Wase Potentiostat POE Motherboard API

V1, chay@embedism.com

Network address

The motherboard must be manually assigned a static IP address, netmask and gateway. A typical small network often uses the 192.168.0.x subnet, with an internet connected router at address 192.168.0.1. Usually the router will be configured to grant dynamic IP addresses (DHCP), and these dynamic addresses will be in a specified range (for example 192.168.0.127 to 192.168.0.254). In such a case, give the motherboard an IP address in the static range, for example 192.168.0.6.

The netmask should be set according to the subnet the device is on, and the gateway should be set to the router/gateway of this subnet.

The IP settings persist through power-cycle.

Web Interface

The WASE potentiostat motherboard hosts a simple webserver on port 80. The server is HTTP1.1 compliant. A web interface is presented at the root URI '/', or '/index.html'.

The web interface consists of a single-page GUI which uses AJAX-style techniques to regularly retrieve and display the latest values from the hardware, and POST settings which are changed. The web interface acts as a simple demonstration of the API.

Writable values (global power, and enable and voltage per card) are only read once, on page load. Therefore if these values are set by a different client, the changes will only be reflected when the page is refreshed.

The web interface should work in any modern non-Microsoft browser, having been developed in Chrome.

Notes:

- The 'link' checkbox sets the values of all cards to be the same as card 1.
- In link mode, changes to card 1 are set to all cards
- link mode is implemented client-side - the server doesn’t know about it.

API

The API is trivial and consists of a means of obtaining data from the unit, and a means of making changes to settings. JSON is used, and the parsing is fairly tolerant (e.g. of non-specified values, etc).

Getting data

To get the latest values, simply perform a HTTP GET request to the "/x" endpoint. This can be done in a browser, or from the command-line using a tool such as curl or wget. The resulting JSON is shown in appendix 1.

Notes on GET requests:

- the cards array will always contain 9 objects
- data is updated at around 0.5-1Hz, so there is no advantage in querying more frequently than this
- it is fine to have multiple clients consuming the data (by performing) periodic GET requests
- there is no way to get a subset of values from the unit. All values are always returned in a single JSON document.

CLI example (get all current values from unit):

```
> curl -s 192.168.0.6/x
```

Modifying settings

To change values, a JSON document must be constructed and then POSTed to the "/x" endpoint. A HTTP response code of 200 indicates success. Since all of the values that can be set in this way are returned in the response to the GET request, performing such subsequently allow verification of set values (in the case that a 200 'OK' code was returned after the POST) or clarification as to which values were or were not accepted (in the case of a non-200 'OK' code). Appendix 2 has an example.

Although it is imagined that values will normally be changed by sending the entire JSON document as per appendix 2 (this is how the Web interface works, and the JSON is pretty small), this is cumbersome for changing a single value, eg from the command line. For this reason there are 2 ways to specify card values.

1. "cards" is an array, as per appendix 2
Each entry is an object, but these can be blank if setting values is not required, see CLI example 3. The position of the object in the cards array defines which card we are referring to.

2. "cards" is an object, as per CLI example 4
Child objects in the "cards" object are the same as in the array method, but have a string 'key' which is the card index (0-based). Multiple cards can be specified.

Notes on POST request:

- data is posted 'raw' to the server - it is not form-encoded or escaped.
- settings are not persisted, so after a power cycle will need to be re-sent.
- it is fine to adjust the settings as frequently as required, but there is no advantage in doing so more frequently than the GET request is made.
- the server always expects

CLI example 1 (enable power to cards):

```
> curl -d '{"power":true}' 192.168.0.6/x
```

CLI example 2 (enable card 1 output):

```
> curl -d '{"cards":[{"enable":true}]}' 192.168.0.6/x
```

CLI example 3 (set card 9 voltage to 1.4, ugly way):

```
> curl -d'{"cards":[{},{},{},{},{},{},{},{},{"voltage":1.4}]}'
192.168.100.6/x
```

CLI example 4 (set card 9 voltage to 1.4, pretty way):

```
> curl -d '{"cards":{"8":{"voltage":1.4}}}' 192.168.100.6/x
```

Logging script

An example logging script is provided in appendix 3. This bash script uses some commonly available open source tools which may need to be installed if they are not already on your distribution of linux:

- curl is used to get the JSON containing all the values from the unit
- jq is used to parse and process the JSON

To use the script, save it somewhere and chmod for execution. Edit the config values to set the IP address of the unit and the log interval.

When executed, the script will output to stdout a line of CSV every log interval. The csv columns are:

0. timestamp - the timestamp of log entry
1. error - "OK" if we got json from the unit, "ERROR" if we did not (unit is off or disconnected).

The following 4 columns are for each card in turn, so repeated 9 times:

2. card N present - "true" or "false " depending if card physically inserted
3. card N enable - "true" or "false" as per settings
4. card N voltage - -1.5 to 1.5 volts, as per settings
5. card N current - current value in mA

The provided logging script should be modified depending on usage requirements or at minimum, the output should be redirected to a rotating log file.

Appendicies

appendix 1: example JSON response (GET request to /x)


```
{
	"power": true,
	"cards": [{
		"present": true,
		"version": "2",
		"enable": true,
		"voltage": 0.120000,
		"current": -3.630871,
		"status": "running"
	}, {
		"present": true,
		"version": "2",
		"enable": true,
		"voltage": 0.120000,
		"current": 88.661369,
		"status": "running"
	}, {
		"present": true,
		"version": "2",
		"enable": true,
		"voltage": 0.120000,
		"current": 89.580124,
		"status": "running"
	}, {
		"present": true,
		"version": "2",
		"enable": true,
		"voltage": 0.120000,
		"current": 90.341698,
		"status": "running"
	}, {
		"present": true,
		"version": "2",
		"enable": true,
		"voltage": 0.120000,
		"current": 92.965347,
		"status": "running"
	}, {
		"present": true,
```

```
            "version": "2",
            "enable": true,
            "voltage": 0.120000,
            "current": 89.616028,
            "status": "running"
      }, {
            "present": true,
            "version": "2",
            "enable": true,
            "voltage": 0.120000,
            "current": 89.982208,
            "status": "running"
      }, {
            "present": true,
            "version": "2",
            "enable": true,
            "voltage": 0.120000,
            "current": 90.341698,
            "status": "running"
      }, {
            "present": true,
            "version": "2",
            "enable": true,
            "voltage": 0.120000,
            "current": 83.976120,
            "status": "running"
      }]
}
```

appendix 2: example JSON request (POST to /x)

```
{
	"power": true,
	"cards": [{
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}, {
		"enable": true,
		"voltage": 0.12
	}]
}
```

appendix 3: example logging script

```
#!/bin/bash

# configure address and interval
addr="192.168.100.6"
interval=5

# this function converts json for a single card into csv
function card2csv {
csv=`echo $1 | jq '.present, .enable, .voltage, .current'`
csv=`echo $csv | tr -s ' ' ','`
}

# this function gets json from server and outputs csv log entry
function logline {
# get the json
json=`curl -s $addr/x -m5`

# no response? log and bail
if [ -z "$json" ]
then
entry=`date`
entry="$entry,ERROR"
echo $entry
return
fi

# extract root values and card count
power=`echo $json | jq .power`
cardCount=`echo $json | jq '.cards | length'`

# build the log entry
entry=`date`
entry="$entry,OK,$power,"

# loop over cards, appending csv for each one
for (( i=0; i<$cardCount; i++ ))
do
cardjson=`echo $json | jq .cards[$i]`
card2csv "$cardjson"
entry="$entry$csv"
```

```
if  (( i  < ($cardCount-1) )); then
entry="$entry,"
fi
done

echo $entry
return
}

while sleep $interval; do
logline
done
```