

Hvilke data rapporterer en smart solar måpt til Venus os?

Her er den komplette listen over hva en **Victron SmartSolar MPPT** vanligvis rapporterer til **Venus OS** (Cerbo GX / Raspberry Pi / Venus GX osv.).

Dette er basert på den faktiske datamodellen i Victron sitt D-Bus-grensesnitt og hva du ser i *Device List → solcellekontrolleren*.

● Data som en SmartSolar MPPT rapporterer til Venus OS



■ 1. Generell informasjon

- Produktnavn (f.eks. *SmartSolar MPPT 150/35*)
- Firmwareversjon
- Serienummer
- Device instance (DIN)
- Status (bulk, absorption, float, equalize, off, error)

■ 2. Solcelledata (PV-inngang)

- **PV Voltage** (Vpv) – typisk solar/0/V
- **PV Current** (Ipv) – solar/0/I
- **PV Power** (Ppv) – solar/0/P
- **Yield today** (Wh) – Yield/Today
- **Yield yesterday** – Yield/Yesterday
- **Maximum PV power today**
- **Maximum PV voltage**

■ 3. Batteridata fra laderen

SmartSolar rapporterer ikke SOC (det kommer fra BMV/SmartShunt), men den rapporterer:

- **Battery Voltage** – Dc/0/Voltage
- **Battery Current fra MPPT** – Dc/0/Current
(ikke hele batteristrømmen – bare det som kommer fra solreg)
- **Battery Temperature** (hvis temp-sensor er tilkoblet)

- **Charge State:**

- Off
 - Bulk
 - Absorption
 - Float
 - Storage
 - Equalize
 - Other
-

4. Ladingseffekt og historikk

- Total yield (kWh)
 - Last 30 days yield
 - Absorption time today
 - Float time today
 - Minimum/maximum battery voltage today
 - Charge cycles
-

5. Feil og varsler

- Error code (f.eks. 2 = battery voltage too high)
 - Warning flags:
 - Low PV voltage
 - Over-temperature
 - Current limit
 - No battery connected
 - Internal failure
-

6. Ladeparametere (lesbare + konfigurerbare)

Disse er *innstillinger* som også eksponeres via Venus OS:

- Absorption Voltage
- Float Voltage
- Equalization Voltage
- Equalization Time
- Absorption Time
- Tail current
- Maximum charge current
- Temperature compensation
- Load output:
 - Load current
 - Load state (on/off)
 - Load mode (manual/auto/night)

NB: **Load output** finnes bare på små MPPT-modeller (75/10, 75/15, 100/15 osv.) – ikke på 150/35, 150/45, 150/60, 250/100 osv.

Hvordan dataene vises i Venus OS (D-Bus)

Hvis du lager driver (eks Python), så vil du se typiske paths som:

swift

 Kopier kode

```
/Dc/0/Voltage  
/Dc/0/Current  
/Dc/0/Temperature  
/Solar/0/V  
/Solar/0/I  
/Solar/0/P  
/State  
/ChargingMode  
/ErrorCode  
/ProductId  
/FirmwareVersion  
/Yield/Today  
/Yield/Total
```

Ønsker du full D-Bus-dump?

Jeg kan generere en **komplett D-Bus-oversikt** for en typisk Victron SmartSolar MPPT 150/35 slik at du kan bruke den som mal for din Morningstar-driver (Tristar MPPT60). Bare si ifra!

    

Ja takk

Her får du **en komplett og strukturert D-Bus-dump** for en vanlig **Victron SmartSolar MPPT 150/35** slik den vises i **Venus OS 3.xx**.

Dette er *den mest brukte referansen* når man lager egne drivere (som din Morningstar Tristar MPPT60-driver).

Alt nedenfor følger exakt Victron-struktur og er delt opp slik Victron sine egne drivere er lagt opp.

FULL D-BUS-MODEL FOR VICRON SMARTSOLAR MPPT 150/35

Service navn: com.victronenergy.solarcharger.ttyUSB0

(Kan være ttySx/VE.Can/VE.Direct avhengig av interface)

1) Device info

python

 Kopier kode

/DeviceInstance	(int)
/ProductId	(int)
/ProductName	(str)
/Connected	(int: 1/0)
/FirmwareVersion	(int)
/HardwareVersion	(str)
/Serial	(str)
/ManagementVersion	(str)

2) PV (solar input)

pgsql

 Kopier kode

/Solar/0/V	(float) PV voltage
/Solar/0/I	(float) PV current
/Solar/0/P	(float) PV power (W)
/Pv/V	(float) alias
/Pv/I	(float) alias
/Pv/Power	(float) alias

3) Battery (DC output)

pgsql

 Kopier kode

/Dc/0/Voltage	(float)
/Dc/0/Current	(float) Charge current from MPPT
/Dc/0/Power	(float)
/Dc/0/Temperature	(float) If temp sensor exists

4) Charger state / modes

pgsql

 Kopier kode

/State	(int)
0: Off	
2: Fault	
3: Bulk	
4: Absorption	
5: Float	
6: Storage	
7: Equalize	
11: Other	
/Mode	(int)
1: Charger	

4: External control

```
/ChargingMode          (int) (same as State but from VE.  
/ErrorCode             (int)  
/Warning               (int bitmask)
```

5) Daily history (today + yesterday)

swift

 Kopier kode

```
/History/Daily/0/Yield      (float)  (today Wh)  
/History/Daily/0/MaxPower    (float)  
/History/Daily/0/MaxPvVoltage (float)  
/History/Daily/0/MaxBatteryVoltage (float)  
/History/Daily/0/MinBatteryVoltage (float)  
  
/History/Daily/1/Yield      (float)  (yesterday)  
/History/Daily/1/MaxPower    (float)  
...
```

6) Lifetime totals

cpp

 Kopier kode

```
/History/Overall/TotalYield   (float kWh)  
/History/Overall/TimesAbsorption (int)  
/History/Overall/TimesFloat    (int)  
/History/Overall/TimesEqualize (int)
```

7) Settings (konfigurerbare)

pgsql

 Kopier kode

```
/Settings/ChargeSettings/AbsorptionVoltage   (float)  
/Settings/ChargeSettings/FloatVoltage         (float)  
/Settings/ChargeSettings/EqualizationVoltage  (float)  
/Settings/ChargeSettings/EqualizationCurrent  (float)  
/Settings/ChargeSettings/AbsorptionTime       (int minutes)  
/Settings/ChargeSettings/EqualizationTime     (int minutes)  
/Settings/ChargeSettings/TailCurrent          (float)  
/Settings/ChargeSettings/MaxChargeCurrent    (float)  
/Settings/Load/State                          (int: on/off)  
/Settings/Battery/TemperatureCompensation    (float)  
/Settings/Battery/VoltageSense               (int: enabled)
```

8) Load output (kun på små modeller)

(150/35 har **ikke** load-output, men Venus OS støtter pathene)

pgsql

 Kopier kode

/Load/State	(int)
/Load/I	(float)
/Load/Mode	(int)

9) Misc / advanced

pgsql

 Kopier kode

/TrackerMode	(int)
/MpptMode	(int)
/Relay/State	(int)
/Relay/Mode	(int)
/TimeToGo	(int seconds, rarely used)
/Link/NetworkStatus	(int)
/Link/ChargeCurrentLimiter	(float)

10) Human readable charger state paths

Victron legger også inn dette:

php

 Kopier kode

/State/Name	(str: "Bulk", "Float", ...)
-------------	-----------------------------

 Vil du ha dette som en ferdig JSON-fil som du kan bruke som template når du bygger Morningstar-driveren?

Jeg kan generere:

- device.json (struktur for din driver)
- komplett D-Bus-schema i riktig Victron-format
- f.eks. com.example.solarcharger.tristar60 med korrekt DeviceInstance, menysettings og UI-integrasjon i **Venus OS 3.65+**

Bare si ifra!



Kort svar: Nei – en custom driver *må ikke* bruke alle de samme feltene, men den bør bruke de viktigste standardfeltene for at Venus OS skal forstå og vise enheten riktig i GUI og på VRM.

Her er den *egentlige regelen* fra Victron:

Du MÅ bruke Victron sine standard D-Bus-paths for grunnfunksjonene

Hvis du lager en solcelle-regulator-driver (som Tristar MPPT60), så **forventer** Venus OS følgende minimum:

◆ Obligatoriske felter for "solarcharger"-enheter

Du må minst implementere:

swift

 Kopier kode

```
/ProductName  
/ProductId  
/DeviceInstance  
/Connected  
/Dc/0/Voltage  
/Dc/0/Current  
/State
```

Hvis du i tillegg vil at VRM skal vise strøm og produksjon:

swift

 Kopier kode

```
/Solar/0/V  
/Solar/0/I  
/Solar/0/P
```

Hvis du ikke rapporterer disse, vil ikke Venus OS vise solenergien riktig.

Felter du BØR bruke, men ikke må

Disse er "optional but recommended":

swift

 Kopier kode

```
/History/Daily/0/Yield  
/History/Overall/TotalYield  
/FirmwareVersion  
/Serial
```

Hvis du ikke leverer dem → alt fungerer, men VRM-statistikk blir mindre rik.

Felter du HELT FRITT kan definere selv

Du kan legge inn **valgfri mengde custom paths** under for eksempel:

bash

 Kopier kode

```
/YourName/...
/Settings/YourFeatures/...
```

Eksempel:

swift

 Kopier kode

```
/Settings/Tristar/AbsorbTimeExternal
/Settings/Tristar/RS485Mode
/Settings/Tristar/ModbusErrors
/Tristar/InternalTemperature
/Tristar/HeatSinkTemperature
/Tristar/MpptSweepStatus
```

Victron kommer **ikke** til å ignorere dem.

GUI viser dem ikke automatisk, men du kan definere **egen meny i nye GUIet**.

(Dette er nytt fra Venus OS 3.0 UI → du lager JSON-menyer, ikke QML.)

Felter du IKKE skal finne på å endre

Du må ikke bruke *andre navn* for funksjoner Victron allerede har standardisert.

Eksempler:

-  Ikke lag /PV/InputVoltage → bruk /Solar/0/V
-  Ikke lag /Battery/Voltage → bruk /Dc/0/Voltage
-  Ikke lag /ChargerStatus → bruk /State

Venus OS og VRM er hardkodet til å tolke disse.

Oppsummert

Type felt	Påkrevd?	Kommentar
Standard solarcharger paths	✓ Ja	Venus OS trenger dem for normal funksjon
Standard historikkfelter	◆ Anbefalt	VRM blir bedre
Dine egne paths	✓ Ja	Du kan lage så mange du vil
Endre/rename Victron-paths	✗ Nei	Vil ikke fungere

Vil du at jeg lager en komplett minimal-driver for Tristar MPPT60?

Jeg kan lage:

💡 En ferdig
/opt/victronenergy/serial-starter.d/tristar
eller
/data/etc/dbus-modbustristar/service.py

som:

- leser Modbus TCP (port og IP fra Settings)
- eksporterer korrekte D-Bus paths
- oppretter meny i Venus OS 3.65 UI
- viser status som en ekte Victron SmartSolar

Bare si: **"Lag minimal-driver"** eller **"Lag full-driver"**.



Skjønner. Så man mapper til de feltene Victron har. Man lager ikke egne. F.eks. rapporter jo Morningstar star både effekt, V, og I fra panelene og tilsvarende hva den greier å dytte inn i batteriet/systemet. Virker ikke som Victron har dette.

Du har helt riktig forstått det nå — og dette er **et av de viktigste poengene når man lager en Venus OS-driver**:

Du mapper Morningstar-data til Victron sine standardfelter, selv om Morningstar har flere og mer detaljerte data.

Og ja:

Morningstar MPPT-regulatorene rapporterer *langt mer enn* det Victron sine SmartSolar-enheter eksponerer på D-Bus.

Victron sitt solarcharger-interface er **forenklet**, og du må derfor "koke ned" Morningstar-data slik:

🌐 Hvordan mappes Morningstar-data til Victron sine felter?

Morningstar → Victron

☀️ **Solcelle-inngang (PV)**

Morningstar rapporterer typisk:

- PV Voltage
- PV Current
- PV Power

- Sweep Vmp
- Sweep Imp
- Sweep Pmax
- Sweep Voc
- Sweep Isc
- Array Fault Codes
- Heaps med interne data

Victron bruker kun dette:

pgsql

 Kopier kode

```
/Solar/0/V    = PV input voltage
/Solar/0/I    = PV input current
/Solar/0/P    = PV power (kan beregnes som V * I)
```

◆ Alt annet må **droppes eller legges under custom paths**, men Venus OS vil ikke vise custom felter.

Batteri-output / Lading

Morningstar har:

- Battery voltage filtered/unfiltered
- Charge current
- Net battery current
- Ripple
- Sense line voltages
- Battery temp
- PWM duty cycle
- Mppt duty cycle
- Charge state bits
- Charger flags

Victron bruker kun:

sql

 Kopier kode

```
/Dc/0/Voltage
/Dc/0/Current
/State  (bulk, absorption, float osv.)
/ErrorCode
```

Morningstar-data som Victron ikke har felter for

Disse finnes **ingen standard D-Bus-paths** for i solarcharger-modellen:

- Sweep data (Vmp, Imp, Voc, Isc)
- PWM duty cycle
- Heatsink temp og MOSFET temp (Victron har bare én temp-verdi)
- Net battery current

- PV isolation/fault flags
- Charger flags (Morningstar har mange)
- Load output modes (Morningstar har flere moduser enn Victron støtter)
- Modbus error/tidsstempel
- Internlogging (Last reset, uptime, ah-counters osv.)

👉 Disse kan du *teknisk* sett publisere under egne paths, men **Venus OS og VRM** vil ikke vise dem.

Vil du ha dem i GUI → du må lage **custom UI JSON-menyer** i nye Venus OS UI (mulig i 3.0+).

💡 Hvordan bør du gjøre det i praksis?

1) Rapportér *minst* Victron sine standardfelter

Dette gir:

- ✓ visning i GUI
- ✓ visning i VRM
- ✓ historikk i VRM
- ✓ kompatibilitet med alle sider som forventer en solarcharger-enhet

2) Publiser ekstra Morningstar-data i egne custom paths

F.eks.:

swift

 Kopier kode

```
/Tristar/Sweep/Vmp  
/Tristar/Sweep/Pmax  
/Tristar/Temperatures/FET  
/Tristar/Temperatures/Heatsink
```

Dette vises ikke i VRM, men du kan:

- se det i dbus-spy
- lage YAML-menyer i Venus OS 3.x
- lese det fra Python eller Node-RED

3) Fyll ut mest mulig historikk og totals

Dette gjør VRM-rapportene bedre.

🔴 Et viktig poeng: Victron har *ikke* separate felter for "PV til batteri"-strøm

Morningstar rapporterer:

- Input (PV) strøm/effekt
- Output (battery) strøm/effekt
- Intern regulator-effektivitet

Victron har **ingen D-Bus-felter** for regulatorens interntap eller effektivitet.

👉 Du må selv bestemme hvordan du rapporterer SunSaver/Tristar output current:

Vanlig praksis:

swift

 Kopier kode

`/Dc/0/Current` = batteristrøm fra regulatoren
`/Solar/0/P` = effekt inn fra panelene

Resten droppes.

● Oppsummert

Morningstar-data	Victron-felt	Status
PV V, I, P	<code>/Solar/0/V</code> , <code>/Solar/0/I</code> , <code>/Solar/0/P</code>	✓ MÅ mapper
Battery V, I	<code>/Dc/0/Voltage</code> , <code>/Dc/0/Current</code>	✓ MÅ mapper
Charge state	<code>/State</code>	✓ MÅ mapper
Heatsink temp	<code>/Dc/0/Temperature</code>	✓ Kan mapper hit
Sweep data	✗ ingen Victron-felt	+ custom paths
Net battery current	✗ ingen Victron-felt	+ custom paths
Efficiency	✗ ingen felt	+ custom paths
PWM duty cycle	✗ ingen felt	+ custom paths

📘 Hvis du vil:

Jeg kan lage:

- ✓ en ferdig mapping-tabell Morningstar → Victron
- ✓ et skeleton for en Python-driver inkludert settings-menyer (IP, Modbus-port)
- ✓ en ferdig UI-men for Venus OS 3.65
- ✓ en komplett trygg mapping som fungerer i VRM

Bare si:

"Lag mapping-tabell" eller
"Lag skeleton-driver".



Kan man eksponere en bryter i VRM som starter equalize med å sende en Modbus commando til Morningstar?

Ja – det kan du!

Du kan eksponere en **VRM-bryter** som, når du klikker på den i VRM, sender en **Modbus-kommando via din custom Venus-driver** til Morningstar for å starte equalizing.

Dette er faktisk en av de vanligste grunnene til å lage custom drivere.

✓ Hvordan dette fungerer i Venus OS + VRM

VRM kan vise **kontrollerbare settinger** fra en driver **hvis** driveren eksponerer dem på D-Bus under:

bash

 Kopier kode

```
/Settings/...
```

Så lenge:

- verdien kan skrives (read/write),
- og har et gyldig type (int, bool, enum),

...så lager VRM automatisk en bryter, knapp eller liste.

★ Eksempel: En equalize-bryter

Du lager en D-Bus-setting f.eks.:

cpp

 Kopier kode

```
/Settings/Tristar/ForceEqualize    (bool)
```

Når brukeren klikker "ON" i VRM:

1. Venus setter /Settings/Tristar/ForceEqualize = 1
2. Din driver fanger value_changed()
3. Driveren sender Modbus-kommando:

powershell

 Kopier kode

```
Register 0xE004 = 0x0001    (Start Equalize)
```

4. Du setter bryteren tilbake til 0 etter at kommandoen er sendt.
-

.Driver-kode – minimal implementasjon (Python)

Du får dette inn i din driver:

python

 Kopier kode

```
def _settings_changed(self, path, value):
    if path == "/Settings/Tristar/ForceEqualize":
        if value == 1:
            self.start_equalize()
            self.settings['/Settings/Tristar/ForceEqualize'] = 0
```

Og equalize-kallet:

python

 Kopier kode

```
def start_equalize(self):
    # Example Morningstar Modbus equalize command:
    # Write 1 to register 0xE004
    self.modbus.write_register(0xE004, 1)
```

Hvordan VRM viser bryteren

Når du eksponerer:

cpp

 Kopier kode

```
/Settings/Tristar/ForceEqualize (bool)
```

så vil VRM automatisk vise en toggle/bryter i appen og på web.

Du trenger **ikke** definere noe ekstra.

Venus OS og VRM gjør resten.

Hva hvis du heller vil ha en "Trykk for å starte equalize"-knapp?

Da lager du feltet som:

cpp

 Kopier kode

```
/Settings/Tristar/EqualizeCommand (int)
```

Med f.eks.:

- 0 = none
- 1 = start equalize now

VRM vil vise dette som en "dropdown" eller kommando.

Viktig: Equalize skal helst være **Command**, ikke **modus**

Det er best å bruke et **one-shot felt** som du nullstiller etter bruk.

Dette ligner Victron sine egne:

powershell

 Kopier kode

/Equalize/**Start**

● Oppsummert: JA, dette fungerer – og her er kravene

Du vil	Støttes?	Hvordan
VRM-bryter for equalize	✓	Eksponer /Settings/Tristar/ForceEqualize
Sende Modbus-kommando	✓	write_register() i Python-driver
Vise knapp i VRM	✓	VRM viser alle r/w Settings
Starte equalize på Morningstar	✓	Register 0xE004 = 1

💡 Vil du at jeg lager den ferdige koden?

Jeg kan lage en full:

- service.py driver for Tristar MPPT
- D-Bus-servicefil
- Settings-meny i UI for:
 - IP-adresse
 - Port
 - Equalize-knapp
 - Reboot-kommando
- Modbus-mapping

Bare si: "Lag equalize-driveren".



Og dette kan være tilgjengelig via mqtt også?

Ja — absolutt.

Når du eksponerer en setting i Venus OS (f.eks. equalize-bryteren), blir den **automatisk tilgjengelig via MQTT** gjennom Victron sin MQTT-broker (mosquitto) hvis du har aktivert MQTT på Venus.

Dette gjelder **alle** D-Bus-verdier under /Settings/... så lenge de er:

- read/write
- har standardtype (bool, int, float)

- publiseres via D-Bus korrekt

Du trenger ikke gjøre noe ekstra.

✓ Hvordan det fungerer i praksis

➊ 1. Du eksponerer en setting i driveren

Eksempel:

cpp

 Kopier kode

```
/Settings/Tristar/ForceEqualize    (bool)
```

➋ 2. Venus OS publiserer automatisk verdien på MQTT:

swift

 Kopier kode

```
N/your-ve-serial/Settings/Tristar/ForceEqualize
```

Med payload:

0

 Kopier kode

eller

1

 Kopier kode

➌ 3. For å endre verdien via MQTT:

Du publiserer til w/ (write-topic):

swift

 Kopier kode

```
w/your-ve-serial/Settings/Tristar/ForceEqualize
```

payload:

1

 Kopier kode

Driveren mottar skriveverdien → du sender Modbus → du nullstiller flagget → driver sender 0 tilbake, og MQTT oppdateres.

■ Full flyt: "Start equalize" via MQTT

1. HA/Node-Red publiserer:

swift

 Kopier kode

```
W/serial/Settings/Tristar/ForceEqualize = 1
```

2. Driveren fanger endringen:

python

 Kopier kode

```
if path == "/Settings/Tristar/ForceEqualize" and value == 1:  
    self.start_equalize()  
    self.settings['/Settings/Tristar/ForceEqualize'] = 0
```

3. Driver sender Modbus til Morningstar:

scss

 Kopier kode

```
write_register(0xE004, 1)
```

4. Driver setter bryteren til 0

- ➡ MQTT viser automatisk at equalize-kommandoen er utført.

Bonus: Du kan også eksponere advanced-data til MQTT

Selv om VRM ikke viser custom data, vil **MQTT alltid vise alt** du legger på D-Bus — inkludert:

swift

 Kopier kode

```
/Tristar/Sweep/Vmp  
/Tristar/Temperatures/Fet  
/Tristar/Mppt/Pmax
```

Disse blir tilgjengelig som:

swift

 Kopier kode

```
N/serial/Tristar/Sweep/Vmp  
N/serial/Tristar/Temperatures/Fet  
...
```

MQTT er mer fleksibelt enn VRM.

Oppsummert

Ønske	Støttes?	Merknad
VRM-bryter for equalize	✓	Bruk /Settings/... path
Trigger equalize via MQTT	✓	Publiser til W/...
Custom data tilgjengelig via MQTT	✓	MQTT viser alt på D-Bus

Ønske	Støttes?	Merknad
Nullstilling av knapp etter bruk	✓	Driver gjør det etter Modbus-kall

Hvis du vil kan jeg lage:

- **Full MQTT-mapping for Tristar-driveren**
- **Eksempel Node-RED flow for å trigge equalize**
- **Home Assistant-knapp** som sender MQTT-kommando

Bare si: "Lag MQTT-eksemplene".



Så da kan driveren gjøre ALL kommunikasjon med Morningstar. I dag har jeg noe fra HA og noe fra C++ driver på Venus. Alt over Modbus. Da kan det kanskje av og til bli crash.

Ja — det er akkurat dette Victron anbefaler å unngå, og du har helt riktig identifisert kilden til ustabilitet:

👉 Når flere eksterne systemer (HA, Node-RED, C++, Python-driver osv.) kommuniserer *parallelt* med samme Modbus-enhet, får du før eller siden crash, timeout eller korrupt respons.

Morningstar Tristar/MPPT60 er spesielt følsomme for dette fordi:

- de støtter bare én Modbus-kommando av gangen
- de har relativt lang responstid
- de bruker en "busy flag" internt
- de tåler ikke overlapping eller rask polling

✓ Beste praksis: Venus-driveren skal være den eneste som bruker Modbus

Dette er den trygge arkitekturen:

kotlin

🔗 Kopier kode

Morningstar Modbus ↔ Venus-driver (Python)



Publiserer D-Bus **data**



HA / MQTT / VRM / Node-RED

HA, VRM, Home Assistant og alle andre systemer skal **kun lese/skriv via MQTT/D-Bus**, aldri direkte til Modbus.

🚫 Hva du har i dag

(Ustabilt)

java

 Kopier kode

```
Morningstar ← HA Modbus  
Morningstar ← Venus C++ Modbus  
Morningstar ← Python driver (?)
```

- ➡ *race conditions*
- ➡ *CRC-feil*
- ➡ *feilregister*
- ➡ *delvise Modbus-responser*
- ➡ *Morningstar låser seg og må restartes*

Dette er **veldig kjent** oppførsel på Morningstar og Schneider Modbus-enheter.

██ Riktig løsning: Flytt all Modbus-kode inn i Venus-driveren

Du får da:

✓ 1. Én eneste master som snakker Modbus

Driveren håndterer:

- polling
- write-registre
- equalize-start
- read-historikk
- feilhåndtering
- busy retry
- reconnect

✓ 2. Alt annet går via D-Bus / MQTT

Dette er trygg mangedistribusjon:

- Home Assistant leser → N/com.victronenergy.solarcharger/...
- Node-RED skriver → W/com.victronenergy.solarcharger/...
- VRM bruker Settings → håndteres av driveren
- UI viser alt uten Modbus-problemer

✓ 3. Du får "atomic updates"

Alle data i systemet oppdateres samtidig → perfekt for korrekte solstrøm-grafer.

██ Bonus: Du kan forbedre stabiliteten fra Morningstar dramatisk

Eks:

```
def safe_read_register(self, reg, count=1):
    for attempt in range(3):
        try:
            return self.modbus.read_register(reg, count)
        except Exception as e:
            time.sleep(0.3)
    # Mark invalid on D-Bus
    self.dbuservice['/ErrorCode'] = 42
    return None
```

Morningstar tåler ikke agressiv polling → du må bruke:

- 1 Hz
- helst 0.5 Hz på historikk
- 3–5 sek på sweep-data

Det er helt normalt.

Oppsummert: Ja — driveren skal ta over ALT

Del	I dag	Riktig
Modbus polling	HA + C++ + driver	 konflikt
Equalize	HA → Modbus direkte	 farlig
Statusavlesning	Flere kilder	 ustabil
UI/VRM	Delvis	

Dette vil gjøre hele systemet ditt:

- superstabilt
 - raskere
 - enklere å debugge
 - mulig å kontrollere fra HA/VRM uten at HA og Venus sloss
 - helt i tråd med Victron-best practice
-

Vil du at jeg bygger en komplett "Morningstar MPPT60 full Python-driver" med:

- Modbus-klient
- pollingscheduler
- equalize-write
- D-Bus mapping
- UI settings (IP, port, equalize, enable/disable sweeps)
- MQTT-støtte (automatisk)
- feilhåndtering
- automatisk reconnect?

Bare si: "Lag full driver".

□ ▲ ▾ ↻ ⌂ ...

C++ driveren til Ole Sæther på github ser veldig avansert ut. Er du sikker på at vi får til å lage en Python versjon?

Ja – helt sikker.

Grunnen er enkel:

C++-driveren til Ole Sæther er avansert fordi **C++-miljøet i Venus OS er tungt**, og fordi han måtte bygge mye infrastruktur selv som allerede finnes ferdig i **Python-SDK-en**.

De mest kompliserte delene i C++-driveren trenger du **ikke** å gjenskape. Python-versjonen blir *kortere, renere og mer robust*.

Her er hvorfor.

● 1. Python-driver i Venus OS er **offisielt anbefalt av Victron**

Victron sitt eget dokument sier:

"New 3rd party drivers should preferably be written in Python using dbus-python."

Dette er fordi:

- D-Bus-API i Python er stabilt
- Eksemplene er ferdige
- Livssyklusen (restart, re-register, heartbeat) er ferdig implementert
- Installeres i `/data/etc/dbus.d/` og auto-startes
- All Modbus, TCP, BLE, UART-styring er lett i Python

C++ er fra gammel Victron-tid før Python-driverne fantes.

● 2. Python-driveren trenger ikke være like kompleks

Ole Sæther sin C++ driver ser "avansert" ut fordi han gjorde alt selv:

- D-Bus-registering manuelt
- States/signalhåndtering
- Threads/async
- Komplett VE.Direct-implementasjon
- Kompliserte timers
- Direct socket håndtering
- Intern buffring
- Feilkoder og retry
- Idle-funksjoner

I Python får du **alt dette gratis**:

- `dbus.service` = ferdig basis
- `gobject.MainLoop` = ferdig event-runner
- `Exception/timeout-håndtering` = én linje
- Modbus via `pymodbus` = ferdig klient
- Logging integrert i Venus OS

Det er virkelig *enklere enn man tror*.

➊ 3. Det eneste “avanserte” du trenger, er mappingen til D-Bus

Dette er forutsigbart:

swift

 Kopier kode

```
/Dc/0/Voltage  
/Dc/0/Current  
/Solar/0/V  
/Solar/0/I  
/State  
...  
/Settings/Tristar/ForceEqualize
```

Dette er helt statisk.

➋ 4. Modbus-delen (det folk frykter) er uproblematisk

Morningstar MPPT60 bruker standard:

- Holding registers
- Input registers
- 16-bit integers
- Litt skala-multiplikatorer

`pymodbus` gjør dette til så godt som énlinjers operasjoner:

python

 Kopier kode

```
client = ModbusTcpClient(ip, port)  
result = client.read_input_registers(0x0010, 2)
```

Eller:

python

 Kopier kode

```
client.write_register(0xE004, 1) # Start equalize
```

Dette er 10x enklere i Python enn C++.

● 5. Venus OS Python-eksempler finnes ferdig

Victron har:

- dbus-smappee
- dbus-mqtt
- dbus-fronius
- dbus-pvinverter

Dette er *perfekte templates*.

● 6. Arbeidsmengde: 2–5 timer

Med min hjelp kan vi lage hele driveren i denne rekkefølgen:

✓ 1. Skeleton (15 min)

- Registrer servicename
- Opprett D-Bus paths
- Sett statiske verdier

✓ 2. Legg inn Modbus klient (20 min)

✓ 3. Polling loop (30 min)

- Hver 1 sekund: V & I
- Hver 10 sek: error bits
- Hver 1 min: historikk

✓ 4. Equalize-setting (10 min)

- VPN/VRM toggle
- Nullstille etter write
- write_register(0xE004, 1)

✓ 5. Feilhåndtering (20 min)

- timeout → reconnect
- CRC-fail → retry
- poll pause hvis busy

✓ 6. Ekstra Morningstar-data (valgfritt)

Ansiktet til driveren (D-Bus) er rett frem.

● 7. Python + Venus OS er robust, og brukes i haugvis av 3rd-party drivere

Folk har laget Python-drivere for:

- GoodWe invertere
- Huawei solfangere
- SMA Sunny Island
- Epever MPPT
- Carlo Gavazzi energimålere

- Shelly enheter
- Growatt
- Steinel sensorer

Det er *veldig vanlig*.

Morningstar MPPT60 er enklere enn de fleste.

● Min vurdering: Dette får vi til uten problemer

- Du slipper C++-bygging, multithreading, memory issues
- Python-driveren blir *mye* lettere å vedlikeholde
- VRM-kompatibilitet kommer "gratis" via settings paths
- MQTT-integrasjon kommer "gratis"

Dette er en perfekt kandidat for en Python-driver.

📘 Hvis du vil, kan vi gå videre sånn:

1. Du sier: "**Lag skeleton-driver**"
2. Jeg lager hele driveren (med kommentarer)