Martel Linux Driver
libmartel Documentation

http://www.martelinstruments.com

Martel Instruments Ltd. reserves the right to make changes without notice to the product to improve reliability, function or design. Martel Instruments Ltd. does not assume any liability arising out of the application or use of the product or circuit described herein.

1. Table of contents

2. Revision history

Rev.	Date	Page	Revision item
A	28-Jul-2006	-	First issue
		-	

3. Introduction

The Martel library provides a standard interface to communicate with Martel printers, regardless of the actual communication port type. The Martel library is a low-level library which basically provides open, close, read from and write to communication ports functionalities. Additionnal function to deal with Martel printer models and printer command sets are also provided.

While the Martel library is used primarily by the Martel CUPS driver, users might also use the Martel library directly. The Martel library gives greater control on the communication with printers, and can speed up the writing of user applications by providing reliable and tested functions.

3.1. Library interface design

Only one single header file needs to be included to use the Martel library.

```
#include <martel/martel.h>
```

Also do not forget to link with the library by passing option `-lmartel` to the linker.

Every type, every function of the Martel library is named after the format `martel_xxx`. This reduces the risk of name conflicts within a single program.

```
errnum = martel_get_error(port);
errnum = martel_open(port);
```

Most functions return a single integer. Positive values have a meaning specific to each function, negative values indicate an error. Refer to library error codes section for a description of each error code.

```
if ((errnum = martel_write(port,buf,size))<0) {
    if (errnum==MARTEL_PORT_NOT_OPEN) {
        fprintf(stderr,"Oops it seems someone forgot to open the port...\n");
    }
    else {
        fprintf(stderr,"Unknown error!\n");
    }
}
```

3.2. Basic sample

The Martel library can be used to build quick sample programs as well as full-featured applications. The following program is an example of a simple application. To keep code simple, this program does not perform error checking.

```
#include <stdio.h>

#include <martel/martel.h>

static const char ticket[] = "Hello world!\n";

int main(int argc,char** argv)
{
        void *port;
        int errnum;

        /* create communication port from URI */
        port = martel_create_port(argv[1]);

        /* open communication port */
        errnum = martel_open(port);

        /* write to communication port */
        errnum = martel_write(port,ticket,sizeof(ticket)-1);

        /* close communication port */
        errnum = martel_close(port);

        /* destroy communication port */
        errnum = martel_destroy_port(port);

        return 0;
}
```

This sample, as well as others, is available in the source samples directory:

- `src/sample/sample1.c`: basic sample program using `libmartel`;
- `src/sample/sample2.c`: basic sample program using `libmartel` and including error checking.

4. Printer port URI

Printer ports are identified within the Martel Library by their Uniform Resource Identifier (URI). A printer port URI describes the device file name, port type and port configuration of a printer port. A printer port URI is a string of the following format.

```
martel:device_file_name?key1=value1+key2=value2...
```

Here are some examples of port URI for serial and parallelprinter ports.

```
martel:/dev/ttyS0?type=serial+baudrate=9600+handshake=rtscts
martel:/dev/parport0?type=parallel+mode=irq
```

The set of *key* = *value* options after the question mark describes port configuration. These options can be written in any order. Option key `type` is mandatory, as it defines the type of printer port.

Key	Applicable port type	Default value	Allowed values	Description
type	any	none	serial parallel	Define type of printer port.
baudrate	serial	9600	1200 2400 4800 9600 19200	Define default serial baudrate in bauds.
handshake	serial	rtscts	none rtscts xonxoff	Define default serial handshake mode.
mode	parallel	poll	poll irq	Define parallel write mode.

4.1.

5. Library functions

5.1. Error handling

```
int martel_get_error(void *port)
```

Return the last error that occured on `port`.

```
const char *martel_strerror(int errnum)
```

Convert the numerical error code `errnum` to a human-readable string.

5.2. Printer models database

The Martel library stores information about the various Martel printer models and provides limited access to this database. Supported printer models are found in `martel/models.def`.

A printer model refers to a specific printer reference within the Martel products range.

A printer model type refers to a specific series of printers, such as MPP5510, MPC7810 or MCP8810 series. All models of a given type use the same command set and the format of their internal status is the same.

```
const char *martel_get_model_name(int model)
```

Return printable name of printer model.

```
int martel_get_model_type(int model)
```

Return type of printer model (one of `martel_model_type_t`).

```
int martel_get_model_width(int model)
```

Return dotline width in pixels of printer model.

```
int martel_decode_status(int type,const void *buf,int size,martel_status_t *status)
```

Decode internal status of printer into a standardized `martel_status_t` structure. Internal printer status is stored in buffer `buf` of `size` bytes. This function needs to know the type of printer model in order to interpret the status buffer. This function does not send commands to the printer to retrieve the actual status, it is the responsability of the application to retrieve the internal status of the printer by using the correct command. See each printer specification for more information.

5.3. Printers port management

All communication with printers are done using printer ports. A printer port defines the current configuration (port type, device file name) and state (open or closed) of a communication port.

```
void *martel_create_port(const char *uri)
```

Create a printer port from an URI.

```
void *martel_create_serial_port(const char *device)
```

Create a serial printer port from device file name and initialize serial settings to default values (9600 bauds, hardware RTS/CTS handshaking).

```
void *martel_create_parallel_port(const char *device)
```

Create a parallel printer port from device file name and initialize settings to default values (polling mode)..

```
int martel_destroy_port(void *port)
```

Destroy a printer port.

```
int martel_get_port_type(void *port)
```

Return the type of a printer port (one of `martel_port_type_t`).

```
int martel_get_port_uri(void *port,char *uri,int size)
```

Format a URI string from a printer port. This function writes the URI string in user-supplied `uri` buffer. This function does not write more than `size` characters (including trailing zero).

5.4. Printers port operations

Several operations can be performed on a printer port. Some of them require the port to be open, while others only perform port configuration. Depending on port type (set during the creation of the printer port), some specific operations may also be available.

```
int martel_open(void *port)
```

Open a printer port.

```
int martel_close(void *port)
```

Close a printer port..

```
int martel_write(void *port,const void *buf,int size)
```

Write `size` bytes of data to port. Data is stored in buffer `buf`.

This function blocks until all data have been transmitted to the kernel buffer. This function also returns with `MARTEL_WRITE_TIMEOUT` if a write timeout has been set and timeout elapsed.

This function does not guarantee that all data have been actually sent to the printer when the function returns. Instead, all data have been transmitted to the kernel, which might still be performing the write operation. This means that some data may still be pending in the kernel buffer. Use function `martel_sync()` to wait until all data have been transmitted to the printer.

```
int martel_write_rt(void *port,const void *buf,int size)
```

This function is similar to `martel_write()` but ignores any flow control condition. This function may be used to force sending data to the printer even if flow control line is asserted.

This function is used in conjunction with real-time commands (for example, hardware reset cmd "ESC @").

```
int martel_read(void *port,void *buf,int size)
```

Read `size` bytes of data from port. Data is stored in buffer `buf`.

This function blocks until all data have been read. This function also returns with `MARTEL_READ_TIMEOUT` if a read timeout has been set and timeout elapsed.

```
int martel_sync(void *port)
```

Wait until all data pending in output kernel buffer of printer port have been transmitted. This function also returns with `MARTEL_WRITE_TIMEOUT` if a write timeout has been set and timeout elapsed.

This function can be used after an `martel_write()` operation to ensure that the printer actually received and processed a command.

```
int martel_flush(void *port)
```

Clear input and output kernel buffers.

```
int martel_gets(void *port,void *s,int size)
```

Read a null-terminated string from port. String is stored in user-supplied buffer `s`. This function does not write more than `size` bytes into string buffer (including trailing zero). If return code is `MARTEL_OK`, this function guarantees that string stored in buffer is always null-terminated.

```
int martel_set_write_timeout(void *port,int ms)
```

Set write timeout of a printer port to `ms` milliseconds. Setting write timeout to zero disables timeout functionality.

```
int martel_set_read_timeout(void *port,int ms)
```

Set read timeout of a printer port to `ms` milliseconds. Setting read timeout to zero disables timeout functionality.

5.4.1. Serial port specific operations

The following port operations are only available for serial printer ports. Performing these operations on port types other than serial will yield an `MARTEL_INVALID_PORT_TYPE` error.

```
int martel_serial_set_baudrate(void *port,int baudrate)
```

Set serial port baudrate (one of `martel_serial_baudrate_t`).

```
int martel_serial_set_handshake(void *port,int handshake)
```

Set serial port handshake mode (one of `martel_serial_handshake_t`).

```
int martel_serial_get_baudrate(void *port)
```

Return current serial port baudrate (one of `martel_serial_baudrate_t`).

```
int martel_serial_get_handshake(void *port)
```

Return current serial port handshake mode (one of `martel_serial_handshake_t`).

5.4.2. Parallel port specific operations

Parallel port can be written in IRQ or polling mode. IRQ mode is faster but requires that an IRQ hardware be attached to the parallel port. Polling is more compatible and works in all cases, but it requires much more processing time.

```
int martel_parallel_reset(void *port)
```

Reset printer device.

```
int martel_parallel_set_mode(void *port,int mode)
```

Set parallel write mode (one of `martel_parallel_mode_t`).

```
int martel_parallel_get_mode(void *port)
```

Get parallel write mode (one of `martel_parallel_mode_t`).

6. Library error codes

Martel library error codes are negative integers, while zero indicates no error in most situations.

Error name	Description
MARTEL_OK	Operation succeeded.
MARTEL_NOT_IMPLEMENTED	Operation is not implemented.
MARTEL_IO_ERROR	A low-level system error occured while performing operation.
MARTEL_INVALID_MODEL	Model number is not valid (not in `models.def`).
MARTEL_INVALID_MODEL_TYPE	Model type is not valid (not in `martel_model_type_t`).
MARTEL_INVALID_PORT	Port structure is not valid.
MARTEL_INVALID_PORT_TYPE	Port type is not valid for required operation.
MARTEL_DEVICE_TOO_LONG	Device name is too long.
MARTEL_INVALID_URI	URI format is not valid.
MARTEL_INVALID_BAUDRATE	Required baudrate value is invalid (not in `martel_serial_baudrate_t`).
MARTEL_INVALID_HANDSHAKE	Required handshake mode is invalid (not in `martel_serial_handshake_t`).
MARTEL_INVALID_TIMEOUT	Required timeout value is invalid (probably negative).
MARTEL_OPEN_FAILED	Open operation failed.
MARTEL_CLOSE_FAILED	Close operation failed.
MARTEL_WRITE_FAILED	Write operation failed.
MARTEL_WRITE_TIMEOUT	Write operation timed out.
MARTEL_READ_FAILED	Read operation failed.
MARTEL_READ_TIMEOUT	Read operation timed out.
MARTEL_SYNC_FAILED	Synchronize operation failed.
MARTEL_FLUSH_FAILED	Flush operation failed.
MARTEL_INVALID_STATUS	Printer status data is not valid
MARTEL_PORT_NOT_OPEN	Port is not open and operation requires it to be open.
MARTEL_PORT_ALREADY_OPEN	Port is already open.
MARTEL_INVALID_PARALLEL_MODE	Parallel write mode is not valid (not in `martel_parallel_mode_t`).