

How to Read Data From Shelly Smart Meter

- [1](#)

1

```
sudo apt install python3-venv
```

```
sudo mkdir -p /opt/shellyem_exporter
sudo python3 -m venv /opt/shellyem_exporter/venv
sudo /opt/shellyem_exporter/venv/bin/pip install requests prometheus_client
```

```
sudo vi /opt/shellyem_exporter/shellyem_exporter.py
```

```
#!/usr/bin/env python3
```

```
"""
```

Shelly EM -> Prometheus exporter (multi-target, on-demand)

Works like blackbox_exporter / snmp_exporter: the exporter itself takes no fixed list of devices. Prometheus tells it which device to scrape via query parameters on each scrape request, and the exporter fetches that device's live RPC data synchronously and returns it as Prometheus metrics.

Requirements:

```
pip install requests prometheus_client
```

Usage:

```
python3 shelly_exporter.py --web.listen-address=:9091 --web.telemetry-path=/metrics
```

Scrape request format:

```
GET /metrics?target=<shelly-
ip>[:port]&id=<em_id>&username=<user>&password=<pass>
```

target required - IP/hostname (optionally "host:port") of the Shelly device

id optional - EM component id (default: 0)

username optional - RPC auth username, if the device has authentication enabled

password optional - RPC auth password, if the device has authentication enabled

Example Prometheus scrape config:

```
scrape_configs:
```

```
- job_name: shellyem
```

```
  metrics_path: /metrics
```

```
  static_configs:
```

```

        - targets:
            - 192.168.1.50
            - 192.168.1.51
    params:
        id: ["0"]
    relabel_configs:
        - source_labels: [__address__]
          target_label: __param_target
        - source_labels: [__param_target]
          target_label: instance
        - target_label: __address__
          replacement: 127.0.0.1:9091 # exporter's own address
"""

import argparse
import logging
import threading
import time
from socketserver import ThreadingMixIn
from urllib.parse import parse_qs
from wsgiref.simple_server import WSGIServer, WSGIRequestHandler, make_server

import requests
from prometheus_client import CollectorRegistry, Gauge, generate_latest,
CONTENT_TYPE_LATEST

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s
%(message)s")
log = logging.getLogger("shelly_exporter")

PHASES = ("a", "b", "c")

class _ThreadingWSGIServer(ThreadingMixIn, WSGIServer):
    daemon_threads = True

class _QuietWSGIRequestHandler(WSGIRequestHandler):
    def log_message(self, fmt, *args):
        pass # suppress default access log; we log per-scrape ourselves

def fetch_em_status(shelly_ip: str, em_id: int, timeout: float, auth=None) ->
dict:
    url = f"http://{shelly_ip}/rpc/EM.GetStatus?id={em_id}"
    resp = requests.get(url, timeout=timeout, auth=auth)
    resp.raise_for_status()
    return resp.json()

```

```

def build_metrics(shelly_ip: str, em_id: int, timeout: float, auth=None) ->
tuple:
    """Fetch one device's data and return (registry, success: bool)."""
    registry = CollectorRegistry()

    labels_phase = ["shelly_ip", "em_id", "phase"]
    labels_node = ["shelly_ip", "em_id"]

    current = Gauge("shelly_em_current_amps", "Current in amps", labels_phase,
registry=registry)
    voltage = Gauge("shelly_em_voltage_volts", "Voltage in volts",
labels_phase, registry=registry)
    act_power = Gauge("shelly_em_active_power_watts", "Active power in watts",
labels_phase, registry=registry)
    aprt_power = Gauge("shelly_em_apparent_power_va", "Apparent power in VA",
labels_phase, registry=registry)
    power_factor = Gauge("shelly_em_power_factor", "Power factor",
labels_phase, registry=registry)
    frequency = Gauge("shelly_em_frequency_hz", "Line frequency in Hz",
labels_phase, registry=registry)

    neutral_current = Gauge("shelly_em_neutral_current_amps", "Neutral current
in amps (0 if not measured)",
labels_node, registry=registry)
    total_current = Gauge("shelly_em_total_current_amps", "Total current in
amps", labels_node, registry=registry)
    total_act_power = Gauge("shelly_em_total_active_power_watts", "Total active
power in watts",
labels_node, registry=registry)
    total_aprt_power = Gauge("shelly_em_total_apparent_power_va", "Total
apparent power in VA",
labels_node, registry=registry)

    scrape_success = Gauge("shelly_scrape_success", "1 if the last scrape of
the device succeeded",
labels_node, registry=registry)
    scrape_duration = Gauge("shelly_scrape_duration_seconds", "Time taken to
fetch data from the device",
labels_node, registry=registry)

    start = time.time()
    success = True
    try:
        data = fetch_em_status(shelly_ip, em_id, timeout, auth)

        for phase in PHASES:
            current.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_current", 0) or 0)

```

```

        voltage.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_voltage", 0) or 0)
        act_power.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_act_power", 0) or 0)
        aprt_power.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_aprt_power", 0) or 0)
        power_factor.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_pf", 0) or 0)
        frequency.labels(shelly_ip, em_id,
phase).set(data.get(f"{phase}_freq", 0) or 0)

        # n_current can be null if the device doesn't measure/report it
        neutral_current.labels(shelly_ip, em_id).set(data.get("n_current") or
0)
        total_current.labels(shelly_ip, em_id).set(data.get("total_current", 0)
or 0)
        total_act_power.labels(shelly_ip,
em_id).set(data.get("total_act_power", 0) or 0)
        total_aprt_power.labels(shelly_ip,
em_id).set(data.get("total_aprt_power", 0) or 0)

        log.info("scrape ok [%s id=%s]: total_act_power=%.2fW", shelly_ip,
em_id,
                data.get("total_act_power", 0) or 0)
    except Exception as exc:
        success = False
        log.warning("scrape failed [%s id=%s]: %s", shelly_ip, em_id, exc)

    scrape_success.labels(shelly_ip, em_id).set(1 if success else 0)
    scrape_duration.labels(shelly_ip, em_id).set(time.time() - start)

    return registry, success

def make_app(telemetry_path: str, timeout: float):
    def app(envIRON, start_response):
        path = environ.get("PATH_INFO", "")

        if path == telemetry_path:
            params = parse_qs(environ.get("QUERY_STRING", ""))
            target = (params.get("target") or [None])[0]
            em_id_raw = (params.get("id") or ["0"])[0]
            username = (params.get("username") or [None])[0]
            password = (params.get("password") or [None])[0]

            if not target:
                body = b"Missing required 'target' query parameter, e.g.
/metrics?target=192.168.1.50&id=0\n"
                start_response("400 Bad Request", [("Content-Type",

```

```

"text/plain"), ("Content-Length", str(len(body)))]])
    return [body]

    try:
        em_id = int(em_id_raw)
    except ValueError:
        body = f"Invalid 'id' query parameter:
{em_id_raw!r}\n".encode()
        start_response("400 Bad Request", [("Content-Type",
"text/plain"), ("Content-Length", str(len(body)))]])
        return [body]

    auth = requests.auth.HTTPDigestAuth(username, password) if username
and password else None

    registry, _ = build_metrics(target, em_id, timeout, auth)
    output = generate_latest(registry)
    start_response("200 OK", [("Content-Type", CONTENT_TYPE_LATEST),
("Content-Length", str(len(output)))]])
    return [output]

    if path == "/":
        body = (
            "<html><body><h1>Shelly EM Exporter</h1>"
            f'<p>Usage: <code>{telemetry_path}?target=&lt;shelly-
ip&gt;&amp;id=&lt;em_id&gt;</code></p>'
            "</body></html>"
        ).encode()
        start_response("200 OK", [("Content-Type", "text/html"), ("Content-
Length", str(len(body)))]])
        return [body]

    start_response("404 Not Found", [("Content-Type", "text/plain")])
    return [b"404 Not Found"]

return app

def parse_listen_address(value: str) -> tuple:
    """Parse a Prometheus-exporter-style listen address like ':9091' or
'127.0.0.1:9091' into (host, port). An empty host means all interfaces."""
    if ":" not in value:
        raise ValueError(f"Invalid --web.listen-address {value!r}, expected
HOST:PORT or :PORT")
    host, _, port_str = value.rpartition(":")
    try:
        port = int(port_str)
    except ValueError:
        raise ValueError(f"Invalid port in --web.listen-address {value!r}")

```

```

    return host or "0.0.0.0", port

def start_server(listen_host: str, listen_port: int, telemetry_path: str,
timeout: float):
    app = make_app(telemetry_path, timeout)
    server = make_server(listen_host, listen_port, app,
                        server_class=_ThreadingWSGIServer,
                        handler_class=_QuietWSGIRequestHandler)
    t = threading.Thread(target=server.serve_forever, daemon=True)
    t.start()
    return server

def main() -> None:
    parser = argparse.ArgumentParser(description="Shelly EM Prometheus exporter
(multi-target, on-demand)")
    parser.add_argument("--web.listen-address", dest="listen_address",
default=":9091",
                        help="Address to expose metrics on, HOST:PORT or :PORT
(default: :9091)")
    parser.add_argument("--web.telemetry-path", dest="telemetry_path",
default="/metrics",
                        help="Path under which to expose metrics (default:
/metrics)")
    parser.add_argument("--scrape-timeout", type=float, default=5.0,
                        help="HTTP request timeout when fetching a target's
data (default: 5)")
    args = parser.parse_args()

    listen_host, listen_port = parse_listen_address(args.listen_address)
    telemetry_path = args.telemetry_path
    if not telemetry_path.startswith("/"):
        telemetry_path = "/" + telemetry_path

    log.info("starting exporter -> %s:%s%s?target=<ip>&id=<em_id>",
listen_host, listen_port, telemetry_path)

    server = start_server(listen_host, listen_port, telemetry_path,
args.scrape_timeout)
    try:
        while True:
            time.sleep(3600)
    except KeyboardInterrupt:
        server.shutdown()

if __name__ == "__main__":
    main()

```

```
sudo groupadd --system shellyem_exporter
sudo useradd --system --no-create-home --shell /usr/sbin/nologin --gid
shellyem_exporter shellyem_exporter
```

```
sudo vi /etc/systemd/system/shellyem_exporter.service
```

```
[Unit]
Description=Shelly EM Prometheus Exporter
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=shellyem_exporter
Group=shellyem_exporter
EnvironmentFile=-/etc/default/shellyem_exporter
ExecStart=/opt/shellyem_exporter/venv/bin/python3
/opt/shellyem_exporter/shellyem_exporter.py $SHELLYEM_OPTS
WorkingDirectory=/opt/shellyem_exporter
Restart=on-failure
RestartSec=5

# Hardening
NoNewPrivileges=true
ProtectSystem=strict
ProtectHome=true
PrivateTmp=true
ReadWritePaths=-/opt/shellyem_exporter
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
RestrictSUIDSGID=true
LockPersonality=true

[Install]
WantedBy=multi-user.target
```

```
sudo vi /etc/default/shellyem_exporter
```

```
# Options passed to shelly_exporter.py, e.g.:
SHELLYEM_OPTS="--config.file=/opt/shellyem_exporter/shellyem_exporter.yml --
web.listen-address=:9091 --web.telemetry-path=/metrics"
```

```
sudo systemctl daemon-reload
sudo systemctl enable --now shellyem_exporter.service
```