AIRENCE

Airence C Library v1.2 for Windows

"Let the Airence control your Radio Automation Software!"

D&R Electronica Weesp BV
Rijnkade 15B
1382GS Weesp
The Netherlands
Phone: +31 (0)294-418014
Fax: +31 (0)294-416987
Website: http://www.d-r.nl
E-mail: info@d-r.nl

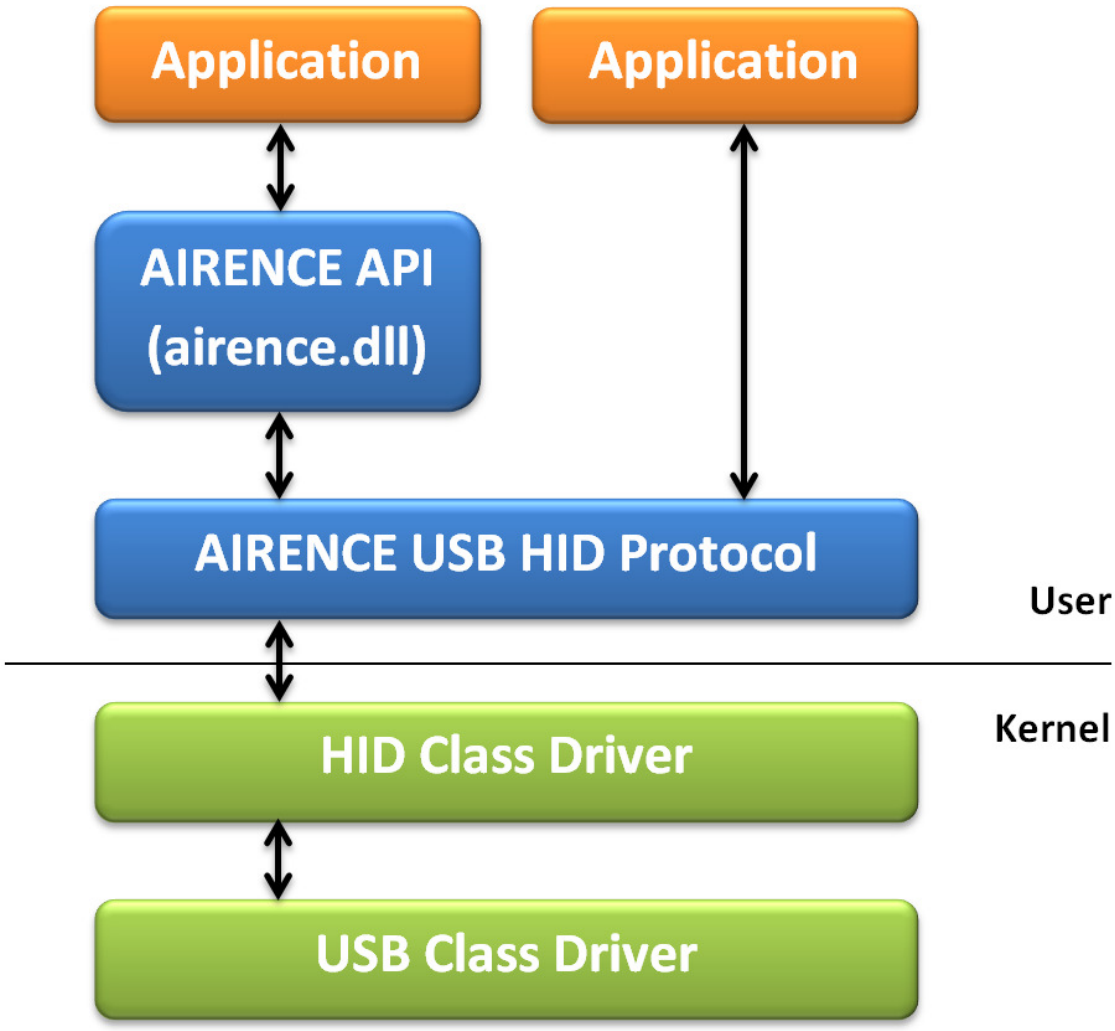
Document Version 1.2 - 2014-09-16

1 Table of contents

2 Introduction

The Airence C Library provides a simple API to access all the control signals available on the Airence mixer. When integrating the library with playout software one is able to control this software with the control signals of the mixer.

The library is intended to be used with any programming language available on Windows which supports loading a DLL (Dynamic Link Library). The name of the library is *airence.dll*. It is not necessary to use this library to be able to communicate with the Airence mixer. Directly implementing the AIRENCE USB HID PROTOCOL in your application is also possible.



3 Using the library

3.1 C Example

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#include "airence.h"

void myCSC_CB( int nControlsignal, bool state, void *data );
void myEC_CB( int direction, unsigned char abs_value, void *data );


int _tmain(int argc, _TCHAR* argv[])
{
    // Connect to Airence
    if( !airenceOpen() )
    {
        // Get library version info
        printf("%s\n",airenceGetLibraryVersion( NULL, NULL ));

        // Get firmware version info
        printf("%s\n\n",airenceGetFirmwareVersion( NULL, NULL ));

        // Register callbacks
        airenceSetControlSignalChangeCB( myCSC_CB, NULL );
        airenceSetEncoderChangeCB( myEC_CB, NULL );

        airenceSetLedBlink( SW_LED_1, RED, GREEN, NORMAL );
        airenceSetLed(SW_LED_4, GREEN);

        std::cin.get();
    }
    airenceClose();
    return 0;
}

void myCSC_CB( int nControlsignal, bool state, void *data )
{
    printf("Control: %d, State: %d\n", nControlsignal, state );
}

void myEC_CB( int direction, unsigned char abs_value, void *data )
{
    if( direction == 1 )
        printf("Encoder increment: %d\n", abs_value);
    if( direction == -1 )
        printf("Encoder decrement: %d\n", abs_value);
}
```

4 API Reference

4.1 Defines

4.1.1 Control Signals

```
#define LED_ALL                 0xFF
#define SW_LED_1                1
#define SW_LED_2                2
#define SW_LED_3                3
#define SW_LED_4                4
#define SW_LED_5                5
#define SW_LED_6                6
#define SW_LED_7                7
#define SW_LED_8                8
#define SW_LED_9                9
#define SW_LED_10               10
#define SW_LED_11               11
#define SW_LED_12               12
#define SW_LED_13               13
#define SW_LED_14               14
#define SW_LED_15               15
#define SW_LED_16               16
#define SW_LED_17               17
#define SW_LED_18               18
#define SW_LED_19               19
#define SW_LED_20               20
#define SW_LED_21               21
#define SW_LED_22               22
#define SW_LED_23               23
#define SW_LED_24               24
#define SW_ENCODER              25
#define SW_NONSTOP              26
#define SW_USB1_FADERSTART      27
#define SW_USB1_ON              28
#define SW_USB1_CUE             29
#define SW_USB2_FADERSTART      30
#define SW_USB2_ON              31
#define SW_USB2_CUE             32
#define SW_USB3_FADERSTART      33
#define SW_USB3_ON              34
#define SW_USB3_CUE             35
#define SW_USB4_FADERSTART      36
#define SW_USB4_ON              37
#define SW_USB4_CUE             38
```

In total there are 38 control signals which can be used with this library. These control signals are numbered in a logical way and can be used as argument in the functions *airenceSetLed()*, *airenceSetLedBlink()*, *airenceGetControlSignal()* and the *CSC_CB()* callback function.

4.2 Data types

4.2.1 LED Color

```
typedef enum {
        NONE,
        RED,
        GREEN,
        YELLOW
} colors_t;
```

The *colors_t* enum type is used as argument in the functions *airenceSetLed()* and *airenceSetLedBlink(),* specifying the color of a particular LED behind a switch of the control section.

4.2.2 LED Blink speed

```
typedef enum {
        SLOW,
        NORMAL,
        FAST
} blink_speed_t;
```

The *blink_speed_t* enum type is used as argument in the function *airenceSetLedBlink(),* specifying the blink speed of a particular LED behind a switch of the control section.

4.3 Functions

4.3.1 airenceOpen

int airenceOpen(void);

The function *airenceOpen()* must be called before any other function of this library may be used. The function will try to setup the communication with the Airence mixer and initializes the library. After initialization the initial states of the control signals are read and stored internally in the library. The LED's behind the control switches are turned off. On succes the function returns 0, or -1 otherwise.

4.3.2 airenceClose

int airenceClose(void);

When done using the library one needs to call this function to free the allocated memory and finalize the library. The LED's behind the control switches are turned off. On succes the function returns 0, or -1 otherwise.

4.3.3 airenceSetLed

int airenceSetLed(int lednr, colors_t color);

Set the specified LED *lednr* with *color*. One can choose a value for *lednr* in the range of 1~24. These LED's are the LED's behind the control switches. On succes the function returns 0, or -1 otherwise.

4.3.4 airenceSetLedBlink

int airenceSetLedBlink(int lednr, colors_t on, colors_t off, blink_speed_t speed);

Set the specified LED *lednr* in blink mode. The on/off-color and speed can be given to the function by the arguments. On succes the function returns 0, or -1 otherwise.

4.3.5 airenceGetControlSignal

int airenceGetControlSignal(int controlsignal, bool *state);

To retrieve the state of a single control signal this function must be called. The *controlsignal* argument can have a value in the range of 1~38 (see control signal defines in this document). The argument *state* is a pointer to a boolean variable. The variable itself will be set to TRUE(1) for press state, and FALSE(0) for release state. On succes the function returns 0, or -1 otherwise.

4.3.6 airenceGetRawControlData

int airenceGetRawControlData(unsigned char *data);

In order to retrieve the state of all the control signals this function must be called. Reserve a buffer of at least 6 bytes to store the data and give the pointer of that buffer as argument to this function. Every bit presents the state of a control signal, '1' for press and '0' for release state. The data will be presented as can be seen in the table below. On succes the function returns 0, or -1 otherwise.

data[]	Symbol	Value	Description
0	sw_8_1	-	bit 0: switch 1 (1=pressed, 0=released) | bit 7: switch 8
1	sw_16_9	-	bit 0: switch 9 (1=pressed, 0=released) | bit 7: switch 16
2	sw_24_17	-	bit 0: switch 17 (1=pressed, 0=released) | bit 7: switch 24
3	sw_enc_non	-	bit 0: encoder switch (1=pressed, 0=released) bit 1: non-stop switch (1=pressed, 0=released)
4	usb_2_1	-	bit 0: USB1 faderstart (1=up, 0=down) bit 1: USB1 ON (1=pressed, 0=released) bit 2: USB1 CUE (1=pressed, 0=released) bit 3: USB2 faderstart (1=up, 0=down) bit 4: USB2 ON (1=pressed, 0=released) bit 5: USB2 CUE (1=pressed, 0=released)
5	usb_4_3	-	bit 0: USB3 faderstart (1=up, 0=down) bit 1: USB3 ON (1=pressed, 0=released) bit 2: USB3 CUE (1=pressed, 0=released) bit 3: USB4 faderstart (1=up, 0=down) bit 4: USB4 ON (1=pressed, 0=released) bit 5: USB4 CUE (1=pressed, 0=released)

4.3.7 airenceGetLibraryVersion

char* airenceGetLibraryVersion(int* major, int* minor);

This function returns the version of this library. The *major* and *minor* arguments can be used to store the version, otherwise one has to fill in these arguments with the value NULL. Besides the version the build date of the library will be returned.

Example:

"Airence Library v1.0 – Jun 10 2013 (16:33:51) – D&R"

4.3.8 airenceGetFirmwareVersion

char* airenceGetFirmwareVersion(int* major, int* minor);

This function returns the version of the firmware present in the Airence mixer. The *major* and *minor* arguments can be used to store the version, otherwise one has to fill in these arguments with the value NULL.

Example:

"Airence Firmware v1.2 – D&R"

4.3.9 airenceSetControlSignalChangeCB

void airenceSetControlSignalChangeCB(void(*CSC_CB)(int controlsignal, bool state, void *data), void *data);

Sets the callback function for a control signal change event. If there occurs a state change in one of the control signals, this callback will be called. The *data* parameter when registering the callback will be provided as parameter in the callback function. The function returns void.

4.3.10 airenceClearControlSignalChangeCB

void airenceClearControlSignalChangeCB(void);

Clears the callback function for a control signal change event which was prior set with the *airenceSetControlSignalChangeCB()* function. The function returns void.

4.3.11 airenceSetEncoderChangeCB

void airenceSetEncoderChangeCB(void(*EC_CB)(int direction, unsigned char abs_value, void *data), void *data);

Sets the callback function for a encoder change event. If there occurs a state change in the push switch or increment/decrement of the encoder, this callback will be called. The *direction* arguments holds the value '1' for increment (clockwise) and '-1' for decrement (counterclockwise). Furthermore the absolute value of the encoder can be used and ranges from 0~255. The *data* parameter when registering the callback will be provided as parameter in the callback function.The function returns void.

4.3.12 airenceClearEncoderChangeCB

void airenceClearEncoderChangeCB(void);

Clears the callback function for an encoder change event which was prior set with the *airenceSetEncoderChangeCB()* function. The function returns void.

4.3.13 airenceSetReadMode

void airenceSetReadMode(int readmode);

```
/* <readmode>, timeout in milliseconds
        -1: Blocking
         0: Non-Blocking
        >0: Non-Blocking with timeout  */
```

With this function one can determine the read mode used by the functions *airenceOpen()*, *airenceGetLibraryVersion()* and *airenceGetFirmwareVersion()*. In blocking mode the above functions will block until data is received. In non-blocking mode the functions always return, with or without setting the readmode. If there is a readmode value set larger than zero, the calling function interpretes the readmode as a timeout value and will wait that time for receiving data from the Airence mixer.

4.3.14 airenceGetLastError

char* airenceGetLastError(void);

This function returns the last occurred error in the library.

4.3.15 airenceEnableLogging

void airenceEnableLogging(bool state);

Logging can be enabled by calling this function with the argument set to true. Calling the function with the argument false disables the logging. A txt-logfile will be created in the same folder whereas the DLL is located and the library will be used by an application. The logfile logs any kind of activity in the library, like events, function calls, or errors.

4.4 Callbacks

4.4.1 CSC_CB

void(*CSC_CB)(int controlsignal, bool state, void *data);

Callback function prototype for Control Signal Change event. The data parameter can be used as a reference to an object or other data provided when registering the callback. Function returns void.

4.4.2 EC_CB

void(*EC_CB)(int direction, unsigned char abs_value, void *data);

Callback function prototype for Encoder Change event. The data parameter can be used as a reference to an object or other data provided when registering the callback. Function returns void.