SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

Getting started with SPBUS (Application Programming) Interfaces

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

Purpose

The purpose of this document is to guide a developer to one of the SPBUS device interfaces most fit for their purpose. For example a developer might want to integrate SPBUS devices into their already existing modbus network and retrieve SPBUS device information using it. In such a case it's obvious to refer to the 'Schleifenbauer hPDU Modbus Specification' document. However, it might be the case there is no predetermined protocol (or programming language) to be used and it is up to the developer what to use. In such a case, this document might clarify the different available interfaces and APIs, and allow the developer to choose which is most fit for their purpose.

Preliminary notes

- This document is excessively practical with interface examples. For more details on almost every interface we will refer to the designated document with a more detailed explanation on the interface.
- The old software APIs each differs in implementation. For example, the php API uses different calls and its design differs from, say, the perl API.
- The new HLAPI is designed with consistency in mind and all (method/function) calls will be the same regardless of programming language (assuming there exists a implementation for the programming language).
- Every API implements a protocol. An interface, however, might already be the used protocol already. For example, the php API implements the IPAPI protocol to communicate with SPBUS devices (refer to the SPBUS document v2 for more information) whilst the SPBUS SNMP interface already defines its protocol, namely, SNMP.
- The protocols available and used are:
    - SPBUS Databus protocol
    - IPAPI protocol
    - WEBAPI
    - Modbus
    - SNMP
- Different permissions are mapped to different protocols i.e. some register can be read by one protocol while the other cannot. Please refer to the SPDM for these permissions.

Php API

The current php API uses a single class for interfacing with SPBUS devices. It implements only the IPAPI protocol. For more information on the used protocol refer to the 'SPBUS protocol v2' document, for more information on the php API download the API from the Schleifenbauer website and investigate functionality using the examples and source code.

Php API example: reading firmware version from SPBUS device with unit address 12

```php
<?php
/*****************************************************************

PHP example for Schleifenbauer PDUs using the SPBus Class
read firmware version of a device with unit address 12

Schleifenbauer Holding B.V.
All rights reserved, no warranties

*****************************************************************/
/*****************************************************************/
require_once("../Sources/class.spbus.php");
/*****************************************************************/
// "Input" variables
$GatewayIP = "127.0.0.1";
$UnitAddress = 12;
$GatewayPort = 7783;

// Initialize
$SPbus = new SPbus("../Sources/datamodel", "../Sources/errors");
$SPbus->setGatewayIPAddress($GatewayIP);
$SPbus->setGatewayPort($GatewayPort);
$SPbus->Connect();

// Retrieve firmware version of device and print it on the webpage.
print "<html><head></head><body><h1>Example</h1><h2>device's with Firmware
version on device with ip $GatewayIP</h2><ul>";
$FWversion = $SPbus->getPDUFirmwareVersion($UnitAddress);
print "<li>device $UnitAddress has Firmware Version
$FWversion</li></ul><p></body></html>";
?>
```

Php API example: switching on outlet 3 of device with unit address 99

```php
<?php
/**********************************************************************

PHP example for Schleifenbauer PDUs using the SPBus Class
switch on the channel 3 outlet of device with unit address 99

Schleifenbauer Holding B.V.
All rights reserved, no warranties

**********************************************************************/

/**********************************************************************/
require_once("../Sources/class.spbus.php");
/**********************************************************************/
// "Input" variables
$GatewayIP = "127.0.0.1";
$UnitAddress = 99;
$GatewayPort = 7783;
$outletChannel = 3;

// Initialize
$SPbus = new SPbus("../Sources/datamodel", "../Sources/errors");
$SPbus->setGatewayIPAddress($GatewayIP);
$SPbus->setGatewayPort($GatewayPort);
$SPbus->Connect();

// Switch on outlet 3
print "<html><head></head><body><h1>Example</h1><h2>switiching on outlet 3 of
device with ip $GatewayIP</h2><ul></ul><p></body></html>";
$SPbus->setPDUOutletUnlock($GatewayIP, $outletChannel);
$SPbus->setPDUOutletStateOn($GatewayIP, $outletChannel);
?>
```

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

Php API example: scanning the SPBUS for all reachable devices

```php
<?php
/*********************************************************************

PHP example for Schleifenbauer PDUs using the SPBus Class
scan the spbus for devices

Schleifenbauer Holding B.V.
All rights reserved, no warranties

*********************************************************************/

/*********************************************************************/
require_once("../Sources/class.spbus.php");
/*********************************************************************/
// "Input" variables
$GatewayIP = "127.0.0.1";
$GatewayPort = 7783;

// Initialize
$SPbus = new SPbus("../Sources/datamodel", "../Sources/errors");
$SPbus->setGatewayIPAddress($GatewayIP);
$SPbus->setGatewayPort($GatewayPort);
$SPbus->Connect();

// Scan spbus
$FoundIDs = $SPbus->ScanBus();

// Print retrieved devices
print "<html><head></head><body><h1>Example</h1><h2>Scan \
results on spbus device $GatewayIP</h2><ul>";

if ((isset($FoundIDs)) && (!empty($FoundIDs)))
{
    foreach ($FoundIDs as $id => $value)
    {
        print "<li>PDU " . $value . "</li>";
    }
}
print "</ul><p></body></html>";
?>
```

Perl API

The perl API uses a single package for interfacing with SPBUS devices. It implements both the SPBUS protocol and the IPAPI protocol. For more information on the used protocols refer to the 'SPBUS protocol v2' document, for more information on the perl API download the API from the Schleifenbauer website and investigate functionality using the examples and source code.

Perl API example: reading firmware version from SPBUS device with unit address 12

```
#!/usr/bin/perl

# (c) Schleifenbauer Holding BV
#
# read firmware version of a device with Unit Address 12
#

use strict;
use spapi qw( InitSpbus ReadRegister );

# "Input" variables
my $hostip = "192.168.1.200";
my $com = "";
my $port = 7783;
my $key = "0000000000000000"; # default ipapi key
my $type = 1; # gateway (2 = RS485/TCP converter)
my $unitaddress = 12;
my $register = "idfwvs";

my $channel = 0;
my $v = 0;

# Initialize
my $device = InitSpbus($type, $com, $hostip, $port, $key);

# Retrieve and print firmware version of device
my $res = ReadRegister($device, $unitaddress, $register, \$v, $channel);
if($res>0)
    {print "read succeeded: $v\n"}
else
    {print "read failed!\n"}
```

SCHLEIFENBAUER

LIVING FOR THE POWER TO DELIVER

Perl API example: switching on outlet 3 of device with unit address 99

```
#!/usr/bin/perl

# (c) Schleifenbauer Holding BV, Alain Schuermans 2008..2016
#
# switch on the channel 3 outlet of device with unit address 99
#

use strict;

use spapi qw( InitSpbus WriteRegister );

my $address = 99;
my $on = 0x01;
my $outlet = 3;
my $unlock = 2081;
my $on = 2000;

my $intf = InitSpbus( 1, "", "192.168.1.3", 7783, "" );

if ( $intf >= 0 ) {
    # unlock outlet 3
    my $res = WriteRegister($intf, $address, 'swounl', \$on, $outlet);
    # perform switch action
    $res = WriteRegister($intf, $address, 'swocst', \$on, $outlet);
}
```

SCHLEIFENBAUER

LIVING FOR THE POWER TO DELIVER

Perl API example: scanning the SPBUS for all reachable devices

```perl
#!/usr/bin/perl

# (c) Schleifenbauer Holding BV
#
# scan the spbus for devices
#

use strict;
use Time::HiRes qw/time usleep/;

use spapi qw( InitSpbus ScanBus );

my @rs_units = ();
my @rs_devices_id1 = ();
my @rs_devices_id2 = ();
my @rs_devices_id3 = ();

my $intf = InitSpbus( 5, "", "192.168.1.200", 7783, "" );

if ( $intf >= 0 ) {
    my $startTime = time();

    my $res = ScanBus( $intf, \@rs_units, \@rs_devices_id1, \@rs_devices_id2,
\@rs_devices_id3 );
    my $duration = time() - $startTime;
    print "duration : $duration \n";
    print "found: $res devices\n";
    my $i = 0;
    foreach (@rs_units) {
        if ($_) {
            $i++;
            print "Unit: # $rs_units[$i], HWID
$rs_devices_id1[$i]-$rs_devices_id2[$i]-$rs_devices_id3[$i] \n";
        }
    }
}
```

Webapi

The webapi behaves similarly to a RESTful API. It is possible to retrieve a register's information from a SPBUS device using resource requests. The webapi does not depend upon a specific implementation and is therefore explained using the messages send over to the SPBUS device over the TCP/IP connection. Note that details (namely ones related to the Authentication token) are omitted from this document. For more information on the webapi refer to the webapi documentation.

Webapi step-by-step example: reading firmware version from SPBUS device with host ip 192.168.9.137

1. Retrieve ‘userid’ and device ‘time’ using the /userid resource. This information is used to construct an authentication token used for further webapi communications. Details on token construction can be found in the webapi documentation. For convenience sake we will assume the token 'aabbccddeeff1122'.

Example request message
POST /userid HTTP/1.1 Host: 192.168.9.137 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Content-Length: 10 user=super

Example response message
HTTP/1.1 200 OK Content-type: application/x-www-form-urlencoded Connection: keep-alive Content-Length: 19 userid=0&time=23386

2. Retrieve firmware version information of device using the /register/<mnemonic> resource. The firmware version information mnemonic is ‘idfwvs’. Note that (1) authentication is done by sending an authentication token in the header, and (2) the Transfer-Encoding is chunked and therefore sends a hexadecimal 0xb to denote the size of the chunk.

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

Example request message
GET /register/idfwvs HTTP/1.1 Host: 192.168.9.137 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Authorization: hPDU-auth-v1 aabbccddeeff1122

Example response message
HTTP/1.1 200 OK Content-type: application/x-www-form-urlencoded Cache-Control: no-cache, no-store Connection: keep-alive Transfer-Encoding: chunked b &idfwvs=236 0

3. Parse the data properly ensuring that the chunked transfer encoding is obeyed.

Webapi example: switching on outlet 3 of a device with host address 192.168.9.137

In this example we won't go into details as much as previous example. We merely show what to write to what register to enable an outlet to be switched on.

1. Retrieve information necessary to create authentication token

Example request message
POST /userid HTTP/1.1 Host: 192.168.9.137 libssh2/1.6.0 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Content-Length: 10 user=super

Example response message
HTTP/1.1 200 OK Content-type: application/x-www-form-urlencoded Cache-Control: no-cache, no-store Connection: keep-alive Content-Length: 19 userid=0&time=19197

2. Set the unlock register of outlet 3 so we can change its state

Example request message
POST /register/swounl_3 HTTP/1.1 Host: 192.168.9.137 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Authorization: hPDU-auth-v1 aabbccddeeff1122 Content-Length: 10 swounl_3=1

Example response message

```
HTTP/1.1 200 OK
Content-type: application/x-www-form-urlencoded
Cache-Control: no-cache, no-store
Connection: keep-alive
Transfer-Encoding: chunked

9f
&swounl_1=0&swounl_2=0&swounl_3=1&swounl_4=0&swounl_5=0&sw
ounl_6=0&swounl_7=0&swounl_8=0&swounl_9=0&swounl_10=0&swoun
l_11=0&swounl_12=0&swounl_13=0&swounl_14=0
9c
&swounl_15=0&swounl_16=0&swounl_17=0&swounl_18=0&swounl_19=
0&swounl_20=0&swounl_21=0&swounl_22=0&swounl_23=0&swounl_24
=0&swounl_25=0&swounl_26=0&swounl_27=0
a8
&swounl_28=0&swounl_29=0&swounl_30=0&swounl_31=0&swounl_32=
0&swounl_33=0&swounl_34=0&swounl_35=0&swounl_36=0&swounl_37
=0&swounl_38=0&swounl_39=0&swounl_40=0&swounl_41=0
9c
&swounl_42=0&swounl_43=0&swounl_44=0&swounl_45=0&swounl_46=
0&swounl_47=0&swounl_48=0&swounl_49=0&swounl_50=0&swounl_51
=0&swounl_52=0&swounl_53=0&swounl_54=0
0
```

3. Switch the state of outlet 3 to ‘on’

Example request message

```
POST /register/swocst_3 HTTP/1.1
Host: 192.168.9.137
Accept: */*
Accept-Encoding: gzip, deflate
Content-Type:application/x-www-forms-urlencoded
Connection:keep-alive
Authorization: hPDU-auth-v1 aabbccddeeff1122
Content-Length: 10

swocst_3=1
```

Example response message

```
HTTP/1.1 200 OK
Content-type: application/x-www-form-urlencoded
Cache-Control: no-cache, no-store
Connection: keep-alive
Transfer-Encoding: chunked
```

9f
&swocst_1=1&swocst_2=1&swocst_3=1&swocst_4=1&swocst_5=1&swocst_6=1&swocst_7=1&swocst_8=1&swocst_9=1&swocst_10=1&swocst_11=1&swocst_12=1&swocst_13=1&swocst_14=1
9c
&swocst_15=1&swocst_16=1&swocst_17=1&swocst_18=1&swocst_19=0&swocst_20=0&swocst_21=0&swocst_22=0&swocst_23=0&swocst_24=0&swocst_25=0&swocst_26=0&swocst_27=0
a8
&swocst_28=0&swocst_29=0&swocst_30=0&swocst_31=0&swocst_32=0&swocst_33=0&swocst_34=0&swocst_35=1&swocst_36=0&swocst_37=0&swocst_38=0&swocst_39=0&swocst_40=0&swocst_41=0
9c
&swocst_42=0&swocst_43=0&swocst_44=0&swocst_45=0&swocst_46=0&swocst_47=0&swocst_48=0&swocst_49=0&swocst_50=0&swocst_51=0&swocst_52=0&swocst_53=0&swocst_54=0
0

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

Webapi example: scanning the SPBUS for all reachable devices

In this example we won't go into details as much as previous example.

1. Retrieve information necessary to construct authentication token

Example request message
POST /userid HTTP/1.1 Host: 192.168.9.137 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Content-Length: 10 user=super

Example response message
HTTP/1.1 200 OK Content-type: application/x-www-form-urlencoded Cache-Control: no-cache, no-store Connection: keep-alive Content-Length: 20 userid=0&time=108097

2. Request scan and retrieve response

Example request message
POST /scan HTTP/1.1 Host: 192.168.9.137 Accept: */* Accept-Encoding: gzip, deflate Content-Type:application/x-www-forms-urlencoded Connection:keep-alive Authorization: hPDU-auth-v1 aabbccddeeff1122 Content-Length: 10 user=super

Example response message

[1] value *n* depends on the amount of units found. This value can be read from register 40000, 'honruf'. Please refer to the SPDM for more information.

```
HTTP/1.1 200 OK
Content-type: application/x-www-form-urlencoded
Cache-Control: no-cache, no-store
Connection: keep-alive
Transfer-Encoding: chunked

29
scan_addr_1=666&scan_hwid_1=42534-6129-0&
29
scan_addr_2=123&scan_hwid_2=57425-5912-0&
27
scan_addr_3=99&scan_hwid_3=6566-6021-0&
26
scan_addr_4=40&scan_hwid_4=125-6129-0&
c
scan_total=4
0
```

Modbus

Modbus is a communication protocol which allows different commands (functions) including reads/writes to and from device registers (memory locations on a device in the modbus network). Modbus is therefore useful for SPBUS device communications considering SPBUS devices use registers to interface with its hardware. For more information on modbus refer to the official specification here and for more information on SPBUS device data over modbus, refer to the 'Schleifenbauer hPDU Modbus Specification' document.

Modbus example: reading firmware version from SPBUS device with unit address 12

Note that sizes are in each 16-bits. I.e. size of 2 is actually 32-bytes.

Modbus/TCP address	192.168.9.137
Device address	12
Modbus function code (Read Holding Register)	0x03
Start address	102
Size in modbus	1
Result	Register value, size 1

Modbus example: switching on outlet 3 of a device with unit address 99

Note that sizes are in each 16-bits I.e. size of 2 is actually 32-bytes

Modbus/TCP address	192.168.9.137
Device address	99
Modbus function code (Write Single Register)	0x06
Start address	2084
Size in modbus	1
Data	1

Modbus example: scanning the SPBUS for all reachable devices

1. Invoke a scan

Note that sizes are in each 16-bits I.e. size of 2 is actually 32-bytes

Modbus/TCP address	192.168.9.137
Modbus function code (Write Single Register)	0x06
Start address	40100
Size in modbus	1
Data	1

2. Read all devices

Note that sizes are in each 16-bits I.e. size of 2 is actually 32-bytes

Modbus/TCP address	192.168.9.137
Modbus function code (Read Holding Registers)	0x03
Start address	40200, 40712, 41224, 41736
Size in modbus[1]	*n*
Result	Register values, size *n*

SNMP

SNMP can be used to communicate with a SPBUS device's registers. At the time of writing this document, SNMP v1, v2, and v2c are supported. Also supported are SNMP traps, for which a trap destination server can be configured on a device. The current SPBUS device traps are:

- Network connectivity
- SNMP authentication failure
- Device status code
- Temperature alert
- Input current alert
- Output current alert
- Input voltage alert
- Output current drop alert
- Input current drop alert
- Sensor change alert
- Ring state changed

All of these SNMP traps can each be enabled or disabled.

For SNMP OID (object identifier) information, the MIB (management information base) files can be found on the Schleifenbauer website. Please refer to the SNMP RFCs for more information on SNMP.

SNMP example: reading firmware version from SPBUS device with unit address 12

Here’s an example of all information necessary to request the firmware version of a SPBUS device with unit address 12. Note that the unit address is included in the OID.

(Master) device IP	192.168.9.137
SNMP version	2c
Community string	public
OID (Text)	sdbDevIdFirmwareVersion.12
OID (Numeric)	.1.3.6.1.4.1.31034.12.1.1.2.1.1.1.2.12

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

SNMP example: switching on outlet 3 of device with unit address 99

Here's an example of all information necessary to switch on the 3rd outlet of device with unit address 99. Note that the unit address and outlet channel is included in the OID.

1. Unlocking outlet

(Master) device IP	192.168.9.137
SNMP version	2c
Community string	private
OID (Text)	sdbDevOutSwUnlock.99.3
OID (Numeric)	.1.3.6.1.4.1.31034.12.1.1.2.7.3.1.4.99.3
Set value	1

2. Switching to 'on'

(Master) device IP	192.168.9.137
SNMP version	2c
Community string	private
OID (Text)	sdbDevOutSwCurrentState.99.3
OID (Numeric)	.1.3.6.1.4.1.31034.12.1.1.2.7.3.1.2.99.3
Set value	1

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

SNMP example: scanning the SPBUS for all reachable devices

1. Invoke a scan

(Master) device IP	192.168.9.137
SNMP version	2c
Community string	private
OID (Text)	sdbMgmtCtrlScan.0
OID (Numeric)	.1.3.6.1.4.1.31034.12.1.1.1.2.1.0
Set value	1

2. Read all devices (note: retrieve the whole subtree)

(Master) device IP	192.168.9.137
SNMP version	2c
Community string	public
OID (Text)	sdbMgmtCtrlDevicesEntry
OID (Numeric)	.1.3.6.1.4.1.31034.12.1.1.1.2.4.1

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

SPBUS

A user might want to implement their own API or directly integrate SPBUS protocol communications in their custom software. For these people there exists the SPBUS protocol description. This document describes the SPBUS protocol in detail. Note that the SPBUS protocol can be used either over RS485, or IPAPI although IPAPI uses additional packet construction steps.

SPBUS protocol example: reading firmware version from SPBUS device with unit address 12

In this example we show the package construction using information from the SPBUS protocol document. We first explain how normal SPBUS packets ought to be constructed (such a packet goes over RS485), then we explain how to use a normal SPBUS packet and construct a IPAPI packet.
The variable information given is:

Field name	**Description**	**Example value used**
Command	Which command we're sending (in this case a 'Read_register' command)	1 = 0x01
Address	The unit address of the device	12 = 0x000c
Identifier	Transaction identifier, the identifier of the message.	1 = 0x0001
Register_start	The register start address	102 = 0x0066
Register_length	The amount of register data to read	2 = 0x0002

Variable information	**Example value used**
ARC4 encryption key	0000000000000000

Request example (RS485)

- Packet fields:
  [Packet_start.STX][Command.Read_register_layer_1][Address][Identifier][Register_start][Register_length][CRC][Packet_end.ETX]
- Packet (delimited): [0x02][0x01][0x000C][0x0001][0x0066][0x0002][0x98AE][0x03]
- Packet (delimited, little-endian):
  [0x02][0x01][0x0C00][0x0100][0x6600][0x0200][0xAE98][0x03]
- Packet: 0x02010C00010066000200AE9803