# APS Linux Driver
# libaps Documentation

# 1. Table of contents



http://www.aps-printers.com/

## 2. Revision history

| Rev. | Date | Page | Revision item |
|---|---|---|---|
| A | 07/03/2006 | - | First issue |
| B | 24/04/2006 | - | Applies to driver rev. 0.2 and above |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 3. Introduction

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

While the APS library is used primarily by the APS CUPS driver, users might also use the APS library directly. The APS 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 APS library.

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

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

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

```
errnum = aps_get_error(port);
errnum = aps_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 = aps_write(port,buf,size))<0) {
    if (errnum==APS_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 APS 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 <aps/aps.h>

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

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

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

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

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

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

        /* destroy communication port */
        errnum = aps_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 `libaps`;
- `src/sample/sample2.c`: basic sample program using `libaps` and including error checking.

# 4. Printer port URI

Printer ports are identified within the APS 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.

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

Here are some examples of port URI for serial, parallel and USB printer ports.

```
aps:/dev/ttyS0?type=serial+baudrate=9600+handshake=rtscts
aps:/dev/parport0?type=parallel+mode=irq
aps:/dev/usb/lp0?type=usb
aps:/proc/bus/usb?type=usb+path=1,7
```

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<br>parallel<br>usb | Define type of printer port. |
| baudrate | serial | 9600 | 1200<br>2400<br>4800<br>9600<br>19200<br>38400<br>57600<br>115200 | Define default serial baudrate in bauds. |
| handshake | serial | rtscts | none<br>rtscts<br>xonxoff | Define default serial handshake mode. |
| mode | parallel | poll | poll<br>irq | Define parallel write mode. |
| path | usb | none | N/A | Describe physical connection path of the printer device. |

## 4.1. USB port URI details

Several URI formats can be used to locate USB printers within the system.

USB printers can be located using printer device files managed by the printer class driver of the kernel.

```
aps:/dev/usb/lp0?type=usb
```

While this solution is the easiest to use, it might not be reliable on systems with several USB printers.

Device files are allocated in the order in which printers are connected. `/dev/usb/lp0` is bound to the first USB printer connected to the system, `/dev/usb/lp1` is bound to the second and so on. There is no guarantee, however, that a specific device file will always point to the same printer. This means that `/dev/usb/lp0` might point to printer *X* at a given time, but might point to printer *Y* after a reboot, because printer *Y* has been enumerated before *X* for some reason.

The APS library offers an additionnal mechanism to locate USB devices. The APS library can identify devices by their *physical connection path*, which is a description of the chain of connectors that data needs to go through from the USB host controller to the USB device. This path does not change as long as the device remains connected to the same USB port.

```
aps:/proc/bus/usb?type=usb+path=1,7
```

Device name part of the URI indicates *usbfs* mount point (usually `/proc/bus/usb`). First number in the path indicates bus number to which the device is connected, while subsequent numbers describe connector port numbers.

```
aps:/proc/bus/usb?type=usb+path=2,4,8
```

This URI describes a USB device which is connected through a hub, because the path is three levels deep. The USB device is connected to port 8 of a hub, which is itself connected to port 4 of bus number 2.

# 5. Library functions

## 5.1. Error handling

```
int aps_get_error(void *port)
```

Return the last error that occured on `port`.

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

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

## 5.2. Printer models database

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

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

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

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

Return printable name of printer model.

```
int aps_get_model_type(int model)
```

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

```
int aps_get_model_width(int model)
```

Return dotline width in pixels of printer model.

```
int aps_decode_status(int type,const void *buf,int size,aps_status_t *status)
```

Decode internal status of printer into a standardized `aps_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 autodetection

The APS library provides an autodetection feature to discover printers attached to a system. This functionality is mainly used by the APS CUPS driver to simplify the process of adding a printer. It has limited use in the context of a customer application, because the number of printers and their references should be known at compile time.

```
int aps_detect_printers(aps_printer_t *printers,int max)
```

This function tries to detect APS printers currently attached to the system and returns the number of printers found. Maximum `max` printers are detected.

The application must be provide an array of `aps_printer_t` structures. Array must be at least `max` structures long. Upon return, the function fills as many structures as detected, starting from offset zero. Each structure stores printer model number, printer identity string and printer URI which can be used to create a port structure using `aps_create_port()`.

```
typedef struct {
        int     model;
        char    identity[APS_IDENTITY_MAX+1];
        char    uri[APS_URI_MAX+1];
} aps_printer_t;
```

## 5.4. 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 *aps_create_port(const char *uri)
```

Create a printer port from an URI.

```
void *aps_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 *aps_create_parallel_port(const char *device)
```

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

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

Create a USB printer port from device file name.

```
void *aps_create_usb_port_from_address(const char *usbfs,int busnum,int devnum)
```

Create a USB printer port. from bus number and device address.

`usbfs` describes mount point of the *usbfs* filesystem (usually `/proc/bus/usb`). `busnum` and `devnum` are, respectively, bus number and device number of the target printer device.

```
int aps_destroy_port(void *port)
```

Destroy a printer port.

```
int aps_get_port_type(void *port)
```

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

```
int aps_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).

## 5.5. 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 aps_open(void *port)
```

Open a printer port.

```
int aps_close(void *port)
```

Close a printer port..

```
int aps_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

`APS_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 `aps_sync()` to wait until all data have been transmitted to the printer.

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

This function is similar to `aps_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, command hardware reset "ESC @" of MRS, HRS and KCP printers).

```
int aps_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 `APS_READ_TIMEOUT` if a read timeout has been set and timeout elapsed.

```
int aps_sync(void *port)
```

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

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

```
int aps_flush(void *port)
```

Clear input and output kernel buffers.

```
int aps_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 `APS_OK`, this function guarantees that string stored in buffer is always null-terminated.

```
int aps_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 aps_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.

### 5.5.1. 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 `APS_INVALID_PORT_TYPE` error.

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

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

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

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

```
int aps_serial_get_baudrate(void *port)
```

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

```
int aps_serial_get_handshake(void *port)
```

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

### 5.5.2. 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 aps_parallel_reset(void *port)
```

Reset printer device.

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

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

```
int aps_parallel_get_mode(void *port)
```

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

### 5.5.3. USB port specific operations

```
int aps_usb_control(void *port,aps_usb_ctrltransfer_t *ctrl)
```

Perform a control request on a USB port.

This function can be used to get printer status in a truly asynchronous way, or to hardware reset a printer. Please check relevant section of the printer datasheet for more information.

A control transfer structure is passed to this function.

```
typedef struct {
        unsigned char   bRequestType;
        unsigned char   bRequest;
        unsigned int    wValue;
        unsigned int    wIndex;
        unsigned int    wLength;
        void *          data;
} aps_usb_ctrltransfer_t;
```

`data` points to a buffer of at least `wLength` bytes, and stores ancillay data required by the control request.

For control OUT requests (transfer direction from host to device), `data` buffer is transmitted along with the request. For control IN requests (transfer direction from device to host), `data` buffer is filled with data received from the device.

Set `data` and `wLength` to zero for control requests that do not require ancillary data.

## 6. Library error codes

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

| Error name | Description |
|---|---|
| APS_OK | Operation succeeded. |
| APS_NOT_IMPLEMENTED | Operation is not implemented. |
| APS_IO_ERROR | A low-level system error occured while performing operation. |
| APS_INVALID_MODEL | Model number is not valid (not in `models.def`). |
| APS_INVALID_MODEL_TYPE | Model type is not valid (not in `aps_model_type_t`). |
| APS_INVALID_PORT | Port structure is not valid. |
| APS_INVALID_PORT_TYPE | Port type is not valid for required operation. |
| APS_DEVICE_TOO_LONG | Device name is too long. |
| APS_INVALID_URI | URI format is not valid. |
| APS_INVALID_BAUDRATE | Required baudrate value is invalid (not in `aps_serial_baudrate_t`). |
| APS_INVALID_HANDSHAKE | Required handshake mode is invalid (not in `aps_serial_handshake_t`). |
| APS_INVALID_TIMEOUT | Required timeout value is invalid (probably negative). |
| APS_OPEN_FAILED | Open operation failed. |
| APS_CLOSE_FAILED | Close operation failed. |
| APS_WRITE_FAILED | Write operation failed. |
| APS_WRITE_TIMEOUT | Write operation timed out. |
| APS_READ_FAILED | Read operation failed. |
| APS_READ_TIMEOUT | Read operation timed out. |
| APS_SYNC_FAILED | Synchronize operation failed. |
| APS_FLUSH_FAILED | Flush operation failed. |
| APS_INVALID_STATUS | Printer status data is not valid |
| APS_PORT_NOT_OPEN | Port is not open and operation requires it to be open. |
| APS_PORT_ALREADY_OPEN | Port is already open. |
| APS_INVALID_PARALLEL_MODE | Parallel write mode is not valid (not in `aps_parallel_mode_t`). |
| APS_INVALID_USB_PATH | USB connection path is not valid (format is incorrect, or no device is connected). |
| APS_USB_DEVICE_NOT_FOUND | USB device cannot be found in list of connected devices. |
| APS_USB_DEVICE_BUSY | USB device is currently registered by a driver and this driver cannot be automatically unregistered. |