MLink 2.1v

Function Reference

March 2018, rev: 1.0
www.microdaq.org

Embedded Solutions

Table of contents

Introduction

The MLink library is a programming interface for programming analog input, analog output, digital I/O, on MicroDAQ DAQ devices. The library can be used on 32-bit and 64-bit Windows, Linux and MacOS operating systems. The user can use MLink library to create custom C/C++ application as well as C# and Java after creating an MLink wrapper to access library functions. Unique MLink features allow MicroDAQ DSP core management allowing the user to embed real-time processing on dedicated hardware with standard C/C++ desktop application.

Some of the presented examples in this document are written for Windows platform. To run it on other systems it has to be modified manually.

Download

https://github.com/microdaq/MLink/releases/tag/2.1.0v

mlink_connect

Connect with MicroDAQ device

Function prototype

```
int mlink_connect( const char *addr, uint16_t port, int *link );
```

Description

This function connects a host PC with MicroDAQ device at given IP address and port. The created connection uses TCP protocol to exchange data between host and MicroDAQ device. Default port number is 4343. On success, 0 is returned and a valid connection descriptor is stored in the variable pointed by *link* pointer. Connector descriptor shall be used for all MLink functions as a *link_fd* argument.

Arguments

- **addr:** MicroDAQ IP address
- **port:** port number (default: 4343)
- **link:** pointer to connection descriptor

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example connects to MicroDAQ device and turns on D1 built-in LED.

```
#include <stdio.h>
#include "MLink.h"
#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, link_fd;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_led_write(&link_fd, 1, 1);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_disconnect

Close connection with MicroDAQ

Function prototype

```
int mlink_disconnect( int link_fd );
```

Description

This function closes connection with MicroDAQ.

Arguments

- **link_fd:** valid connection descriptor

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example connects to MicroDAQ device and turns on D1 built-in LED and closes connection.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, link_fd;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_led_write(&link_fd, 1, 1);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_disconnect_all

Close all active connections with MicroDAQ

Function prototype

```
int mlink_disconnect_all( void );
```

Description

This function closes all active connections with MicroDAQ. It can be used to close active connection e.g. from previous MLink session.

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example connects to two MicroDAQ devices with different IP address.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, mdaq1_fd, mdaq2_fd;

        result = mlink_connect("10.10.1.1", 4343, &mdaq1_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_connect("10.10.1.2", 4343, &mdaq2_fd);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect_all();
        return 0;
}
```

mlink_error

Get MLink error description

Function prototype

```
char *mlink_error( int err );
```

Description

This function returns error description for the given MLink error code.

Arguments

- **err:** MLink error code

Return value

String containing MLink error code description.

Examples

This example connects to MicroDAQ device and turns on D1 built-in LED.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return
1;}

int main()
{
        int result, link_fd;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_led_write(&link_fd, 1, 1);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_fw_version

Get MicroDAQ firmware version

Function prototype

```
int mlink_fw_version(int *link_fd, int *major, int *minor, int *fix, int *build);
```

Description

This function reads MicroDAQ installed firmware version. The function read major, minor, fix and build number into variables pointed by *major, minor, fix, build.*

Arguments

- **link_fd:** valid connection descriptor
- **major:** pointer to int variable
- **minor:** pointer to int variable
- **fix:** pointer to int variable
- **build:** pointer to int variable

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example prints MicroDAQ firmware version.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, link_fd;
        int major, minor, fix, build;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_fw_version(&link_fd, &major, &minor, &fix, &build);
        if(result < 0)
                MLINK_ERROR(result);

        printf("MicroDAQ firmware: %d.%d.%d (build: %d)\n", major, minor, fix, build);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_lib_version

Get MLink library version

Function prototype

```
int mlink_lib_version(int *link_fd, int *major, int *minor, int *fix, int *build);
```

Description

This function reads MLink library version. The function read major, minor, fix and build number into variables pointed by *major, minor, fix, build.*

Arguments

- **link_fd:** valid connection descriptor
- **major:** pointer to int variable
- **minor:** pointer to int variable
- **fix:** pointer to int variable
- **build:** pointer to int variable

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example prints information of MLink library version.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, link_fd;
        int major, minor, fix, build;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_lib_version(&link_fd, &major, &minor, &fix, &build);
        if(result < 0)
                MLINK_ERROR(result);

        printf("MLink library: %d.%d.%d (build: %d)\n", major, minor, fix, build);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_hwid

Get MicroDAQ hardware ID

Function prototype

```
int mlink_hwid( int *link_fd, int *hwid );
```

Description

This function reads MicroDAQ hardware ID. Information is stored in a five element integer array. Array index description: 0 - MicroDAQ model, 1 - ADC type, 2 - DAC type, 3 - CPU type, 4 - storage size

Arguments

- **link_fd:** valid connection descriptor
- **hwid:** pointer to integer array (5 element)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example prints information of connected MicroDAQ device.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err) {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result, hw_id[5];

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_hwid(&link_fd, hw_id);
        if(result < 0)
                MLINK_ERROR(result);

        printf("MicroDAQ hardware info:\nModel:E%d, ADC:%d, DAC:%d, CPU:%d, Storage:%d\n",
                hw_id[0], hw_id[1], hw_id[2], hw_id[3], hw_id[4]);

        mlink_disconnect(link_fd);

        return 0;
}
```

mlink_dsp_run

Run DSP application on MicroDAQ

Function prototype

```
int mlink_dsp_run(int *link_fd,  const char *dsp_binary_path, double period);
```

Description

This function loads and starts application generated MicroDAQ DSP processor. The function can be used with DSP executables generated from Xcos model only. It loads binary file from path pointed by *dsp_binary_path.* The function can modify Xcos model step time setting by providing *period* argument. This way user can run generated DSP application with different model step time without a need to re-generated DSP executable. If *period* is equal -1 Xcos model step time setting will be not overwritten.

Limitation: This function can be used with MicroDAQ E1100 and E2000 devices only.

Arguments

- **link_fd:** valid connection descriptor
- **dsp_binary_path:** XCos generated DSP application path
- **period:** DSP application interval in seconds.

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example loads (from local directory) and executes **'blinkingled.out'** application on MicroDAQ DSP core. When loaded and started on DSP, it works independently and after 5 seconds application is terminated by *mlink_dsp_stop()* function call.
The example can be only executed with MicroDAQ E1100 and E2000 series. The Xcos DSP application **'blinkingled.out'** is located in **'MLink dsp examples\ mlink_dsp_run stop\'** directory attached to this document.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>

#define MLINK_ERROR(err) {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result, i;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
```

```
        if(result < 0)
                MLINK_ERROR(result);

        //Run with period 0.1 sec = 10 Hz
        result = mlink_dsp_run(&link_fd, "blinkingled.out", 0.1);
        if(result < 0)
                MLINK_ERROR(result);

        Sleep(5000);

        result = mlink_dsp_stop(&link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_dsp_signal_read

Read data from DSP application

Function prototype

```
int mlink_dsp_signal_read(int signal_id, int signal_size, double *data, int data_size,
int timeout);
```

Description

This function reads data from DSP application during its execution. The function reads signal data from application generated from Xcos model containing SIGNAL block.

Arguments

- **signal_id:** Xcos SIGNAL block ID
- **signal_size:** SIGNAL block data size
- **data:** pointer to data
- **data_size:** size of data.
- **timeout:** timeout in milliseconds

Return **value**

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example runs the **'signalmodel.out'** application generated from Xcos model on MicroDAQ DSP core. The DSP application sends data from STEP block through SIGNAL block (TCP protocol) to host application. The host application receives data from device using *mlink_dsp_signal_read()* function. The example can be only executed with MicroDAQ E1100 and E2000 series.

The example can be only executed with MicroDAQ E1100 and E2000 series. The Xcos DSP application **'signalmodel.out'** is located in **'MLink dsp examples\ mlink_dsp_signal_read\'** directory attached to this document.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                 (20)

int main()
{
        int result, link_fd, i;
        double data[DATA_SIZE];
        double period = 0.1; //10 Hz

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_dsp_run(&link_fd, "signalmodel.out", 0.1);
    if(result < 0)

                MLINK_ERROR(result);

        result = mlink_dsp_signal_read(1, 1, data, DATA_SIZE, 1000);
        if(result < 0)
                MLINK_ERROR(result);

        for(i = 0; i < DATA_SIZE; i++)
        {
                printf("%.1f sec, output: %.1f\n", i*period, data[i]);
        }

        result = mlink_dsp_stop(&link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

```
0.0 sec, output: 0.0
0.1 sec, output: 0.0
0.2 sec, output: 0.0
0.3 sec, output: 0.0
0.4 sec, output: 0.0
0.5 sec, output: 0.0
0.6 sec, output: 0.0
0.7 sec, output: 0.0
0.8 sec, output: 0.0
0.9 sec, output: 0.0
1.0 sec, output: 0.0
1.1 sec, output: 1.0
1.2 sec, output: 1.0
1.3 sec, output: 1.0
1.4 sec, output: 1.0
1.5 sec, output: 1.0
1.6 sec, output: 1.0
1.7 sec, output: 1.0
1.8 sec, output: 1.0
1.9 sec, output: 1.0
```

Figure 1. Console output of the example.

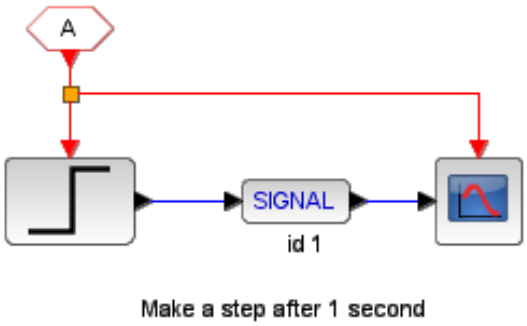

Figure 2. Xcos model (signalmodel.zcos).

mlink_dsp_mem_write

Write data to DSP application

Function prototype

```
int mlink_dsp_mem_write(int *link_fd, int start_idx, int len, float *data);
```

Description

This function allows writing user data to DSP application during its execution. The function has to be used with Xcos MEM read block. Function writes *data* from *start_idx* to *start_inx* + *len*. MEM read block can read up to 250000 values.

Arguments

- **link_fd:** valid connection descriptor
- **start_idx:** Xcos MEM read block start index (1 - 250000)
- **len:** data length (max 250000 values starting with *start_idx*=1)
- **data:** data to be written

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example runs **'dspmemrd.out'** the application generated from Scilab Xcos model on MicroDAQ DSP core. The DSP application reads data from device memory and passes to DAC analog output channel 1 (AO1). The host application writes data to device memory over Ethernet or Wi-Fi. After 5 seconds application is terminated by *mlink_dsp_stop()* function.

The example can be only executed with MicroDAQ E1100 and E2000 series. The Xcos DSP application **'dspmemrd.out'** is located in **'MLink dsp examples\mlink_dsp_mem_write\'** directory attached to this document.

```
#include <stdio.h>
#include <Windows.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int result, link_fd;
        float data[] = {1.0f, 2.0f};
        int data_size = 2;
        int start_idx = 1;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
```

```
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_dsp_run(&link_fd, "dspmemrd.out", 0.1);
    if(result < 0)

                MLINK_ERROR(result);

        result = mlink_dsp_mem_write(&link_fd, start_idx, data_size, data);
        if(result < 0)
                MLINK_ERROR(result);

        Sleep(5000);

        result = mlink_dsp_stop(&link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

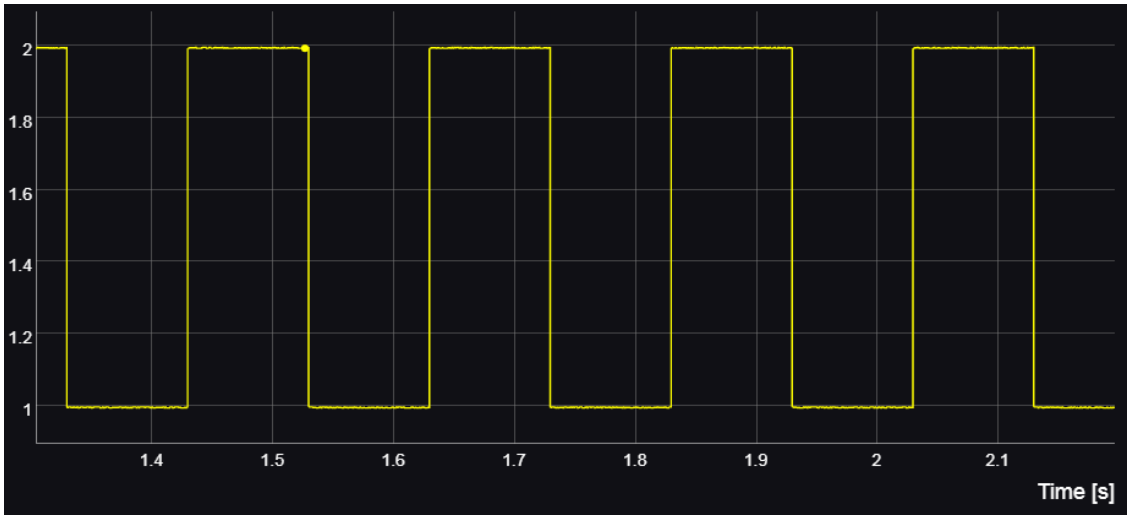

Figure 3. Result of the example , analog output - AO1.

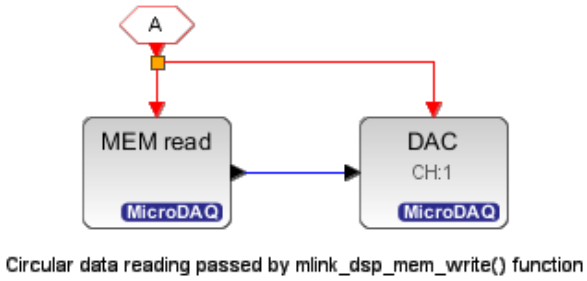

Figure 4. Xcos model (dspmemrd.zcos)

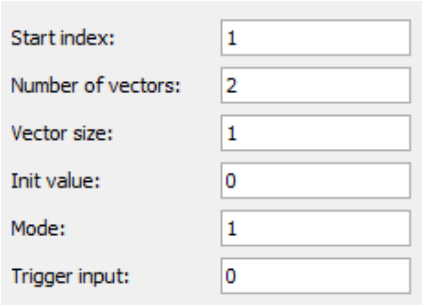

Figure 5. MEM read block parameters

mlink_dsp_stop

Stop DSP application

Function prototype

```
int mlink_dsp_stop(int *link_fd);
```

Description

This function stops DSP application. It can be used for Xcos generated application only.

Arguments

- **link_fd:** valid connection descriptor

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example runs **'blinkingled.out'** application on MicroDAQ DSP core. After 5 seconds application is terminated by *mlink_dsp_stop()* function. The example can be only executed with MicroDAQ E1100 and E2000 series. The Xcos DSP application **'blinkingled.out'** is located in **'MLink dsp examples\ mlink_dsp_run stop\'** directory attached to this document.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>

#define MLINK_ERROR(err)  {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result, i;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_dsp_run(&link_fd, "blinkingled.out", 0.1);
        if(result < 0)
                MLINK_ERROR(result);

        Sleep(5000);

        //End execution
        result = mlink_dsp_stop(&link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

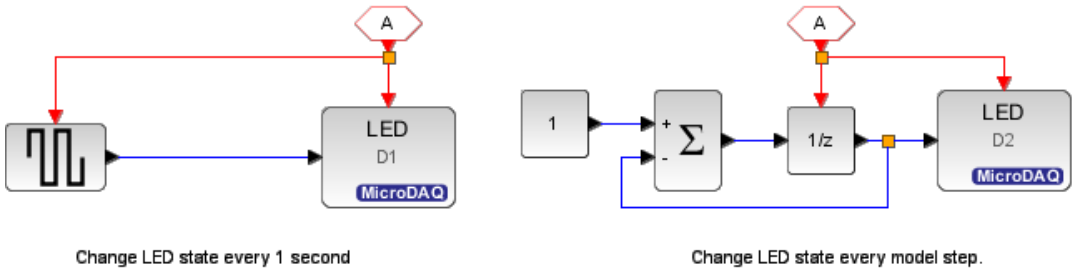

Figure 6. Xcos model (blinking-led.zcos)

mlink_dio_set_func

Control DIO channel alternative functions

Function prototype

```
int mlink_dio_set_func( int *link_fd, uint8_t function, uint8_t enable);
```

Description

This function controls DIO alternative functions. It enables or disables alternative DIO functions. By default all alternative DIO functions are enabled.

Arguments

- **link_fd:** valid connection descriptor
- **function:** DIO alternative function:
  1 - ENC1: DIO1 - Channel A, DIO2 - Channel B (enabled by default)
  2 - ENC2: DIO3 - Channel A, DIO4 - Channel B (enabled by default)
  3 - PWM1: DIO10 - Channel A, DIO11 - Channel B (enabled by default)
  4 - PWM2: DIO12 - Channel A, DIO13 - Channel B (enabled by default)
  5 - PWM3: DIO14 - Channel A, DIO15 - Channel B (enabled by default)
  6 - UART: DIO8 - Rx, DIO9 - Tx (enabled by default)
- **enable:** Function state (1/0 to enable/disable function)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example disables all MicroDAQ DIO alternative functions (ENC1, ENC2, PWM1, PWM2, PWM3 and URAT) and reads DIO1…DIO8 states.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)  {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result;
        uint8_t di_val, i;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Disable encoder1, encoder2, PWM1-3, UART functions to free DIO1-8
        for(i=1; i<7; i++){
                result = mlink_dio_set_func(&link_fd, i, 0);
                if(result < 0)
                        MLINK_ERROR(result);
        }

        // Read 1-8 DI states
        for (i=1; i < 9; i++){
                result = mlink_dio_read(&link_fd, i, &di_val);
                if(result < 0)
                        MLINK_ERROR(result);

                printf("DI%d state: %d\n", i, di_val);
        }

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_dio_set_dir

Set DIO bank direction

Function prototype

```
int mlink_dio_set_dir( int *link_fd, uint8_t bank, uint8_t dir, uint8_t init_value );
```

Description

This function sets DIO bank direction. The DIO bank contains eight DIO channels (bank 1 - DIO1...8, bank 2 - DIO9....16, bank 3 - DIO17...24, bank 4 - DIO25...32). The function allows DIO bank direction change if it is supported by MicroDAQ hardware.

Arguments

- **link_fd:** valid connection descriptor
- **bank:** bank number (1-4)
- **dir:** bank direction (0 - input, 1 - output)
- **init_value:** Initial state of DIO bank

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example sets DIO bank 2 to output and writes states to DIO9-16 (bank 2).

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)   {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
int main()
{
        int link_fd, result;
        uint8_t i;
        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);
        // Set first 9-16 digital channels to output mode
        result = mlink_dio_set_dir(&link_fd, 2, 1, 0);
        if(result < 0)
                MLINK_ERROR(result);
        // Write HIGH state to DO9-16
        for (i=9; i < 17; i++){
                result = mlink_dio_write(&link_fd, i, 1);
                if(result < 0)
                        MLINK_ERROR(result);
        }
        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_dio_write

Write DIO channel state

Function prototype

```
int mlink_dio_write( int *link_fd, uint8_t channel, uint8_t state);
```

Description

This function writes DIO channel state. Function sets DIO *channel* with *state* (0 or 1).

Arguments

- **link_fd: valid connection descriptor**
- **channel:** DIO channel number (1-16|32)
- **state:** state to be set (0|1)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example sets DIO bank 2 to output mode and writes states to DIO9…16 (bank 2).

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result;
        uint8_t i;
        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Set first 9-16 digital channels to output mode
        result = mlink_dio_set_dir(&link_fd, 2, 1, 0);
        if(result < 0)
                MLINK_ERROR(result);

        // Write HIGH state to DO1-8
        for (i=9; i < 17; i++){
                result = mlink_dio_write(&link_fd, i, 1);
                if(result < 0)
                        MLINK_ERROR(result);
        }

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_dio_read

Read DIO channel state

Function prototype

```
int mlink_dio_read( int *link_fd, uint8_t channel, uint8_t *state );
```

Description

This function reads DIO channel state. The function read *channel* and stores its state in variable pointed by *state.*

Arguments

- **link_fd:** valid connection descriptor
- **channel:** DIO channel number (1-16|32)
- **state:** pointer to state variable

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example sets DIO bank 2 to input mode and reads states of DIO9…16 (bank 2).

```
#include <stdio.h>
#include "MLink.h"
#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result;
        uint8_t di_val, i;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Set first 1-8 digital channels to input mode
        result = mlink_dio_set_dir(&link_fd, 2, 0, 0);
        if(result < 0)
                MLINK_ERROR(result);
        // Read 1-8 DI states
        for (i=9; i < 17; i++){
                result = mlink_dio_read(&link_fd, i, &di_val);
                if(result < 0)
                        MLINK_ERROR(result);

                printf("DI%d state: %d\n", i, di_val);
        }
        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_led_write

Set LED state

Function prototype

```
int mlink_led_write( int *link_fd, uint8_t led, uint8_t state);
```

Description

This function sets states of MicroDAQ built in D1 and D2 LEDs.

Arguments

- **link_fd:** valid connection descriptor
- **led:** LED number (1|2)
- **state:** LED state (0|1)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example turns on MicroDAQ built-in D1 and D2 LEDs for 1 second.

```
#include <stdio.h>
#include <Windows.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
	int link_fd, result;
	result = mlink_connect("10.10.1.1", 4343, &link_fd);
	if(result < 0)
	  MLINK_ERROR(result);

	mlink_led_write(&link_fd, 1, 1);
	 mlink_led_write(&link_fd, 2, 1);

	Sleep(1000);

	 mlink_led_write(&link_fd, 1, 0);
	 mlink_led_write(&link_fd, 2, 0);

	mlink_disconnect(link_fd);
	return 0;
}
```

mlink_func_read

Get F1/F2 function key state

Function prototype

```
int mlink_func_read( int *link_fd, uint8_t key, uint8_t *state );
```

Description

This function reads F1/F2 function key state. The function read *key* and stores its state in variable pointed by *state.*

Arguments

- **link_fd:** valid connection descriptor
- **channel:** function key (1|2)
- **state:** pointer to state variable

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example reads state of the MicroDAQ built-in function keys.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result, i;
        uint8_t key1_state, key2_state;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        for(i=0; i < 100; i++){
                mlink_func_read(&link_fd, 1, &key1_state);
                mlink_func_read(&link_fd, 2, &key2_state);
                printf("Func key 1:%d, Func key 2: %d\n", key1_state, key2_state);
                Sleep(50);
        }

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_enc_init

Initialize quadrature encoder

Function prototype

```
int mlink_enc_init( int *link_fd, uint8_t channel, int32_t init_value );
```

Description

This function initializes selected quadrature encoder module (ENC1, ENC2) with provided initial value. The function has to be called before *mlink_enc_read()* function call.

Arguments

- **link_fd:** valid connection descriptor
- **channel:** ENC channel (1|2)
- **init_value:** initial position register value

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example reads position and rotation direction of the MicroDAQ ENC1 module.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>
#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
int main(){
        int link_fd, result, i, enc_val;
        uint8_t enc_dir;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        result = mlink_enc_init(&link_fd, 1, 0);
        if(result < 0)
                MLINK_ERROR(result);

        for(i=0; i < 100; i++){
                mlink_enc_read(&link_fd, 1, &enc_dir, &enc_val);
                printf("Direction: %d | Value: %d\n", enc_dir, enc_val);
                Sleep(50);
        }
        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_enc_read

Read quadrature encoder

Function prototype

```
int mlink_enc_read( int *link_fd, uint8_t channel, uint8_t *dir, int32_t *value);
```

Description

This function reads the current value of quadrature encoder (ENC1, ENC2) position register on selected ENC channel. The function returns current position and rotation direction. The position is a signed 32-bit value containing the number of pulses counted in x4 mode. Value of direction indicates rotation direction (0 - no motion, 1 - CW, 2 - CCW).

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Arguments

- **link_fd:** valid connection descriptor
- **channel:** ENC module (1|2)
- **dir:** pointer to direction variable
- **value:** pointer to encoder counter value

Examples

This example reads position and rotation direction of the MicroDAQ ENC1 module.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
int main()
{
        int link_fd, result, i, enc_val;
                uint8_t enc_dir;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                        MLINK_ERROR(result);

        result = mlink_enc_init(&link_fd, 1, 0);
        if(result < 0)
                MLINK_ERROR(result);

        for(i=0; i < 100; i++){
                mlink_enc_read(&link_fd, 1, &enc_dir, &enc_val);
```

```
                printf("Direction: %d | Value: %d\n", enc_dir, enc_val);
                Sleep(50);
        }

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_pwm_init

Initialize PWM module

Function prototype

```
int mlink_pwm_init( int *link_fd, uint8_t module, uint32_t period, uint8_t active_low,
float duty_a, float duty_b );
```

Description

This function initializes MicroDAQ PWM module. Each PWM module has A and B channel which can generate PWM waveform with different duty and same period defined for PWM module. PWM waveform *period* is defined in microseconds (us). The function allows to generate inverted PWM waveform by setting *active_low* variable to 1. The *duty_a* and *duty_b* determines initial PWM duty (0-100).

Arguments

- **link_fd:** valid connection descriptor
- **module:** PWM module (1|2|3)
- **period:** PWM module period in microseconds (2-500000)
- **active_low:** generate inverted PWM waveform (1|0 to enable or disable)
- **duty_a:** initial PWM channel A duty (0-100)
- **duty_b:** initial PWM channel B duty (0-100)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example generates PWM waveform for 5 seconds with 1000 microseconds period and 50% duty on PWM1 module on channel A and B.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result;
        uint8_t pwm_module = 1;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Setup PWM output channels. Period 1000 microseconds.
        result = mlink_pwm_init(&link_fd, pwm_module, 1000, 0, 0, 0);
        if(result < 0)
                MLINK_ERROR(result);
        mlink_pwm_write( &link_fd, pwm_module, 50, 50);
        Sleep(5000);
        mlink_pwm_write( &link_fd, pwm_module, 0, 0);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_pwm_write

Write PWM duty

Function prototype

```
int mlink_pwm_write( int *link_fd, uint8_t module, float duty_a, float duty_b );
```

Description

This function sets PWM waveform duty for A and B channels for selected PWM module. PWM module has to be initiated with *mlink_pwm_init()* function call.

Arguments

- **link_fd:** valid connection descriptor
- **module:** PWM module (1|2|3)
- **duty_a:** initial PWM channel A duty (0-100)
- **duty_b:** initial PWM channel B duty (0-100)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example generates PWM waveform for 5 seconds with 1000 microseconds period and 50% duty on PWM1 module on channel A and B.

```
#include <stdio.h>
#include "MLink.h"
#include <Windows.h>

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result;
        uint8_t pwm_module = 1;

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Setup PWM output channels. Period 1000 microseconds.
        result = mlink_pwm_init(&link_fd, pwm_module, 1000, 0, 0, 0);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_pwm_write( &link_fd, pwm_module, 50, 50);
        Sleep(5000);
        mlink_pwm_write( &link_fd, pwm_module, 0, 0);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_ai_read

Read analog input

Function prototype

```
int mlink_ai_read( int *link_fd, uint8_t *channels, uint8_t ch_count, double *range,
uint8_t *mode, double *data );
```

Description

This function returns immediately acquired values from MicroDAQ input channels as a 1-by-n array of doubles. The values are stored in an array pointed by *data*, where *n* is the number of input channels. The *ch_count* argument determines number of used channels. The *channels* argument is an array containing channels numbers according to MicroDAQ hardware configuration. The *range* argument specifies channel measurement input range. An array n-by-2 where n is number of used channels shall be provided. In order to obtain supported ranges check your ADC configuration. The *mode* argument specifies measurement mode - differential or single-ended. An *mode* array of mode settings for used channels shall be provided.

Arguments

- **link_fd:** valid connection descriptor
- **channels:** array with channels to be read
  Analog input channel selection in differential mode:
  - Channel 1 - AI1(+), AI2(-)
  - Channel 2 - AI3(+), AI4(-)
  - Channel 3 - AI5(+), AI6(-)
  - Channel 4 - AI7(+), AI8(-)
  - Channel 5 - AI9(+), AI10(-)
  - Channel 6 - AI11(+), AI12(-)
  - Channel 7 - AI13(+), AI14(-)
  - Channel 8 - AI15(+), AI16(-)
- **ch_count:** length of *channels* array
- **range:** array with range parameters for selected channels
- **mode:** differential or single-ended terminal configuration selection (0|1 for single-ended or respectively)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example reads data from analog input 1..8 (AI1..AI8). Input range is -10 to 10V for all channels.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
        int link_fd, result, i;
        double data[8];
        double ranges[] = {-10, 10, -10, 10, -10, 10, -10, 10, -10, 10, -10, 10, -10, 10, -10, 10};
        uint8_t channels[] = {1, 2, 3, 4, 5, 6, 7, 8};
        uint8_t modes[] = {0, 0, 0, 0, 0, 0, 0, 0};

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        // Read analog input channels AI1..8
        result = mlink_ai_read(&link_fd, channels, sizeof(channels), ranges, modes,
data);
        if(result < 0)
                MLINK_ERROR(result);

        for(i=0; i<8; i++)
                printf("AI%d: %f V\n", i, data[i]);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_ai_scan_init

Initialize analog input scanning session

Function prototype

```
int  mlink_ai_scan_init(int *link_fd, uint8_t *channels, uint8_t ch_count, double
*range, uint8_t *mode, float *rate, float duration);
```

Description

This function initiates analog input scanning session. The function must be called before acquisition started. The *channels* argument is an array containing channels numbers according to MicroDAQ hardware configuration. The *ch_count* argument determines number of used channels. The *range* argument specifies channel measurement input range. An array n-by-2 where n is number of used channels shall be provided. In order to obtain supported ranges

check your ADC configuration. The *mode* argument specifies measurement mode - differential or single-ended. An *mode* array of mode settings for used channels shall be provided.

The *rate* argument determines scans per second rate for selected analog input channels. Minimum value is 1 scan per second, maximum depends on MicroDAQ analog input type.

The *duration* argument specifies a duration of acquisition in seconds. When set to -1, the session will run continuously, acquiring data until stopped with *mlink_ai_scan_stop*() function call.

Limitation: ADC1-DAC01 MicroDAQ configuration doesn't support running simultaneously AI and AO scanning sessions.

Arguments

- **link_fd:** valid connection descriptor
- **channels:** array with channels to be read
  Analog input channel selection in differential mode:
  - Channel 1 - AI1(+), AI2(-)
  - Channel 2 - AI3(+), AI4(-)
  - Channel 3 - AI5(+), AI6(-)
  - Channel 4 - AI7(+), AI8(-)
  - Channel 5 - AI9(+), AI10(-)
  - Channel 6 - AI11(+), AI12(-)
  - Channel 7 - AI13(+), AI14(-)
  - Channel 8 - AI15(+), AI16(-)
- **ch_count:** length of *channels* array
- **range:** array with range parameters for selected channels
- **mode:** differential or single-ended terminal configuration selection (0|1 for single-ended or respectively)
- **rate:** analog input per second update rate (1 - depends on ADC type)
- **duration:** analog input scan duration in seconds (-1 - continuous)

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example acquires data from analog input 1 (AI1). Sampling rate is set to 10ksps, input range is -10 to 10V. After reading of 10 000 samples (1 second) the acquisition is finished.

```
#include <stdio.h>
#include <stdint.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
    int result, link_fd;
    uint8_t ai_channel[] = {1};
    uint8_t ch_count = 1;
    uint8_t diff[] = {0};
    float duration = 1.0, rate = 10000.0;
    double ai_range[] = {-10,10};
    double data[10000];

    result = mlink_connect("10.10.1.1", 4343, &link_fd);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan_init(&link_fd, ai_channel, ch_count, ai_range, diff,
&rate, duration);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan(data, 10000, 1);
    if (result < 0)
        MLINK_ERROR(result);

    mlink_disconnect(link_fd);
    return result;
}
```

mlink_ai_scan

Start analog input scanning session and read acquired data

Function prototype

```
int  mlink_ai_scan(double *data, uint32_t scan_count, int32_t blocking);
```

Description

This function starts analog input scanning session and reads acquired data. A function call has to be preceded with *mlink_ai_scan_init()* which initiates scanning session parameters. Blocking or non-blocking operation is defined by *blocking* argument. When blocking (*blocking = 1*) mode is used, function will block until desired number of scan is acquired. The function has fixed 2-second timeout, a user has to provide *scan_count* argument which will not cause *mlink_ai_scan*() function to time-out e.g scanning rate = 100 scans per second and user wants to read 500 samples by single *mlink_ai_scan()* call. When non-blocking (*blocking=0*) mode is used, function acquire scan data which is currently available in analog input data queue. The number of acquired scans is returned by function. The *scan_count* argument in non-blocking mode determines maximum number of scans which can be acquired by function call. Values of acquired data, returned as an m-by-n array, where m is the number of acquired scans, and n is the number of used input channels (defined during initialization with *mlink_ai_scan_init()* function call).

Limitation: ADC1-DAC01 MicroDAQ configuration doesn't support running simultaneously AI and AO scanning sessions.

Return value

On success, number of acquired scans. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Arguments

- **link_fd: valid connection descriptor**
- **scan_count:** number of scans to read, when 0 function starts acquisition without reading data
- **blocking:** blocking or non-blocking read (1|0)

Examples

This example acquires data from analog input 1 (AI1) . Sampling rate is set to 10ksps, input range is -10 to 10V. After reading of 10 000 samples (1 second) the acquisition is finished.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}

int main()
{
    int result, link_fd;
    uint8_t ai_channel[] = {1};
    uint8_t ch_count = 1;
    uint8_t diff[] = {0};
    float duration = 1.0, rate = 10000.0;
    double ai_range[] = {-10,10};
    double data[10000];

    result = mlink_connect("10.10.1.1", 4343, &link_fd);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan_init(&link_fd, ai_channel, ch_count, ai_range, diff, &rate, duration);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan(data, 10000, 1);
    if (result < 0)
        MLINK_ERROR(result);

    mlink_disconnect(link_fd);
    return result;
}
```

mlink_ai_scan_stop

Stop analog input scanning session

Function prototype

```
int  mlink_ai_scan_stop( void );
```

Description

This function stops analog input acquisition. Function can be used to interrupt acquisition when scanning session initialized with *duration* > 0, or to stop continuous acquisition when scanning session initialized with *duration* = -1 . After calling *mlink_ai_scan_stop()* function scanning session has to be re-initialized.

Return value

On success, 0 is returned. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Examples

This example acquires data from analog input 1 (AI1). Sampling rate is set to 10ksps, input range is -10 to 10V. After reading of 10 000 samples (1 second) the acquisition is stopped manually by *mlink_ai_scan_stop()* function.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
int main()
{
    int result, link_fd;
    uint8_t ai_channel[] = {1}, ch_count = 1, uint8_t diff[] = {0};
    float rate = 10000.0, duration = 60.0;
    double ai_range[] = {-10,10}, data[10000];

    result = mlink_connect("10.10.1.1", 4343, &link_fd);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan_init(&link_fd, ai_channel, ch_count, ai_range, diff, &rate, duration);
    if (result < 0)
        MLINK_ERROR(result);

    result = mlink_ai_scan(data, 10000, 1);
    if (result < 0)
        MLINK_ERROR(result);

    mlink_ai_scan_stop();
    mlink_disconnect(link_fd);
    return result;
}
```

mlink_ao_write

Write analog output

Function prototype

```
int mlink_ao_write( int *link_fd, uint8_t *channels, uint8_t ch_count, double *range,
uint8_t mode, double *data );
```

Description

This function writes MicroDAQ analog outputs. The *channels* argument is an array containing channels numbers according to MicroDAQ hardware configuration. The *ch_count* argument determines number of used channels. The *range* argument specifies channel output range. An array n-by-2 where n is number of used channels shall be provided. In order to obtain supported ranges check your analog output specification. The *data* argument points to array containing data to be set. The *data* array size must be same size as *channels* array and *ch_count* parameter.

Return value

On success, number of acquired scans. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Arguments

- **link_fd:** valid connection descriptor
- **channels:** array with channels numbers
- **ch_count:** length of *channels* array
- **range:** array with range parameters for selected channels
- **data:** pointer to array with data to be written

Examples

This example writes to MicroDAQ analog outputs 1..8 (AO1..AO8) values from 0.0V up to 3.5V. Output range is set to 0-5V on all channels.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
int main()
{
        int link_fd, result;
        double ranges[] = {0, 5, 0, 5, 0, 5, 0, 5, 0, 5, 0, 5, 0, 5, 0, 5};
        uint8_t channels[] = {1, 2, 3, 4, 5, 6, 7, 8};
        double data[] = {0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5};

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
```

```
                MLINK_ERROR(result);

        // Write to analog output AO1..8
        result = mlink_ao_write(&link_fd, channels, 8, ranges, 0, data);
        if(result < 0)
                MLINK_ERROR(result);

        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_ao_scan_init

Initiates analog outputs scanning session.

Function prototype

```
int mlink_ao_scan_init(int *link_fd, uint8_t *channels, uint8_t ch_count, float *data,
int data_size, double *range, uint8_t stream_mode, float rate, float duration);
```

Description

This function initiates analog output scanning session. The function must be called before acquisition is started. The *channels* argument is an array containing channels numbers according to MicroDAQ hardware configuration. The *ch_count* argument determines number of used channels. The *range* argument specifies channel output range. An array n-by-2 where n is number of used channels shall be provided. In order to obtain supported ranges check your analog output specification. The *data* argument contains data to be output and is specified as a float array of n x m elements lenght (*data_size*), where m is the number of scans to generate, and n is the number of used output channels.

Important: The *data* argument determines size of data which can be queued with mlink_ao_scan_data(). Queued data size must be the same size as data argument in mlink_ao_scan_init() function.

The *rate* argument determines scans per second rate for selected analog input channels. Minimum value is 1 scan per second, maximum depends on MicroDAQ analog output type. The *duration* argument specifies a duration of acquisition in seconds. When set to -1, the session will run continuously, writing data until stopped with *mlink_ao_scan_stop()* function. The *stream_mode* argument determines scanning session behavior. Two modes are available - periodic and stream.

The periodic mode (*stream_mode = 0*) uses a single buffer which data is output from. When the end of the buffer is reached, data index is switched to the beginning of the buffer. This mode is suitable for generating periodic signals e.g sine waveform. The buffer can be changed during signal generation with *mlink_ao_scan_data()* function.

The stream mode (*stream_mode = 1*) uses two buffers architecture to ensure uninterrupted analog signal generation. This mode of operation is suitable for stream data type e.g.

generating audio stream, this mode requires from user to queue data with certain time constraints. If new data isn't queued on time, last value remains on analog output until new data has been queued.

Return value

On success, number of acquired scans. On error, negative value is returned. The *mlink_error()* can be used to get error description.

Arguments

- **link_fd:** valid connection descriptor
- **channels:** array with channels numbers
- **ch_count:** size of channels array
- **range:** array with range parameters for selected channels
- **data:** pointer to array with data to be written
- **data_size:** size of data array (max: stream mode - 1048576 values | periodic mode - 2097152 values)
- **stream_mode:** periodic or stream mode (0|1)
- **rate:** analog output per second update rate (1 - depends on DAC type)
- **duration:** analog output scan duration in seconds (-1 - continuous)

Examples

This example generates sawtooth waveform on analog output channel 1 (AO1). Waveform period has 100 samples and DAC output rate is set to 1000 samples per second which results in 10Hz sawtooth waveform. After 10 second, generation is stopped.

```
#include <stdio.h>
#include <stdint.h>
#include "MLink.h"

#define MLINK_ERROR(err)         {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                (100)

int main()
{
        int link_fd, result, i;
        uint8_t channels[] = {1};
        float data[DATA_SIZE];
        double ranges[] = {0, 5};
        uint8_t stream_mode = 0;
        float rate = 1000, duration = 10;

        //Generate sawtooth wave
        float acc = 0;
        for(i = 0; i < DATA_SIZE; i++){
                data[i] = acc;
```

```
                acc += 5.0 / (float)DATA_SIZE;
        }

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        //Set up periodic mode (stream_mode=0)
        result = mlink_ao_scan_init(&link_fd, channels, 1, data, DATA_SIZE, ranges,
        stream_mode, rate, duration);
        if(result < 0)
                MLINK_ERROR(result);

        //Start generating signal, frequency = rate / DATA_SIZE
        mlink_ao_scan(&link_fd);
        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_ao_scan

Starts analog output scanning.

Function prototype

```
int mlink_ao_scan(int *link_fd);
```

Description

This function starts analog output scanning. A function call has to be preceded with *mlink_ao_scan_init()* which initiates analog output scanning session parameters. Function enables MicroDAQ hardware to output data on selected AO channels. In order to stop scanning, function *mlink_ao_scan_stop()* has to be called.

Limitation: ADC1-DAC01 MicroDAQ configuration doesn't support running simultaneously AI and AO scanning sessions.

Arguments

- **link_fd: valid connection descriptor**

Examples

This example generates sawtooth waveform on analog output channel 1 (AO1). Waveform period has 100 samples and DAC output rate is set to 1000 samples per second which results in 10Hz sawtooth waveform. After 10 second, generation is finished.

```
#include <stdio.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                 (100)

int main()
{
        int link_fd, result, i;
        uint8_t channels[] = {1};
        float data[DATA_SIZE];
        double ranges[] = {0, 5};
        uint8_t stream_mode = 0;
        float rate = 1000, duration = 10;

        //Generate sawtooth wave
        float acc = 0;
        for(i = 0; i < DATA_SIZE; i++){
                data[i] = acc;
                acc += 5.0 / (float)DATA_SIZE;
        }

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
```

```
                MLINK_ERROR(result);

        //Set up periodic mode (stream_mode=0)
        result = mlink_ao_scan_init(&link_fd, channels, 1, data, DATA_SIZE, ranges, stream_mode, rate,
        duration);
        if(result < 0)
                MLINK_ERROR(result);

        //Start generating signal, frequency = rate / DATA_SIZE
        mlink_ao_scan(&link_fd);
        mlink_disconnect(link_fd);
        return 0;
}
```

mlink_ao_scan_data

Queues data to be output.

Function prototype

```
int mlink_ao_scan_data(int *link_fd, uint8_t *channels, int ch_count, float *data, int
data_size, uint8_t opt);
```

Description

This function queues data to be output. A function call has to be preceded with *mlink_ao_scan_init()* . Function queues data in the stream and periodic mode and its behavior depends on selected scan mode. In periodic mode, the function can queue data for every channel combination from used channels (defined in *mlink_ao_scan_init()*). If e.g. in scanning session four channels are used: 1, 2, 3, 4 *mlink_ao_scan_data()* can be called to queue data for 1 or 4 or 1,2 or 1,4, or 3,4 etc. channel or queue data for all selected channels. In periodic mode channels argument can contain every combination of used channels, while in stream mode channel argument must be the same as used in *mlink_ao_scan_init()* . The data argument in stream mode must have the same size as the data argument in function *mlink_ao_scan_init()* . The *mlink_ao_scan_init()* perform initial data queue operation. The periodic mode uses a single buffer which data is output from. When the end of the buffer is reached, data index is switched to the beginning of the buffer. The *mlink_ao_scan_data()* function overwrites the whole buffer with new data. Depending on the scanning mode opt argument has a different meaning. When the periodic mode is used opt argument allows controlling data index after queuing data. When *opt=1* data index will be set to the beginning of the buffer. If *opt=0* queue operation doesn't affect data index.

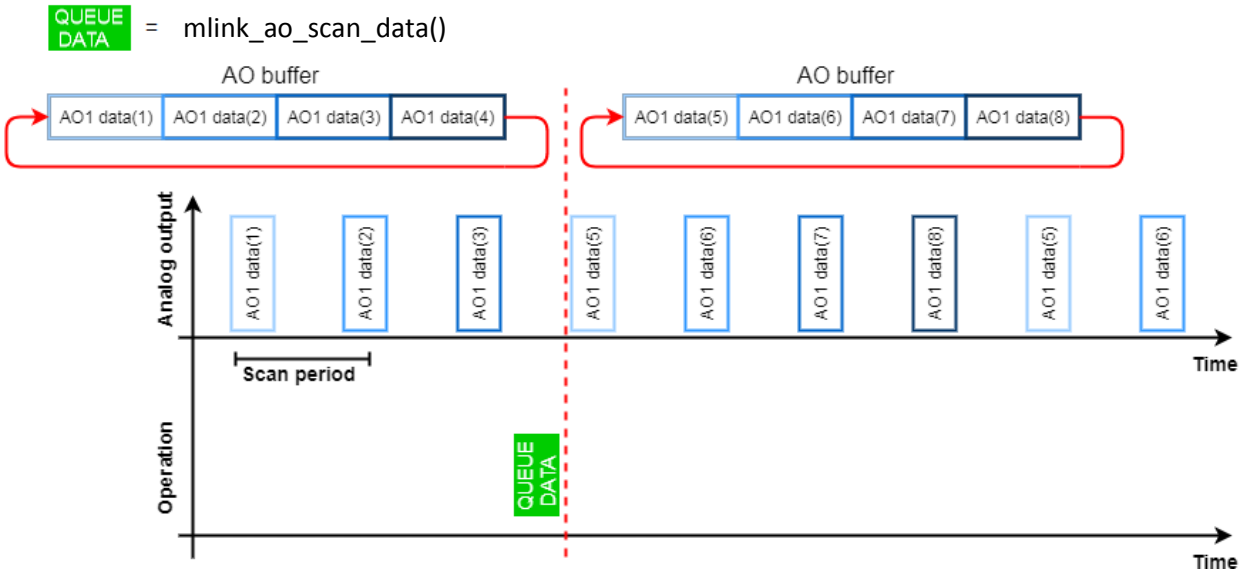

Figure 7. Queuing data in periodic mode (opt=1).

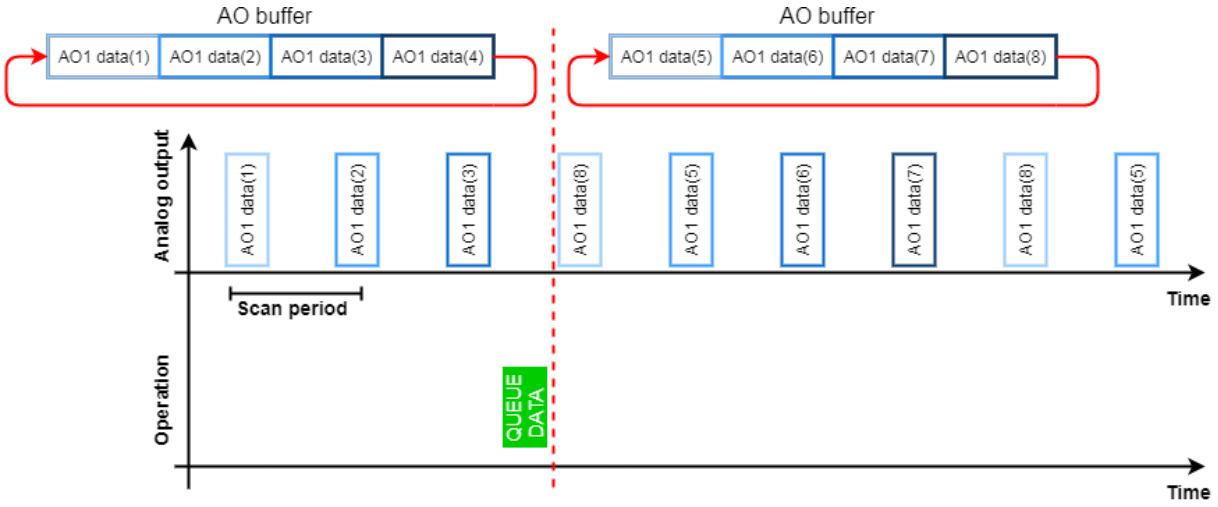

Figure 8. Queuing data in periodic mode (opt=0).

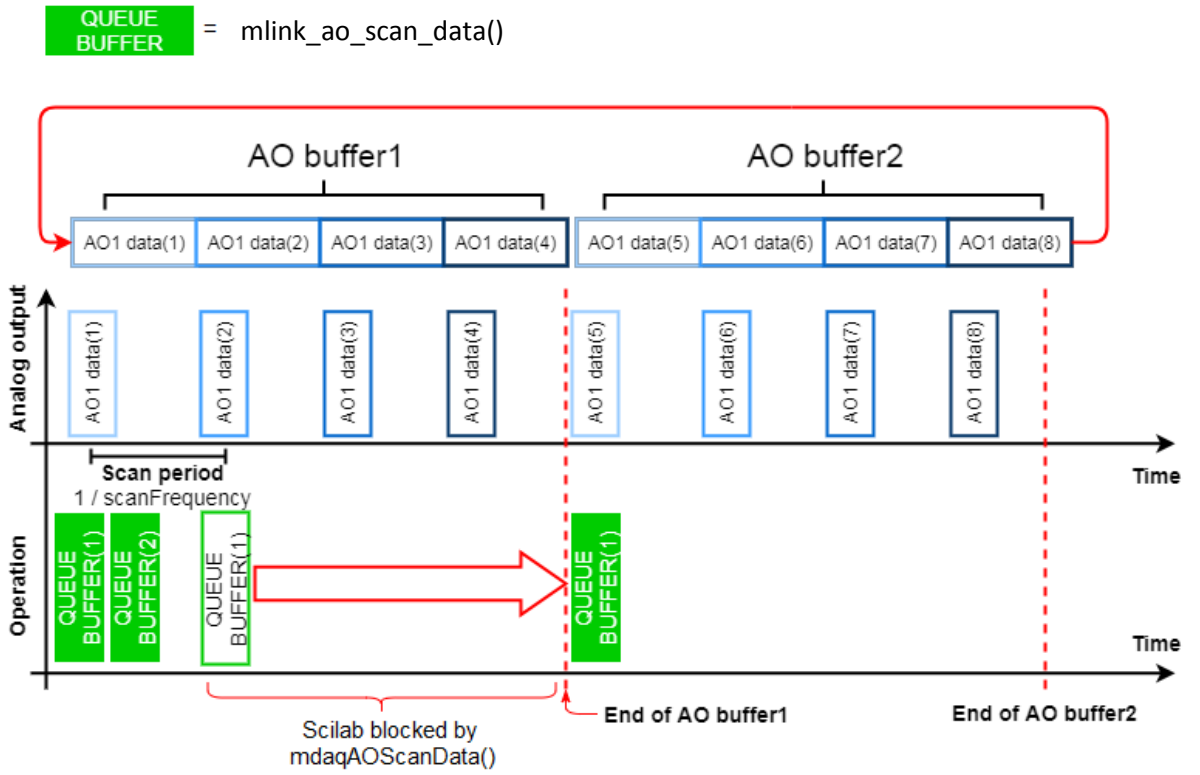

Figure 9. Queuing data in stream mode, blocking operation (opt=0).

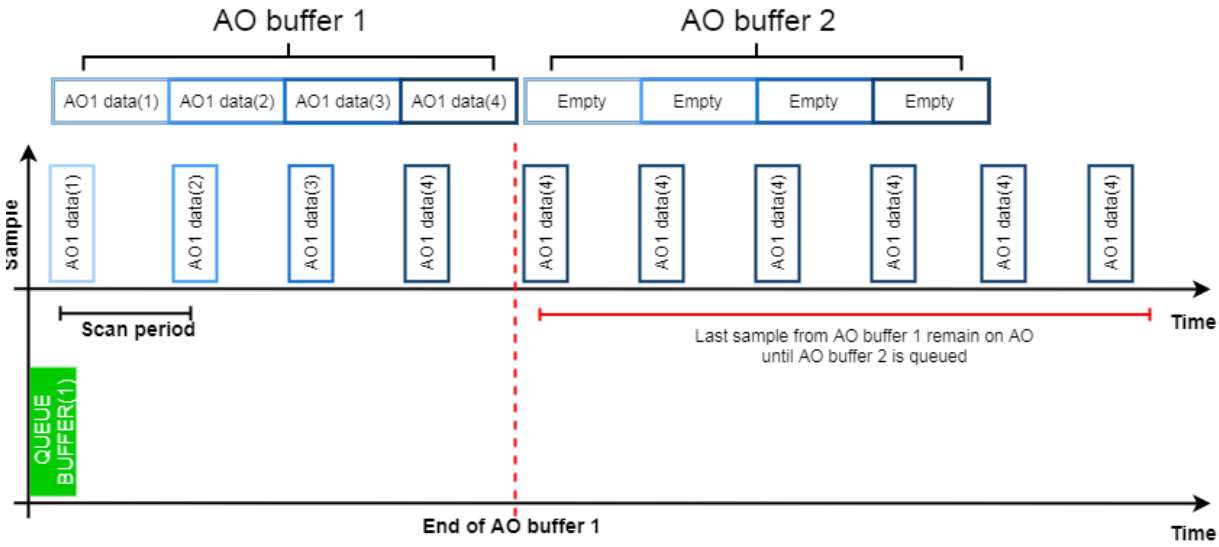

Figure 10. Queuing data in stream mode, data isn't queued on time.

Arguments

- **link_fd:** valid connection descriptor
- **channels:** array with channels numbers
- **ch_count:** size of channels array
- **data:** pointer to array with data to be written
- **data_size:** size of data array (max: stream mode - 1048576 values | periodic mode - 2097152 values)
- **opt:** if 1 reset data index after upload in *periodic mode*. Blocking or non-blocking operation in *stream mode* (1|0)

Examples

Periodic mode

This example generates sawtooth waveform on analog output channel 1 (AO1). Waveform period has 100 samples and DAC output rate is set to 1000 samples per second which results in 10Hz waveform. After 5 second sawtooth is changed to sine waveform for the next 5 seconds and then stopped by the *mlink_ao_scan_stop()* function call.

```
#include <stdio.h>
#include <math.h>
#include <Windows.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                 (100)
#define M_PI                      (3.14159265358979323846)

int main()
{
        int link_fd, result, i;

        uint8_t channels[] = {1};
        float data[DATA_SIZE], data2[DATA_SIZE];
        double ranges[] = {0, 5};
        uint8_t stream_mode = 0;
        float rate = 1000;
        float duration = -1; // no time limit

        //Generate sawtooth and sine wave
        float saw_acc = 0, saw_step = 5.0 / (float)DATA_SIZE;
        float sine_acc = 0, sine_step = (2*M_PI) / (float)DATA_SIZE;
        for(i = 0; i < DATA_SIZE; i++, sine_acc += sine_step, saw_acc += saw_step){
                data[i] = (sin(sine_acc)*2.5) + 2.5;
                data2[i] = saw_acc;
        }

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                        MLINK_ERROR(result);
```

```
        //Set up periodic mode (stream_mode=0)
        result = mlink_ao_scan_init(&link_fd, channels, 1, data, DATA_SIZE, ranges, stream_mode, rate,
duration);
        if(result < 0)
                        MLINK_ERROR(result);

        //Start generating sine signal
        mlink_ao_scan(&link_fd);

        Sleep(5000);

        //Start generating sawtooth signal
        mlink_ao_scan_data(&link_fd, channels, 1, data2, DATA_SIZE, 1);

        Sleep(5000);

        //Stop generating signal
        mlink_ao_scan_stop(&link_fd);

        mlink_disconnect(link_fd);
        return 0;
}
```

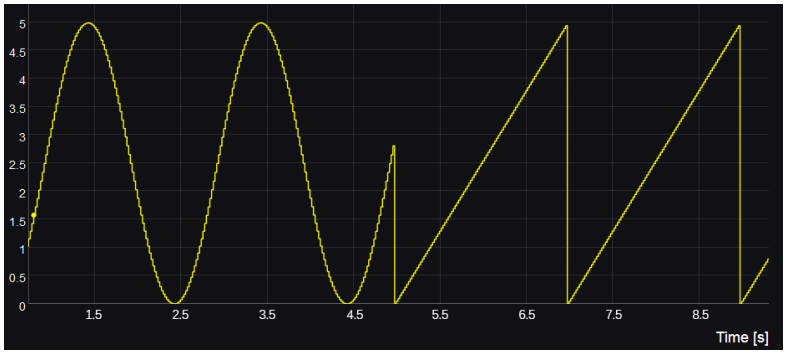

Figure 11. Periodic mode - analog output - AO1.

Stream mode

This example generates noise on analog output channel 1 (AO1) for 1 second. The output noise range 0-1V. The example uses *mlink_ao_scan_data()* function to queue data to analog output buffer.

```
#include <stdio.h>
#include <stdint.h>
#include <time.h>
#include "MLink.h"
#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                 (100)

void noise_generator(float *data, int data_size){
        int i;
        for(i = 0; i < data_size; i++)
                data[i] = rand() / (float)RAND_MAX;
}
int main()
{
        int link_fd, result, i;
        uint8_t channels[] = {1};
        float data[DATA_SIZE];
        double ranges[] = {0, 5};
        uint8_t stream_mode = 1;
        float rate = 1000;
        float duration = 1;
        srand(time(NULL));

        //Generate simple noise
        noise_generator(data, DATA_SIZE);

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        //Set up stream mode (stream_mode=1)
        result = mlink_ao_scan_init(&link_fd, channels, 1, data, DATA_SIZE, ranges, stream_mode, rate,
duration);
        if(result < 0)
                MLINK_ERROR(result);

        //Start generating noise signal
        mlink_ao_scan(&link_fd);

        for(i=0; i < (rate/DATA_SIZE)*duration; i++)
        {
                //Generate new set of noise
                noise_generator(data, DATA_SIZE);
                //Queue data do output
                mlink_ao_scan_data(&link_fd, channels, 1, data, DATA_SIZE, 1);
        }

        mlink_disconnect(link_fd);
        return 0;
}
```

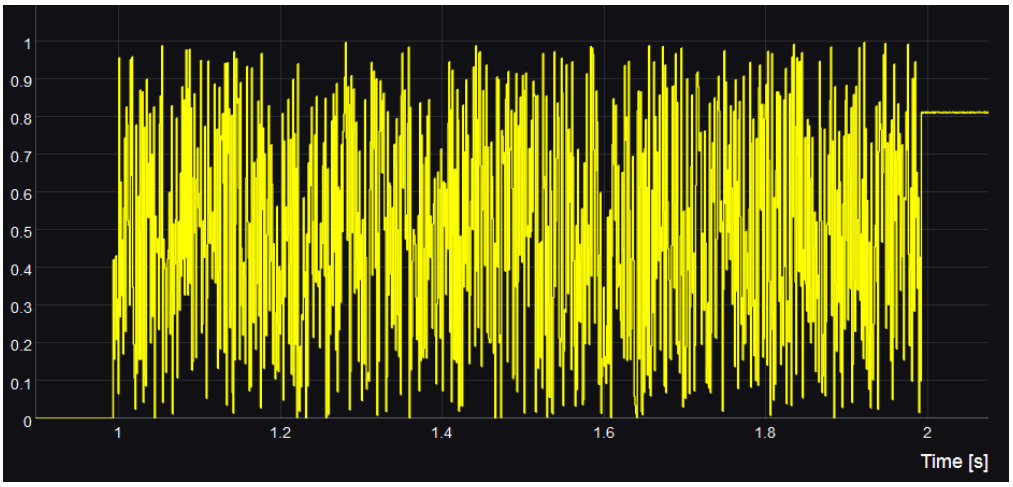

Figure 12. Stream mode - analog output - AO1

mlink_ao_scan_stop

Stops analog output scanning.

Function prototype

```
int mlink_ao_scan_stop(int *link_fd);
```

Description

This function stops analog output scanning session. Function can be used to interrupt acquisition (duration > 0), or to stop continuous acquisition (duration=-1). After calling *mlink_ao_scan_stop()* function, scanning session has to be re-initialized.

Arguments

- **link_fd:** valid connection descriptor

Examples

This example generates sawtooth wavform on analog output channel 1 (AO1). Waveform period has 100 samples and DAC output rate is set to 1000 samples per second which results in 10Hz sawtooth waveform. After 10 second, generation is stopped manually by the *mlink_ao_scan_stop()* function.

```
#include <stdio.h>
#include <Windows.h>
#include "MLink.h"

#define MLINK_ERROR(err)          {printf("MLink error %d: %s\n", err, mlink_error(err)); return 1;}
#define DATA_SIZE                 (100)

int main()
{
        int link_fd, result, i;

        uint8_t channels[] = {1};
        float data[DATA_SIZE];
        double ranges[] = {0, 5};
        uint8_t stream_mode = 0;
        float rate = 1000;
        float duration = -1; // no time limit
        float amplitude = 5.0;

        //Generate sawtooth wave
        float acc = 0;
        for(i = 0; i < DATA_SIZE; i++){
                data[i] = acc;
                acc += amplitude / (float)DATA_SIZE;
        }

        result = mlink_connect("10.10.1.1", 4343, &link_fd);
        if(result < 0)
                MLINK_ERROR(result);

        //Set up periodic mode (stream_mode=0)
        result = mlink_ao_scan_init(&link_fd, channels, 1, data, DATA_SIZE, ranges, stream_mode, rate,
duration);

        if(result < 0)
                MLINK_ERROR(result);

        //Start generating signal
        mlink_ao_scan(&link_fd);

        Sleep(10000);

        //Terminate generation
        mlink_ao_scan_stop(&link_fd);

        mlink_disconnect(link_fd);
        return 0;
}
```