SCHLEIFENBAUER

LIVING FOR THE POWER TO DELIVER

WEBAPI DOCUMENT

SCHLEIFENBAUER
LIVING FOR THE POWER TO DELIVER

1. Table of Contents

2. Authentication

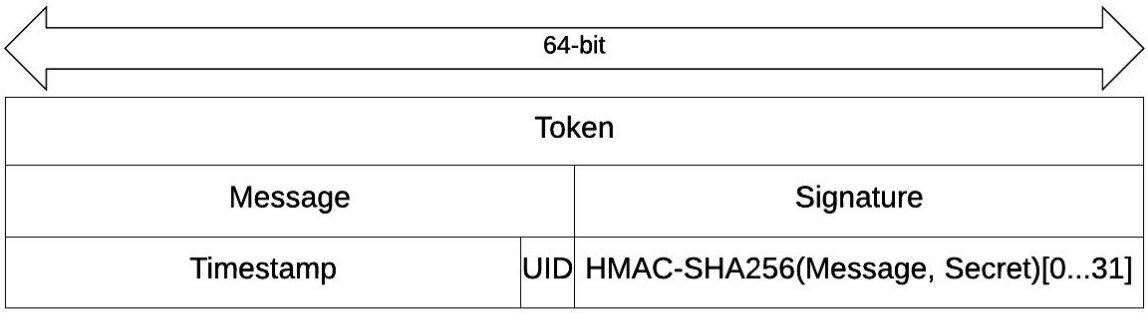
Currently, the WEBAPI uses a custom authentication method to authenticate and allow WEBAPI resource requests. This authentication method requires the caller to include an authentication header in requests which has the following format:
Authorization: <hPDU authentication version> <16 digit hexadecimal token (64-bit number)>

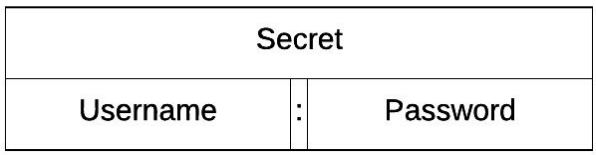
Example:
Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

At the time of writing, the only used hPDU authentication version is v1 ('hPDU-auth-v1').

2.1. hPDU-auth-v1 token

For all WEBAPI resource requests related to a spbus devices' data/information, an authentication header with token must be included in the request. Construction of this token uses username, user identifier, password, and timestamp of which the user identifier and timestamp must be retrieved from the server (see chapter 'Resources'). The token's format looks as follows (**note: the token and every field within it is big-endian encoded**):





- A **Token** is a 64-bit number. The first 32-bits consists of a **Message** and the remaining 32-bits consists of a **Signature**.
- A **Message** is a 32-bit number. The first 29-bits consists of the **Timestamp**, the remaining 3-bits consists of the **User identifier (UID)**.
- A **Signature** is a 32-bit number made using the first 32-bits of a HMAC-SHA256 hash using the **Message** as the data and **Secret** as the secret key.
- A **Timestamp** is retrieved from the server (using the '/userid' resource) and then kept in sync by the requester. The number represents the time of the server in seconds and is

only allowed to deviate 10 seconds from the server's internal clock (+-10 seconds). This means the requester must use the number and increment it every second.

- A **User identifier (UID)** is retrieved from the server (using the '/userid' resource). This number will always be within the 3-bit constraint set by the UID field.
- A **Secret** is string of format "Username:Password" (colon included). Both Username and Password are only allowed to be 16 characters long (I.e. with the colon this can add up to 33 characters).

Construction example:

- Input:
  - Username: foo
  - Password: bar
  - Timestamp: 0x1bd5b7dd
  - UID: 7
- **Secret:** "foo:bar"
- **Message:** 0xdeadbeef
- **HMAC-SHA256(Message = 0xdeadbeef, Secret = "foo:bar"):** 0x5d1fa4d6a66f2829043b22def8d56873c54b9fab1df4d4540ce800035ed86538
- **Signature:** 0x5d1fa4d6
- **Token:** 0xdeadbeef5d1fa4d6

3. HTTP considerations

3.1. Content

The content type used for the majority of Resources (both requests and responses) will be of type application/x-www-form-urlencoded. With this content type the keys will always be a mnemonic or descriptive alternative (e.g. 'scan_addr' denoting the address of a device found using the scan resource request). Please refer to the RFC for special character encodings.

3.2. Transmission encoding

Currently, the encoding used for the resource responses is HTTP/1.1 chunked encoding. This is present in the response header and should be respected when parsing.

3.3. HTTP response codes

This chapter explains the HTTP response codes and their meaning.

HTTP response code	Description
200	Successful request.
400	Bad request, client did not abide to the API
401	Unauthorized, authentication error. Token invalid (e.g. time is out of bounds or credentials are wrong).
403	Forbidden, permission error. Current user is not allowed to do that action (highest user role level can still receive this if, for example, a write-only register is written to).
404	Not found, resource cannot be found on the device (e.g. it simply does not exist).
405	Method not allowed, client did not abide to the API's command limitations (e.g. trying to GET a only-POST-able resource.)
422	Unprocessable entity, (invalid) value was rejected by server.
500	Internal server error, request couldn't be processed due to a device internal error e.g. due to a lack of memory.

3.4. Result codes

Some resources' responses include a mnemonic "result". This response mnemonic is a virtual mnemonic (i.e. does not really exist on the device as mnemonic). The result has the following meaning(s).

Result code	Description
OK	Successful request.
ERR_PARTIAL	Only a part of the response is given. The given response will be up until a certain point when an error occurred. The body should be ignored
ERR_WOULDLOCKOUT	The request would lock out the user that made the request from the device if the requests was accepted.
ERR_IPAPI_KEY_LENGTH_16	The length of the IPAPI key the user is trying to configure is either greater than or less than 16 characters.

4. Resources

4.1. Bridging over Databus (BoD)

As it's possible multiple devices exists on a single spbus network it might be desired to requests data from these devices. To do this using the webapi we use something called Bridging over Databus (BoD). To use BoD we need (1) the webapi resource to support BoD, and (2) a prefix with the following format:

/databus/<unit_address>/<Resource>

Where <unit_address> denotes the unit address of the device we try to reach (note: a unit address of 0 refers to the current device) and <Resource> is the resource we try to request. Note that the <Resource> ought to support BoD, otherwise behaviour will be undefined.

Example construction (read value of "stdvnm" on device with unit address 27):
/databus/27/register/stdvnm

4.2. Multi-channel mnemonic construction

Most registers can be reached by using the mnemonics as expected meaning that their functionality only depends on the use of the 6-character mnemonic. However, some registers consist out of multiple repeats such as the 'stolnm' register which has 27 repeats plus another 27 extended repeats totalling up to 54 repeats. To approach registers like these the following mnemonic construction is used:

<mnemonic>_<repeat number>

Concrete example:

stolnm_30

4.3. Resource descriptions

4.3.1. POST /userid

Resource:	/userid
Methods:	POST
Request parameters (path):	-
Request parameters (body):	1. 'user='string, username of the requester. Response will include the user identifier associated with the

	username. If there exists no user with username as given by parameter, then there'll still be a reply but authentication will fail.
Response parameters:	1. 'userid='int, the user identification number. This number ought to be used to construct the authentication token. 2. 'time='int, the current timestamp of the server. Must be kept in sync and used to construct the authentication token.
Description:	Retrieve information necessary to construct an authentication token for all remaining resource requests. This request will always return a userid regardless of whether the request parameter can be found. **Note that this resource does not need any authentication to retrieve.**
BoD support:	No
Example request:	POST /userid HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Content-Length: 10 user=super

4.3.2. GET /readable

Resource:	/readable
Methods:	GET
Request parameters (path):	-
Request parameters (body):	-
Response parameters:	All mnemonics that the user is allowed to read from in the format <mnemonic>'=true'. This format is repeated for all readable mnemonics.
Description:	Retrieve all register mnemonics of registers that the current user is allowed to read from. Note that if used in conjunction with BoD it might display incorrect permissions as BoD is technically over databus.

BoD support:	Yes
Example request:	GET /readable HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.3. GET /writable

Resource:	/writable
Methods:	GET
Request parameters (path):	-
Request parameters (body):	-
Response parameters:	All mnemonics that the user is allowed to write to in the format <mnemonic>'=true'. This format is repeated for all writable mnemonics.
Description:	Retrieve all register mnemonics of registers that the current user is allowed to write to. Note that if used in conjunction with BoD it might display incorrect permissions as BoD is technically over databus.
BoD support:	Yes
Example request:	GET /writable HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.4. GET /register/<mnemonic>

Resource:	/register/<mnemonic>
Methods:	GET
Request parameters (path):	1. <mnemonic>'='value, a register's mnemonic. Please refer to the SPDM for all register mnemonics available.
Request parameters (body):	-

Response parameters:	1. <mnemonic>'='value, the current value of the requested register. The actual datatype depends on the requested mnemonic. 2. (Optional) <mnemonic>'='value, more mnemonics returned with their current value because they're affected by this API call or the requested register is a multiple channel register.
Description:	Read register data. Response might contain more mnemonics depending on the requested mnemonic. The additional mnemonics added might be due to the register being a multiple repeats register or the register affected other registers which the caller must be notified of. Only registers the current user is allowed to read can be read.
BoD support:	Yes
Example request:	GET /register/ssstat HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.5. POST /register/<mnemonic>

Resource:	/register/<mnemonic>
Methods:	POST
Request parameters (path):	1. <mnemonic>'='value, a register's mnemonic. Please refer to the SPDM for all register mnemonics available.
Request parameters (body):	1. <mnemonic>'='value, the new value to be assigned to the register. The actual datatype depends on the requested mnemonic.
Response parameters:	1. <mnemonic>'='value, the new value assigned to the register. The actual datatype depends on the requested mnemonic. 2. (Optional) <mnemonic>'='value, more mnemonics returned with their current value because they're affected by this API call.
Description:	Write register data. A successful write does not imply the intended value was written. It might be possible that value is cut off (only a part of a string is send), hence the newest current value of the register is put in the response. Only

	registers the current user is allowed to write can be written to.
BoD support:	Yes
Example request:	POST /register/stdvnm HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Authorization: hPDU-auth-v1 deadbeef5d1fa4d6 Content-Length: 16 stdvnm=SP_Rack_5

4.3.6. GET /group/<group_name>

Resource:	/group/<group_name>
Methods:	GET
Request parameters (path):	1. <group_name>'='string, the group name of a group of registers related to each other. Please refer to the SPDM for the group names and their registers.
Request parameters (body):	-
Response parameters:	1. <mnemonic>'='value, the value of the register. The actual datatype depends on the register itself. 2. (Optional) <mnemonic>'='value, more mnemonics returned with their current value.
Description:	Retrieves all readable register within a register group. The group 'all' can be used to retrieve every readable register available.
BoD support:	No
Example request:	GET /group/identification HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.7. POST /save/<save_group>

Resource:	/save/<save_group>
Methods:	POST
Request parameters (path):	1. String <save_group>, the different save groups available. Subset of the register groups available. The usable groups are: • ethernet • ethernet/access • ethernet/mode • snmp • snmp/traps • modbus • http • ipapi • user
Request parameters (body):	Depends on the <save_group> value. Must contain only mnemonics of registers in <save_group>. All mnemonics must be filled in.
Response parameters:	Depends on the <save_group> value. Will contain the new values of all mnemonics of <save_group>[1] and the 'result' mnemonic. Refer to the '/save/<save_group> parameters' chapter for the groups' parameters.
Description:	Updates the values of whole register blocks at once.
BoD support:	No
Example request:	POST /save/ipapi HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Authorization: hPDU-auth-v1 deadbeef5d1fa4d6 Content-Length: 32 iaenab=1&iarc4k=foobarfizzzbuzzz

[1] Except the user group. This resource depends on virtual mnemonics not defined in the SPDM.

4.3.8. GET /ui/status

Resource:	/ui/status
Methods:	GET
Request parameters (path):	-
Request parameters (body):	-
Response parameters:	1. 'curr_page='int, current page visible on the display. 2. 'num_pages='int, total pages available on the display. 3. 'backlight_state'={'on', 'off', 'blinking'}, denotes the state of the backlight.
Description:	Retrieve user interface status information. Namely the current page, total number of pages and the backlight status.
BoD support:	No
Example request:	GET /ui/status HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.9. POST /ui/status

Resource:	/ui/status
Methods:	POST
Request parameters (path):	-
Request parameters (body):	1. 'curr_page='int, changes the current visible page to the number put into the body.
Response parameters:	
Description:	1. 'curr_page='int, (new) current page visible on the display. 2. 'num_pages='int, total pages available on the display. 3. 'backlight_state'={'on', 'off', 'blinking'}, denotes the state of the backlight.

BoD support:	No
Example request:	POST /ui/status HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Authorization: hPDU-auth-v1 deadbeef5d1fa4d6 Content-Length: 11 curr_page=3

4.3.10. GET /ui/framebuffer

Resource:	/ui/framebuffer
Methods:	GET
Request parameters (path):	-
Request parameters (body):	-
Response parameters:	Raw data.
Description:	Retrieve the raw (display) framebuffer of a device (1024 bytes). Note that the Content-type of the return will not be application/x-www-form-urlencoded. The raw framebuffer will be of the following format: ● Each line of the framebuffer array is a vertical line of 8 pixels ● The LCD is 128 pixels in width and 64 in height, hence after array index 128 comes the next row. ● The HSB of the byte is the highest y-axis of the vertical pixels.
BoD support:	No
Example request:	GET /ui/framebuffer HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.11. POST /ui/buttonpress

Resource:	/ui/buttonpress
Methods:	POST
Request parameters (path):	-
Request parameters (body):	1. 'button_1='int, 1 indicates a short press, 2 indicates a long press and 0 indicates no press at all. 2. 'button_2='int, 1 indicates a short press, 2 indicates a long press and 0 indicates no press at all.
Response parameters:	A reply as expected from GET /ui/status
Description:	Used to simulate a button press.
BoD support:	No
Example request:	POST /ui/buttonpress HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Authorization: hPDU-auth-v1 deadbeef5d1fa4d6 Content-Length: 10 button_1=1

4.3.12. POST /scan

Resource:	/scan
Methods:	POST
Request parameters (path):	-
Request parameters (body):	1. 'user='string, the username of the requester.
Response parameters:	1. scan_addr_<int>'='int, shows the unit address of one of the scanned devices. The <int> value starts at 1 and increments as more devices are found using the scan. 2. 'scan_hwid_'<int>'='int, shows the hardware identifier of one of the scanned devices.The <int> value starts at 1 and increments as more devices are found using the scan.

	3. ‘scan_total=’int, the total amount of devices found during the scan.
Description:	Do a scan on the spbus network and find all reachable devices on it.
BoD support:	No
Example request:	POST /scan HTTP/1.1 Host: 192.168.9.137 Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */* Content-type: application/x-www-form-urlencoded Authorization: hPDU-auth-v1 deadbeef5d1fa4d6

4.3.13. POST /address/<hardware_id>

Resource:	/address/<hardware_id>
Methods:	POST
Request parameters (path):	1. <hardware_id>, the hardware id of a device as 3 integer number delimited by dashed (e.g. 42534-6129-0).
Request parameters (body):	1. ‘idaddr=’int, the unit address to be assigned to the device with hardware id <hardware_id>.
Response parameters:	-
Description:	Change the unit address of a device with hardware id <hardware_id>.
BoD support:	No
Example request:	POST /address/42534-6129-0 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 deadbeef5d1fa4d6 Content-Length: 10 idaddr=123

4.4. /save/<save_group> parameters

Note that the parameters are mnemonics for device registers defined in the SPDM. For more information refer to the SPDM.

4.4.1. /save/ethernet/access

- etaips_1, first allowed ip address
- etaips_2, second allowed ip address
- etaips_3, third allowed ip address
- etaipm_1, first allowed ip address' netmask in CIDR notation
- etaipm_2, second allowed ip address' netmask in CIDR notation
- etaipm_3, third allowed ip address' netmask in CIDR notation

4.4.2. /save/ethernet/mode

- ethmod, ethernet mode of the device

4.4.3. /save/ethernet

- etdhen, DHCP enable bit
- etdhfb, DHCP fallback enable bit
- etdhfd, DHCP fallback delay
- etsip4, fallback ipv4 address
- etsnm4, fallback subnet mask
- etsgw4, fallback gateway
- etsdn1, static primary DNS
- etsdn2, static secondary DNS
- etshnm, static hostname

4.4.4. /save/snmp/traps

- sntrds, device status code trap enable
- sntrta, temperature alert trap enable
- sntric, input current alert trap enable
- sntroc, output current alert trap enable
- sntrod, output current drop alert trap enable
- sntrid, input current drop alert trap enable
- sntraf, snmp authentication failure alert trap enable
- sntrnc, network connectivity trap enable
- sntrsc, sensor change alert trap enable
- sntrrc, ring state changed trap enable

4.4.5. /save/snmp

- snmpv1, snmp v1 enable
- snmplp, snmp listen port
- snmptp, snmp trap port
- sntrap, trap enable
- sndst1, trap destination ip 1

- sndst2, trap destination ip 2
- snmpro, snmp behavior configuration
- sncmpb, public community
- sncmpr, private community
- sncmtr, trap community

4.4.6. /save/modbus

- mbtcen, modbus enable bit
- mbtcpo, modbus port
- mbtcro, modbus behavior configuration

4.4.7. /save/http

- hthpen, http enable bit
- hthppo, http port
- hthsen, https enable bit
- hthspo, https port

4.4.8. /save/ipapi

- iaenab, ipapi enable bit
- iarc4k, ipapi arc4 shared key

4.4.9. /save/user

- userid, the userid of the user we try to change. Note that this value goes from 1 to 5 and maps from highest role to lowest role (1 = super, 5 = viewer).
- username, the new username for the user with user identifier userid
- (optional) chpasswd, password change invoke (enabling will invoke password to change)
- (optional) password, new password
- (optional) chprmssns, permissions change enable bit
- (optional) prmssns, new permissions