx1::dyn::CInstance, 139
sound4::cloudx1::dyn::CInstance::CClient, 91
sound4::cloudx1::dyn::helper::CDynLoader, 108
sound4::helper::CDynLoader, 111
OutputChannels
sound4::cloudx1, 78
sound4::cloudx1::dyn, 81
OutputSampleSize
sound4::cloudx1, 79
sound4::cloudx1::dyn, 81

prefix
SOUND4_ProcessInfo, 188
sound4cl_SOUND4_ProcessInfo, 190
PresetChange_Kind
sound4, 74
sound4::cloudx1, 77
sound4::cloudx1::dyn, 80
PresetManager_InformChange
sound4::CInstance, 117
sound4::cloudx1::CInstance, 128
sound4::cloudx1::dyn::CDynLib, 101
sound4::cloudx1::dyn::CInstance, 139
Process creation/destruction, 26
cloudx1_ExitProcess, 27
cloudx1_FreeParameters, 27
cloudx1_FreeParameterValue, 27
cloudx1_GetParameter, 27
cloudx1_GetProcessChunkFrames, 28
cloudx1_InitProcess, 28
cloudx1_InitProcess2, 28
cloudx1_InitProcess3, 29
cloudx1_NewParameters, 30
cloudx1_SetParameter, 30
cloudx1_TerminateProcess, 31
ProcessAnyAudio
sound4::CInstance, 118

sound4::cloudx1::CInstance, 129
sound4::cloudx1::dyn::CInstance, 140
ProcessAudio
sound4::CInstance, 118
sound4::cloudx1::CInstance, 129
sound4::cloudx1::dyn::CDynLib, 101
sound4::cloudx1::dyn::CInstance, 140
ProcessAudio_Planar
sound4::CInstance, 118
sound4::cloudx1::CInstance, 129
sound4::cloudx1::dyn::CDynLib, 101
sound4::cloudx1::dyn::CInstance, 140
ProcessJson
sound4::CInstance::CClient, 87
sound4::cloudx1::CInstance::CClient, 89
sound4::cloudx1::dyn::CDynLib, 101
sound4::cloudx1::dyn::CInstance::CClient, 91
ProcessLoop
sound4.x1.cloud-pipe-cxx.cpp, 192
sound4.x1.cloud-pipe-dyn.cpp, 196
sound4.x1.cloud-pipe.cpp, 202
sound4cl-pipe.cpp, 237

Read
MyPresetManager, 178
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 148
sound4::CPresetLoader, 151
Remove
MyPresetManager, 178
sound4::cloudx1::CPresetLoader, 146
sound4::cloudx1::dyn::CPresetLoader, 149
sound4::CPresetLoader, 151
Rename
MyPresetManager, 179
sound4::cloudx1::CPresetLoader, 147
sound4::cloudx1::dyn::CPresetLoader, 149
sound4::CPresetLoader, 151

S16_BE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S16_LE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S16_NATIVE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S24_BE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S24_LE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S24_NATIVE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S32_BE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S32_LE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
S32_NATIVE
Audio managing (any audio), 44
sound4cl_cdef.h, 261
Sample using C interface, 17
Sample using C++ interface wrapper, 66
Sample using C++ interface wrapper with dynamic library loading, 67
Sample using Generic C++ interface (multi-process compatible), 69
SAMPLE_RATE
sound4.x1.cloud-pipe.cpp, 201
SampleFormat
sound4::cloudx1, 78
sound4::cloudx1::dyn, 80
SampleRate
sound4::cloudx1, 79
sound4::cloudx1::dyn, 81
SaveState
sound4::cloudx1::dyn::CDynLib, 101
SetBus
sound4::CInstance, 119
SetLoggerCallback
sound4::cloudx1::dyn::CDynLib, 102
sound4::CProcessor, 160
SetLogSeverity
sound4::cloudx1::dyn::CDynLib, 102
sound4::CProcessor, 160
SetMetadata
sound4::CInstance, 119
sound4::cloudx1::CInstance, 130
sound4::cloudx1::dyn::CDynLib, 102
sound4::cloudx1::dyn::CInstance, 141
SetMetadataMulti
sound4::CInstance, 120
sound4::cloudx1::CInstance, 130
sound4::cloudx1::dyn::CDynLib, 102
sound4::cloudx1::dyn::CInstance, 141
SetParam
sound4::CInstance, 120
sound4::cloudx1::CInstance, 131
sound4::cloudx1::dyn::CInstance, 142
SetParameter
sound4::cloudx1::dyn::CDynLib, 102
SetPresetManager
sound4::CInstance, 120
sound4::cloudx1::CInstance, 131
sound4::cloudx1::dyn::CDynLib, 102
sound4::cloudx1::dyn::CInstance, 142
SetupPath
MyPresetManager, 179
SetWebServerAppHealth
sound4::CInstance, 121
sound4::cloudx1::CInstance, 132
sound4::cloudx1::dyn::CInstance, 143

shortname
SOUND4_ProcessInfo, 188
sound4cl_SOUND4_ProcessInfo, 190
sound4, 73
error, 75
fatal, 75
info, 75
log_cb_t, 74
LogSeverity, 74
none, 75
PresetChange_Kind, 74
verbose, 75
verbose2, 75
verbose3, 76
verbose4, 76
verbose5, 76
warning, 76
sound4.x1.cloud-pipe-cxx.cpp, 191, 193
GetEnv, 191
main, 191
MyLogger, 192
ProcessLoop, 192
Usage, 192
sound4.x1.cloud-pipe-dyn.cpp, 194, 197
GetEnv, 195
main, 195
MyLogger, 196
ProcessLoop, 196
Usage, 196
sound4.x1.cloud-pipe.cpp, 200, 202
CHANNEL_COUNT, 200
CHUNK_SIZE, 201
main, 201
MyLogger, 201
ProcessLoop, 202
SAMPLE_RATE, 201
Usage, 202
sound4.x1.cloud.h, 205, 209
CLOUDX1_DECL, 208
CLOUDX1_HAS_CLOUDBUS, 208
CLOUDX1_HAS_WEBSERVER, 208
sound4.x1.cloud.hpp, 213, 214
fs_strdup, 214
sound4.x1.cloud_dyn.hpp, 221, 223
fs_strdup, 223
sound4::CBus, 85
~CBus, 85
CBus, 85
Get, 86
sound4::CInstance, 111
~CInstance, 113
AddAudio, 113
CInstance, 113
Create, 113
GetAudio, 114
GetBufferIn, 114
GetBufferOut, 114
GetChunkFrames, 115
GetEstimatedDelay, 115
GetMetadataInfos, 115
GetParam, 116
GetWebServerAppHealth, 116
IsOk, 116
NewClient, 116
OnUpdateThreadStart, 117
OnUpdateThreadStop, 117
operator=, 117
PresetManager_InformChange, 117
ProcessAnyAudio, 118
ProcessAudio, 118
ProcessAudio_Planar, 118
SetBus, 119
SetMetadata, 119
SetMetadataMulti, 120
SetParam, 120
SetPresetManager, 120
SetWebServerAppHealth, 121
StartWebServer, 121
Stop, 121
StopWebServer, 122
Terminate, 122
sound4::CInstance::CClient, 86
~CClient, 87
CClient, 87
operator=, 87
ProcessJson, 87
sound4::cloudx1, 76
ChunkFrames, 78
InputChannels, 78
InputSampleSize, 78
log_cb_t, 77
LogSeverity, 77
OutputChannels, 78
OutputSampleSize, 79
PresetChange_Kind, 77
SampleFormat, 78
SampleRate, 79
sound4::cloudx1::CInstance, 122
~CInstance, 124
AddAudio, 124
CInstance, 124
Create, 124
GetAudio, 125
GetBufferIn, 125
GetBufferOut, 125
GetChunkFrames, 126
GetEstimatedDelay, 126
GetMetadataInfos, 126
GetParam, 127
GetWebServerAppHealth, 127
IsOk, 127
NewClient, 127
OnUpdateThreadStart, 128
OnUpdateThreadStop, 128
operator=, 128
PresetManager_InformChange, 128

ProcessAnyAudio, 129
ProcessAudio, 129
ProcessAudio_Planar, 129
SetMetadata, 130
SetMetadataMulti, 130
SetParam, 131
SetPresetManager, 131
SetWebServerAppHealth, 132
StartWebServer, 132
Stop, 132
StopWebServer, 132
Terminate, 133
sound4::cloudx1::CInstance::CClient, 88
~CClient, 89
CClient, 89
operator=, 89
ProcessJson, 89
sound4::cloudx1::CPresetLoader, 145
~CPresetLoader, 146
CPresetLoader, 146
Exists, 146
GetAll, 146
IsReadOnly, 146
Read, 146
Remove, 146
Rename, 147
Write, 147
sound4::cloudx1::dyn, 79
ChunkFrames, 81
InputChannels, 81
InputSampleSize, 81
log_cb_t, 80
LogSeverity, 80
OutputChannels, 81
OutputSampleSize, 81
PresetChange_Kind, 80
SampleFormat, 80
SampleRate, 81
sound4::cloudx1::dyn::CDynLib, 92
AddAudio, 94
AddPadAudio, 95
AudioConvertFrom, 95
AudioConvertTo, 95
AudioMonoFromLiveStereo, 95
AudioMonoToLiveStereo, 95
CDynLib, 94
DeleteClient, 95
ExitProcess, 96
FreeJsonAnswer, 96
FreeMetadataInfos, 96
FreeParameters, 96
FreeParameterValue, 96
GetAudio, 96
GetAudioInputCount, 97
GetAudioOutputCount, 97
GetBufferIn, 97
GetBufferOut, 97
GetBytesFromFormat, 97
GetChannelCount, 97
GetChunkSizeInFrames, 98
GetEstimatedDelay, 98
GetFormatFromName, 98
GetFormatName, 98
GetMaxPacketFrame, 98
GetMetadataInfos, 98
GetOutputCount, 99
GetParameter, 99
GetPossibleChunkSizeInFrames, 99
GetProcessChunkFrames, 99
GetSampleRate, 99
GetVersion, 99
InitProcess, 100
InitProcess2, 100
InitProcess3, 100
IsOk, 94
Load, 94
MonoToStereo, 100
MonoToStereo_Planar, 100
NewClient, 100
NewParameters, 101
PresetManager_InformChange, 101
ProcessAudio, 101
ProcessAudio_Planar, 101
ProcessJson, 101
SaveState, 101
SetLoggerCallback, 102
SetLogSeverity, 102
SetMetadata, 102
SetMetadataMulti, 102
SetParameter, 102
SetPresetManager, 102
StartUpdateThread, 103
StereoToMono, 103
StereoToMono_Planar, 103
StopUpdateThread, 103
TerminateProcess, 103
WaitUpdateThreadReady, 103
Webserver, 104
Webserver_FreeString, 104
Webserver_GetAppHealth, 104
Webserver_SetAppHealth, 104
Webserver_Status, 104
Webserver_Stop, 104
Webserver_tcp, 105
Webserver_tcp2, 105
sound4::cloudx1::dyn::CInstance, 133
~CInstance, 135
AddAudio, 135
CInstance, 134, 135
Create, 135
GetAudio, 136
GetBufferIn, 136
GetBufferOut, 136
GetChunkFrames, 137
GetEstimatedDelay, 137
GetMetadataInfos, 137

GetParam, 138
GetWebServerAppHealth, 138
IsOk, 138
NewClient, 138
OnUpdateThreadStart, 139
OnUpdateThreadStop, 139
operator=, 139
PresetManager_InformChange, 139
ProcessAnyAudio, 140
ProcessAudio, 140
ProcessAudio_Planar, 140
SetMetadata, 141
SetMetadataMulti, 141
SetParam, 142
SetPresetManager, 142
SetWebServerAppHealth, 143
StartWebServer, 143
Stop, 143
StopWebServer, 143
Terminate, 144
sound4::cloudx1::dyn::CInstance::CClient, 90
~CClient, 91
CClient, 90, 91
operator=, 91
ProcessJson, 91
sound4::cloudx1::dyn::CPresetLoader, 147
~CPresetLoader, 148
CPresetLoader, 148
Exists, 148
GetAll, 148
IsReadOnly, 148
Read, 148
Remove, 149
Rename, 149
Write, 149
sound4::cloudx1::dyn::helper, 82
DynFunc_t, 82
DynLib_t, 82
sound4::cloudx1::dyn::helper::CDynLoader, 105
~CDynLoader, 106
CDynLoader, 106
Close, 106
GetSymbol, 107
GetSymbol_safe, 107
GetThisLibraryPath, 107
IsOk, 107
Load, 107
operator bool, 108
operator=, 108
sound4::cloudx1::dyn::helper::DynFuncHolder< T >, 172
DynFuncHolder, 173
IsOk, 173
operator bool, 173
operator T*, 173
sound4::cloudx1::dyn::helper::SampleFormat< float >, 182
format, 182
sound4::cloudx1::dyn::helper::SampleFormat< int16_t >, 183
format, 184
sound4::cloudx1::dyn::helper::SampleFormat< int32_t >, 185
format, 186
sound4::cloudx1::dyn::helper::SampleFormat< T >, 180
sound4::cloudx1::helper, 83
sound4::cloudx1::helper::SampleFormat< float >, 182
format, 183
sound4::cloudx1::helper::SampleFormat< int16_t >, 184
format, 184
sound4::cloudx1::helper::SampleFormat< int32_t >, 186
format, 186
sound4::cloudx1::helper::SampleFormat< T >, 181
sound4::CPresetLoader, 149
~CPresetLoader, 150
CPresetLoader, 150
Exists, 150
GetAll, 150
IsReadOnly, 151
Read, 151
Remove, 151
Rename, 151
Write, 151
sound4::CProcessor, 152
AudioConvertFrom, 154
AudioConvertTo, 154
AudioMonoFromLiveStereo, 155
AudioMonoToLiveStereo, 155
CBus, 161
CInstance, 161
CProcessor, 154
fnAddAudio, 161
fnAddPadAudio, 161
fnAudioConvertFrom, 161
fnAudioConvertTo, 162
fnAudioMonoFromLiveStereo, 162
fnAudioMonoToLiveStereo, 162
fnDeleteClient, 162
fnExitProcess, 162
fnFreeBus, 162
fnFreeJsonAnswer, 163
fnFreeMetadataInfos, 163
fnFreeParameters, 163
fnFreeParameterValue, 163
fnGetAudio, 163
fnGetAudioInputCount, 163
fnGetAudioOutputCount, 164
fnGetBufferIn, 164
fnGetBufferOut, 164
fnGetBytesFromFormat, 164
fnGetChannelCount, 164
fnGetChunkSizeInFrames, 164
fnGetEstimatedDelay, 165

fnGetFormatFromName, 165
fnGetFormatName, 165
fnGetMaxPacketFrame, 165
fnGetMetadataInfos, 165
fnGetOutputCount, 165
fnGetParameter, 166
fnGetPossibleChunkSizeInFrames, 166
fnGetProcessChunkFrames, 166
fnGetSampleRate, 166
fnGetVersion, 166
fnInitProcess, 166
fnInitProcess2, 167
fnInitProcess3, 167
fnMonoToStereo, 167
fnMonoToStereo_Planar, 167
fnNewBus, 167
fnNewClient, 167
fnNewParameters, 168
fnPresetManager_InformChange, 168
fnProcessAudio, 168
fnProcessAudio_Planar, 168
fnProcessJson, 168
fnSaveState, 168
fnSetInstanceBus, 169
fnSetLoggerCallback, 169
fnSetLogSeverity, 169
fnSetMetadata, 169
fnSetMetadataMulti, 169
fnSetParameter, 169
fnSetPresetManager, 170
fnStartUpdateThread, 170
fnStereoToMono, 170
fnStereoToMono_Planar, 170
fnStopUpdateThread, 170
fnTerminateProcess, 170
fnWaitUpdateThreadReady, 171
fnWebserver, 171
fnWebserver_FreeString, 171
fnWebserver_GetAppHealth, 171
fnWebserver_SetAppHealth, 171
fnWebserver_Status, 171
fnWebserver_Stop, 172
fnWebserver_tcp, 172
fnWebserver_tcp2, 172
GetAudioInputCount, 155
GetAudioOutputCount, 155
GetBytesFromFormat, 156
GetChannelCount, 156
GetChunkSizeInFrames, 156
GetFormatFromName, 157
GetFormatName, 157
GetPossibleChunkSizeInFrames, 158
GetPrefixSymbol, 158
GetSampleRate, 158
GetVersion, 158
IsOk, 159
Load, 159
MonoToStereo, 159
MonoToStereo_Planar, 160
SetLoggerCallback, 160
SetLogSeverity, 160
StereoToMono, 160
StereoToMono_Planar, 160
sound4::helper, 83
DynFunc_t, 84
DynLib_t, 84
sound4::helper::CDynLoader, 108
~CDynLoader, 109
CDynLoader, 109
Close, 109
GetSymbol, 110
GetSymbol_safe, 110
GetThisLibraryPath, 110
IsOk, 110
Load, 110
operator bool, 111
operator=, 111
sound4::helper::DynFuncHolder< T >, 174
DynFuncHolder, 174
IsOk, 175
operator bool, 175
operator T*, 175
sound4::helper::SampleFormat< float >, 183
format, 183
sound4::helper::SampleFormat< int16_t >, 185
format, 185
sound4::helper::SampleFormat< int32_t >, 187
format, 187
sound4::helper::SampleFormat< T >, 181
SOUND4_GetProcessInfo
Global setup, 26
SOUND4_INVALID_WEBSERVER_ID
sound4cl_cdef.h, 258
Web server, 61
SOUND4_ProcessInfo, 187
name, 188
prefix, 188
shortname, 188
struct_size, 188
version, 189
SOUND4_WEBSERVER_HTTP_OK
sound4cl_cdef.h, 258
Web server, 61
SOUND4_WEBSERVER_HTTPS_OK
sound4cl_cdef.h, 258
Web server, 62
sound4cl-pipe.cpp, 236, 238
GetEnv, 237
main, 237
ProcessLoop, 237
sound4cl.hpp, 241, 242
sound4cl_AddAudio
sound4cl_cdef.h, 261
sound4cl_AddPadAudio
sound4cl_cdef.h, 261
sound4cl_AudioConvertFrom

sound4cl_cdef.h, 261
sound4cl_AudioConvertTo
sound4cl_cdef.h, 262
sound4cl_AudioMonoFromLiveStereo
sound4cl_cdef.h, 262
sound4cl_AudioMonoToLiveStereo
sound4cl_cdef.h, 262
sound4cl_cdef.h, 255, 272
change_kind_created, 260
change_kind_deleted, 260
change_kind_modified, 260
F32_BE, 261
F32_LE, 261
F32_NATIVE, 261
fs_char, 258
fs_strdup, 257
INVALID_FORMAT, 261
S16_BE, 261
S16_LE, 261
S16_NATIVE, 261
S24_BE, 261
S24_LE, 261
S24_NATIVE, 261
S32_BE, 261
S32_LE, 261
S32_NATIVE, 261
SOUND4_INVALID_WEBSERVER_ID, 258
SOUND4_WEBSERVER_HTTP_OK, 258
SOUND4_WEBSERVER_HTTPS_OK, 258
sound4cl_AddAudio, 261
sound4cl_AddPadAudio, 261
sound4cl_AudioConvertFrom, 261
sound4cl_AudioConvertTo, 262
sound4cl_AudioMonoFromLiveStereo, 262
sound4cl_AudioMonoToLiveStereo, 262
sound4cl_DeleteClient, 262
sound4cl_error, 260
sound4cl_ExitProcess, 262
sound4cl_fatal, 260
sound4cl_FreeBus, 262
sound4cl_FreeJsonAnswer, 263
sound4cl_FreeMetadataInfos, 263
sound4cl_FreeParameters, 263
sound4cl_FreeParameterValue, 263
sound4cl_GetAudio, 263
sound4cl_GetAudioInputCount, 263
sound4cl_GetAudioOutputCount, 264
sound4cl_GetBufferIn, 264
sound4cl_GetBufferOut, 264
sound4cl_GetBytesFromFormat, 264
sound4cl_GetChannelCount, 264
sound4cl_GetChunkSizeInFrames, 264
sound4cl_GetEstimatedDelay, 264
sound4cl_GetFormatFromName, 265
sound4cl_GetFormatName, 265
sound4cl_GetMaxPacketFrame, 265
sound4cl_GetMetadataInfos, 265
sound4cl_GetOutputCount, 265
sound4cl_GetParameter, 265
sound4cl_GetPossibleChunkSizeInFrames, 265
sound4cl_GetProcessChunkFrames, 266
sound4cl_GetSampleRate, 266
sound4cl_GetVersion, 266
sound4cl_info, 260
sound4cl_InitProcess, 266
sound4cl_InitProcess2, 266
sound4cl_InitProcess3, 266
sound4cl_loggerfn, 258
sound4cl_LogSeverity, 260
sound4cl_MonoToStereo, 267
sound4cl_MonoToStereo_Planar, 267
sound4cl_NewBus, 267
sound4cl_NewClient, 267
sound4cl_NewParameters, 267
sound4cl_none, 260
sound4cl_PresetChange_Kind, 260
sound4cl_PresetManager_InformChange, 267
sound4cl_ProcessAudio, 268
sound4cl_ProcessAudio_Planar, 268
sound4cl_ProcessJson, 268
sound4cl_SampleFormat, 261
sound4cl_SaveState, 268
sound4cl_SetInstanceBus, 268
sound4cl_SetLoggerCallback, 268
sound4cl_SetLogSeverity, 269
sound4cl_SetMetadata, 269
sound4cl_SetMetadataMulti, 269
sound4cl_SetParameter, 269
sound4cl_SetPresetManager, 269
sound4cl_SOUND4_GetProcessInfo, 269
sound4cl_StartUpdateThread, 270
sound4cl_StereoToMono, 270
sound4cl_StereoToMono_Planar, 270
sound4cl_StopUpdateThread, 270
sound4cl_storage_exists, 258
sound4cl_storage_getall, 259
sound4cl_storage_getall_free, 259
sound4cl_storage_reader, 259
sound4cl_storage_reader_free, 259
sound4cl_storage_remove, 259
sound4cl_storage_rename, 259
sound4cl_storage_writer, 260
sound4cl_TerminateProcess, 270
sound4cl_verbose, 260
sound4cl_verbose2, 260
sound4cl_verbose3, 260
sound4cl_verbose4, 260
sound4cl_verbose5, 260
sound4cl_WaitUpdateThreadReady, 270
sound4cl_warning, 260
sound4cl_Webserver, 271
sound4cl_Webserver_FreeString, 271
sound4cl_Webserver_GetAppHealth, 271
sound4cl_Webserver_SetAppHealth, 271
sound4cl_Webserver_Status, 271
sound4cl_Webserver_Stop, 271

sound4cl_Webserver_tcp, 272
sound4cl_Webserver_tcp2, 272
sound4cl_DeleteClient
sound4cl_cdef.h, 262
sound4cl_error
sound4cl_cdef.h, 260
sound4cl_ExitProcess
sound4cl_cdef.h, 262
sound4cl_fatal
sound4cl_cdef.h, 260
sound4cl_FreeBus
sound4cl_cdef.h, 262
sound4cl_FreeJsonAnswer
sound4cl_cdef.h, 263
sound4cl_FreeMetadataInfos
sound4cl_cdef.h, 263
sound4cl_FreeParameters
sound4cl_cdef.h, 263
sound4cl_FreeParameterValue
sound4cl_cdef.h, 263
sound4cl_GetAudio
sound4cl_cdef.h, 263
sound4cl_GetAudioInputCount
sound4cl_cdef.h, 263
sound4cl_GetAudioOutputCount
sound4cl_cdef.h, 264
sound4cl_GetBufferIn
sound4cl_cdef.h, 264
sound4cl_GetBufferOut
sound4cl_cdef.h, 264
sound4cl_GetBytesFromFormat
sound4cl_cdef.h, 264
sound4cl_GetChannelCount
sound4cl_cdef.h, 264
sound4cl_GetChunkSizeInFrames
sound4cl_cdef.h, 264
sound4cl_GetEstimatedDelay
sound4cl_cdef.h, 264
sound4cl_GetFormatFromName
sound4cl_cdef.h, 265
sound4cl_GetFormatName
sound4cl_cdef.h, 265
sound4cl_GetMaxPacketFrame
sound4cl_cdef.h, 265
sound4cl_GetMetadataInfos
sound4cl_cdef.h, 265
sound4cl_GetOutputCount
sound4cl_cdef.h, 265
sound4cl_GetParameter
sound4cl_cdef.h, 265
sound4cl_GetPossibleChunkSizeInFrames
sound4cl_cdef.h, 265
sound4cl_GetProcessChunkFrames
sound4cl_cdef.h, 266
sound4cl_GetSampleRate
sound4cl_cdef.h, 266
sound4cl_GetVersion
sound4cl_cdef.h, 266
sound4cl_info
sound4cl_cdef.h, 260
sound4cl_InitProcess
sound4cl_cdef.h, 266
sound4cl_InitProcess2
sound4cl_cdef.h, 266
sound4cl_InitProcess3
sound4cl_cdef.h, 266
sound4cl_loggerfn
sound4cl_cdef.h, 258
sound4cl_LogSeverity
sound4cl_cdef.h, 260
sound4cl_MonoToStereo
sound4cl_cdef.h, 267
sound4cl_MonoToStereo_Planar
sound4cl_cdef.h, 267
sound4cl_NewBus
sound4cl_cdef.h, 267
sound4cl_NewClient
sound4cl_cdef.h, 267
sound4cl_NewParameters
sound4cl_cdef.h, 267
sound4cl_none
sound4cl_cdef.h, 260
sound4cl_PresetChange_Kind
sound4cl_cdef.h, 260
sound4cl_PresetManager_InformChange
sound4cl_cdef.h, 267
sound4cl_ProcessAudio
sound4cl_cdef.h, 268
sound4cl_ProcessAudio_Planar
sound4cl_cdef.h, 268
sound4cl_ProcessJson
sound4cl_cdef.h, 268
sound4cl_SampleFormat
sound4cl_cdef.h, 261
sound4cl_SaveState
sound4cl_cdef.h, 268
sound4cl_SetInstanceBus
sound4cl_cdef.h, 268
sound4cl_SetLoggerCallback
sound4cl_cdef.h, 268
sound4cl_SetLogSeverity
sound4cl_cdef.h, 269
sound4cl_SetMetadata
sound4cl_cdef.h, 269
sound4cl_SetMetadataMulti
sound4cl_cdef.h, 269
sound4cl_SetParameter
sound4cl_cdef.h, 269
sound4cl_SetPresetManager
sound4cl_cdef.h, 269
sound4cl_SOUND4_GetProcessInfo
sound4cl_cdef.h, 269
sound4cl_SOUND4_ProcessInfo, 189
name, 189
prefix, 190
shortname, 190

struct_size, 190
version, 190
sound4cl_StartUpdateThread
sound4cl_cdef.h, 270
sound4cl_StereoToMono
sound4cl_cdef.h, 270
sound4cl_StereoToMono_Planar
sound4cl_cdef.h, 270
sound4cl_StopUpdateThread
sound4cl_cdef.h, 270
sound4cl_storage_exists
sound4cl_cdef.h, 258
sound4cl_storage_getall
sound4cl_cdef.h, 259
sound4cl_storage_getall_free
sound4cl_cdef.h, 259
sound4cl_storage_reader
sound4cl_cdef.h, 259
sound4cl_storage_reader_free
sound4cl_cdef.h, 259
sound4cl_storage_remove
sound4cl_cdef.h, 259
sound4cl_storage_rename
sound4cl_cdef.h, 259
sound4cl_storage_writer
sound4cl_cdef.h, 260
sound4cl_TerminateProcess
sound4cl_cdef.h, 270
sound4cl_verbose
sound4cl_cdef.h, 260
sound4cl_verbose2
sound4cl_cdef.h, 260
sound4cl_verbose3
sound4cl_cdef.h, 260
sound4cl_verbose4
sound4cl_cdef.h, 260
sound4cl_verbose5
sound4cl_cdef.h, 260
sound4cl_WaitUpdateThreadReady
sound4cl_cdef.h, 270
sound4cl_warning
sound4cl_cdef.h, 260
sound4cl_Webserver
sound4cl_cdef.h, 271
sound4cl_Webserver_FreeString
sound4cl_cdef.h, 271
sound4cl_Webserver_GetAppHealth
sound4cl_cdef.h, 271
sound4cl_Webserver_SetAppHealth
sound4cl_cdef.h, 271
sound4cl_Webserver_Status
sound4cl_cdef.h, 271
sound4cl_Webserver_Stop
sound4cl_cdef.h, 271
sound4cl_Webserver_tcp
sound4cl_cdef.h, 272
sound4cl_Webserver_tcp2
sound4cl_cdef.h, 272
StartUpdateThread
sound4::cloudx1::dyn::CDynLib, 103
StartWebServer
sound4::CInstance, 121
sound4::cloudx1::CInstance, 132
sound4::cloudx1::dyn::CInstance, 143
std, 84
StereoToMono
sound4::cloudx1::dyn::CDynLib, 103
sound4::CProcessor, 160
StereoToMono_Planar
sound4::cloudx1::dyn::CDynLib, 103
sound4::CProcessor, 160
Stop
sound4::CInstance, 121
sound4::cloudx1::CInstance, 132
sound4::cloudx1::dyn::CInstance, 143
StopUpdateThread
sound4::cloudx1::dyn::CDynLib, 103
StopWebServer
sound4::CInstance, 122
sound4::cloudx1::CInstance, 132
sound4::cloudx1::dyn::CInstance, 143
struct_size
SOUND4_ProcessInfo, 188
sound4cl_SOUND4_ProcessInfo, 190

Terminate
sound4::CInstance, 122
sound4::cloudx1::CInstance, 133
sound4::cloudx1::dyn::CInstance, 144
TerminateProcess
sound4::cloudx1::dyn::CDynLib, 103

Update Management, 39
cloudx1_StartUpdateThread, 39
cloudx1_StopUpdateThread, 40
cloudx1_WaitUpdateThreadReady, 40
Usage
sound4.x1.cloud-pipe-cxx.cpp, 192
sound4.x1.cloud-pipe-dyn.cpp, 196
sound4.x1.cloud-pipe.cpp, 202

verbose
Global setup, 23
sound4, 75
verbose2
Global setup, 23
sound4, 75
verbose3
Global setup, 23
sound4, 76
verbose4
Global setup, 23
sound4, 76
verbose5
Global setup, 23
sound4, 76
version

SOUND4_ProcessInfo, 189
sound4cl_SOUND4_ProcessInfo, 190

WaitUpdateThreadReady
sound4::cloudx1::dyn::CDynLib, 103
warning
Global setup, 23
sound4, 76
Web server, 61
cloudx1_Webserver, 62
cloudx1_Webserver_FreeString, 63
cloudx1_Webserver_GetAppHealth, 63
cloudx1_Webserver_SetAppHealth, 63
cloudx1_Webserver_Status, 64
cloudx1_Webserver_Stop, 64
cloudx1_Webserver_tcp, 65
cloudx1_Webserver_tcp2, 65
SOUND4_INVALID_WEBSERVER_ID, 61
SOUND4_WEBSERVER_HTTP_OK, 61
SOUND4_WEBSERVER_HTTPS_OK, 62
Webserver
sound4::cloudx1::dyn::CDynLib, 104
Webserver_FreeString
sound4::cloudx1::dyn::CDynLib, 104
Webserver_GetAppHealth
sound4::cloudx1::dyn::CDynLib, 104
Webserver_SetAppHealth
sound4::cloudx1::dyn::CDynLib, 104
Webserver_Status
sound4::cloudx1::dyn::CDynLib, 104
Webserver_Stop
sound4::cloudx1::dyn::CDynLib, 104
Webserver_tcp
sound4::cloudx1::dyn::CDynLib, 105
Webserver_tcp2
sound4::cloudx1::dyn::CDynLib, 105
Write
MyPresetManager, 179, 180
sound4::cloudx1::CPresetLoader, 147
sound4::cloudx1::dyn::CPresetLoader, 149
sound4::CPresetLoader, 151