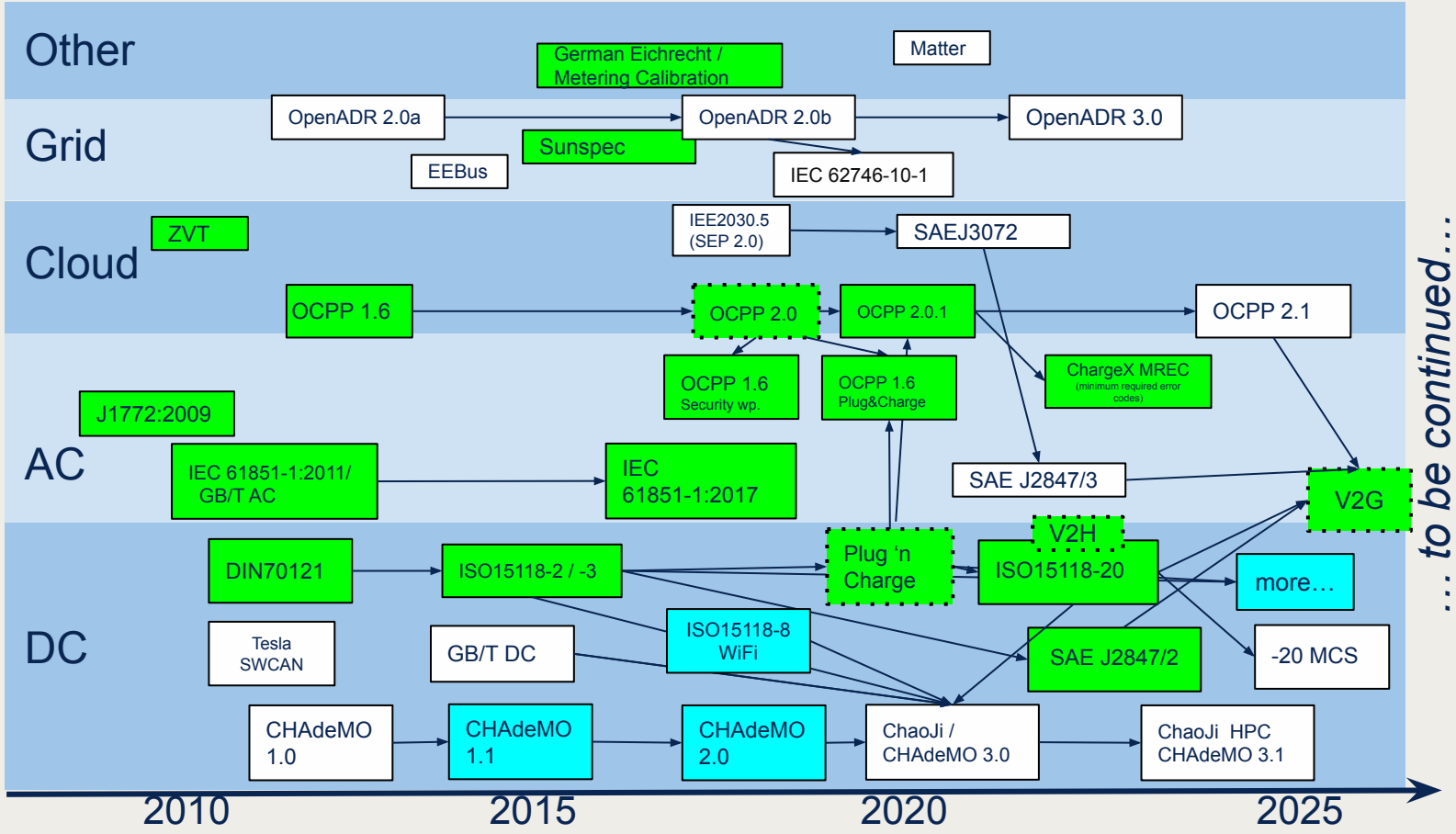

PIONIX
e-mobility powered by open source

PIONIX

What is EVerest?



Tech Overview &
Example Configurations

Beautiful **modular** microservice architecture & middleware

Many **protocols** included:
- OCPP 1.6 (all) / 2.0.1 / 2.1 (basics)
- ISO 15118-2(AC+DC)
  (ISO 15118-20 upcoming)
- DC-BiDi: SAE J2847/2
- DIN SPEC 70121
- IEC 61851 / SAE J1772
- ModBus
- Sunspec
- MQTT
- CCS1, CCS2, NACS/Tesla

Many **language** bindings:
C++ 17, Rust, Python, JavaScript

Seamless buildings blocks for all use cases

Software (& Hardware) in the Loop **Simulations** - develop on single laptop

NodeRed integration for rapid prototyping

Opt. local **energy management**

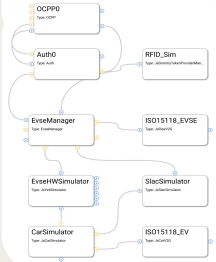
Opt. **user interface** available (Flutter based)

Well **tested:**
Automated & manual tests, testivals, codescanning

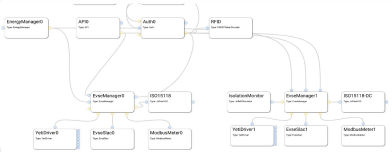
Visual Configuration Editor
Example configurations:

Simple AC or DC Charger

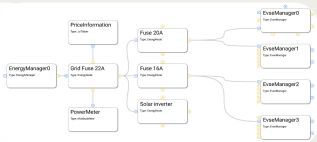
SW-in-the-loop testing



AC+DC Hybrid Charger



Site Energy Mgmt



Distributed charging park

Deeply integrated HW reference designs available:

CPUs supported & tested so far:

- Raspberry Pi CM4
- AM6X Sitara
- NXP i.MX6 / i.MX8
- ... most LINUX capable systems

OS supported & tested so far:

- Yocto:
  - **(thud)**
  - **dunfell**
  - **hardknott**
  - **kirkstone**
- Debian / Ubuntu
- OpenSuse
- Arch
- Fedora
- Other Linux distributions

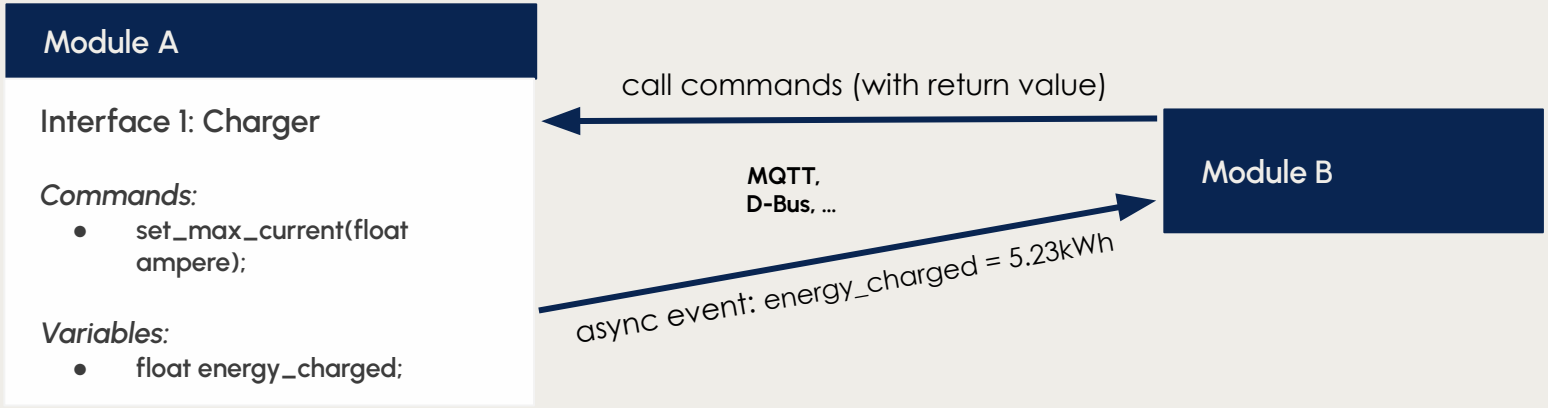
Always **up-to-date**: Online updates / OTA, **secure boot**, multiple redundant partitions

for all details, check our webinar recording: https://youtu.be/OJ6kjHRPkyY

PIONIX

2) Beautiful modular architecture: introducing EVerest Framework

Microservice architecture



Typical architecture found in many commercial solutions for EV charger software

- Each module is a separate Linux process
- Use publish/subscribe pattern (e.g. MQTT) for communication between modules

Module APIs / MQTT topics

Typical implementation

Hard coded topic paths on MQTT, human readable documentation

- Excel sheet MQTT topics
- MQTT does not standardize on data formats
- Are dependencies met?
- What if a module requires multiple instances of another module?
- What if topic path changes?

➜ hard to maintain and configure

	A	B
1	**Module: EvseManager**	
2	*MQTT topic*	*Description*
3		
4	evse_manager/commands/pause	Publish "true" to this topic to pause charging
5	evse_manager/data/progess	Subscribe to this topic to get progress. Example:
6		{ "state": "Charging" "energy": 23.5 }
7		

VS.

EVerest Framework

Machine readable definition of module interfaces, dependencies and data types

- Automatic dependency checking (with versioning)
- Graphical configuration tool to connect modules
- Auto generate human readable documentation
- Native C++17 support, JS, Python, Rust, to abstract communication
- Auto complete for C++, compile time type safety
- Can spread over multiple computers/chargers - EVerest can run on a whole charging park

```yaml
description: EVSE Manager
config:
  charge_mode:
    description: Select charging mode
    type: string
    enum:
      - AC
      - DC
    default: AC

provides:
  evse:
    description: This is the main evsemanager interface
    interface: evse_manager

requires:
  ac_rcd:
    interface: ac_rcd
    min_connections: 0
    max_connections: 1
  bsp:
    interface: evse_board_support
```

PIONIX

3) Seamless buildings blocks for all use cases: build an AC charger in 2 minutes

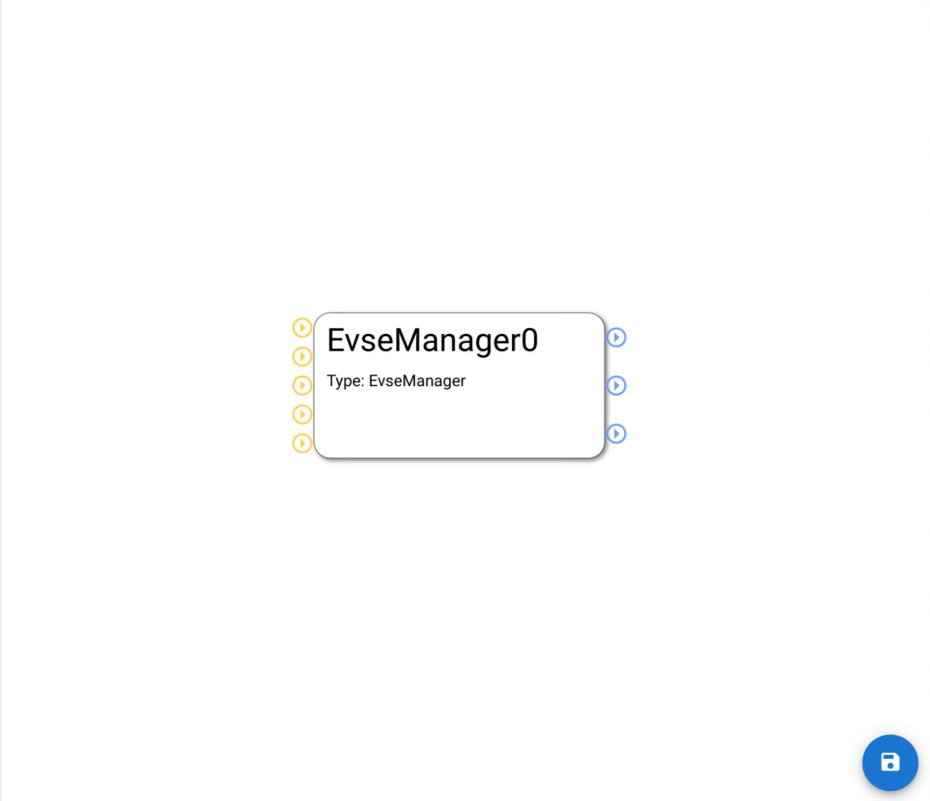
Build a simple AC Wallbox
with ISO 15118 and OCPP

https://youtu.be/iUKniRe4PKA

Build a simple AC Wallbox
Step 1 of 7

EvseManager

- One charging connector
- Charging logic and session
- Orchestrates all other modules access to this one connector

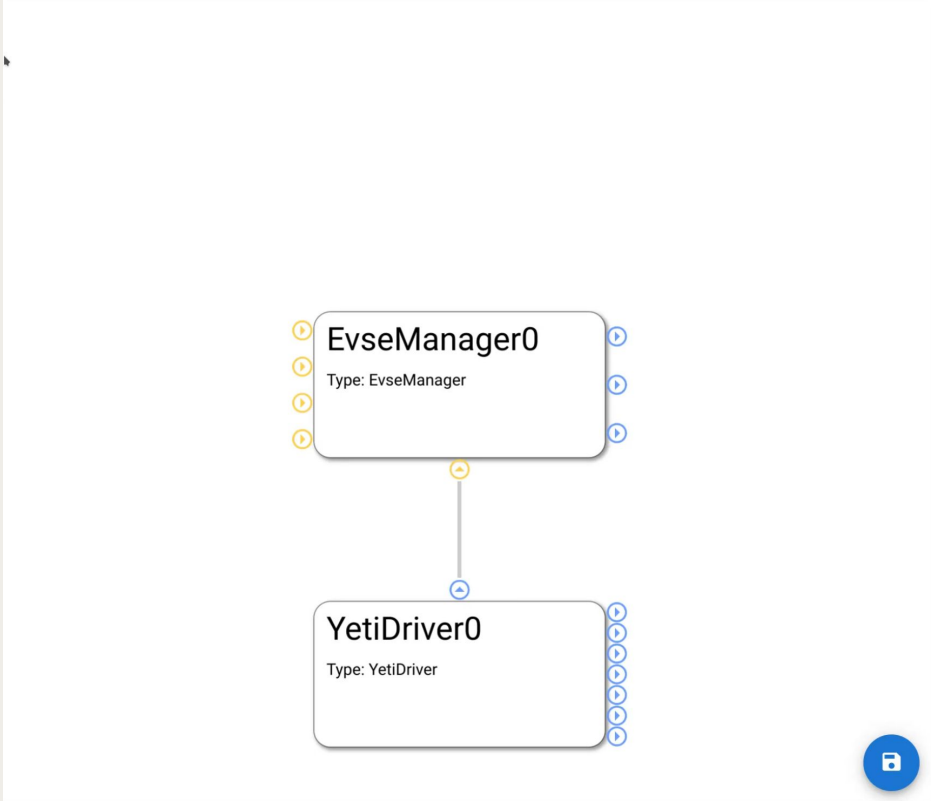


Build a simple AC Wallbox
Step 2 of 7

Board support

Hardware driver:

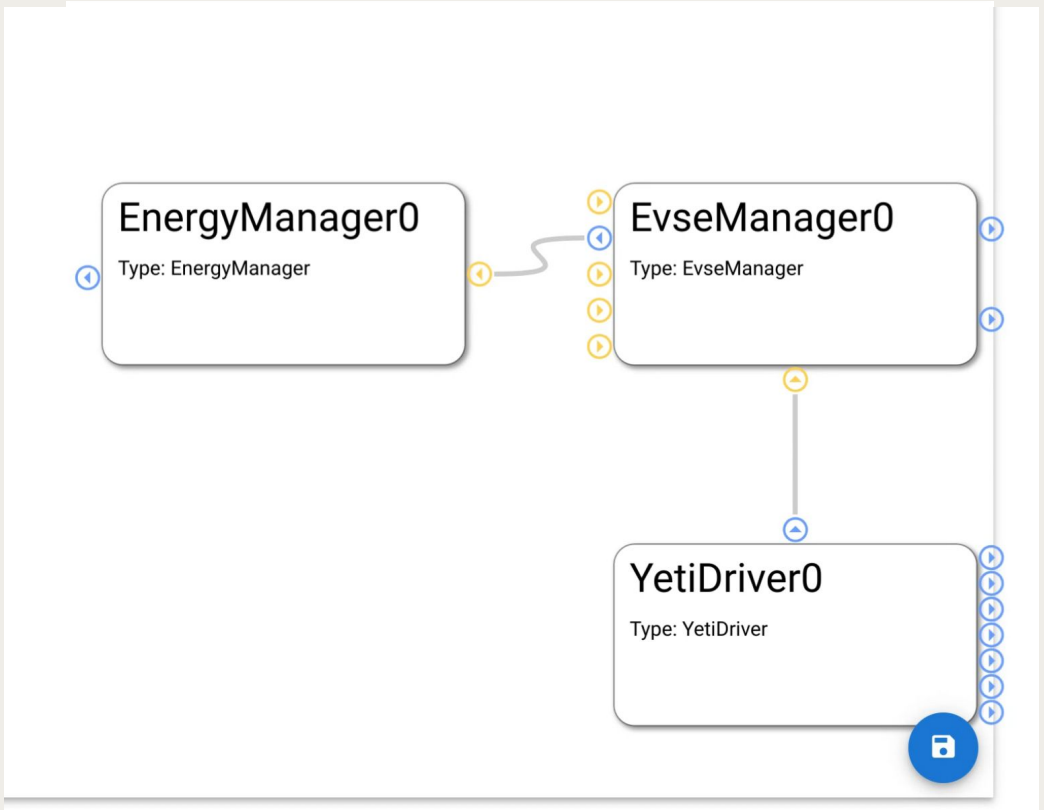
- CP, Relais, RCD



Build a simple AC Wallbox
Step 3 of 7

Energy manager

- Minimal configuration, more advanced later



Build a simple AC Wallbox
Step 4 of 7

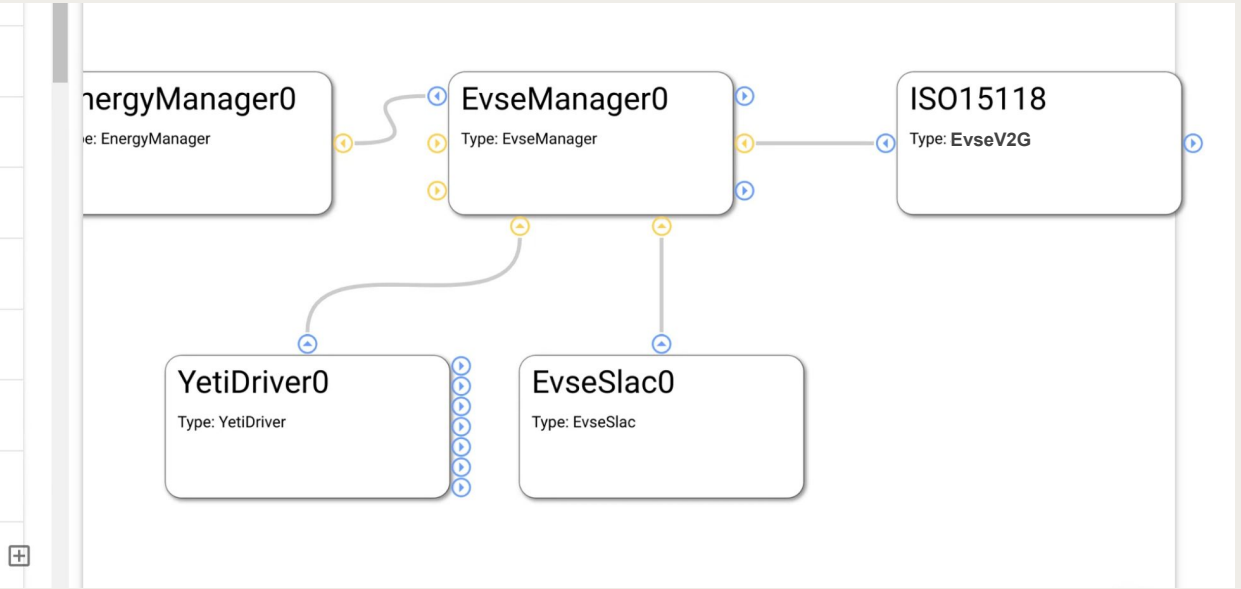
Add ISO15118-2

- Protocol stack

  currently: EvseV2G, commercial options

- Upcoming: New C++ ISO15118-20 implementation

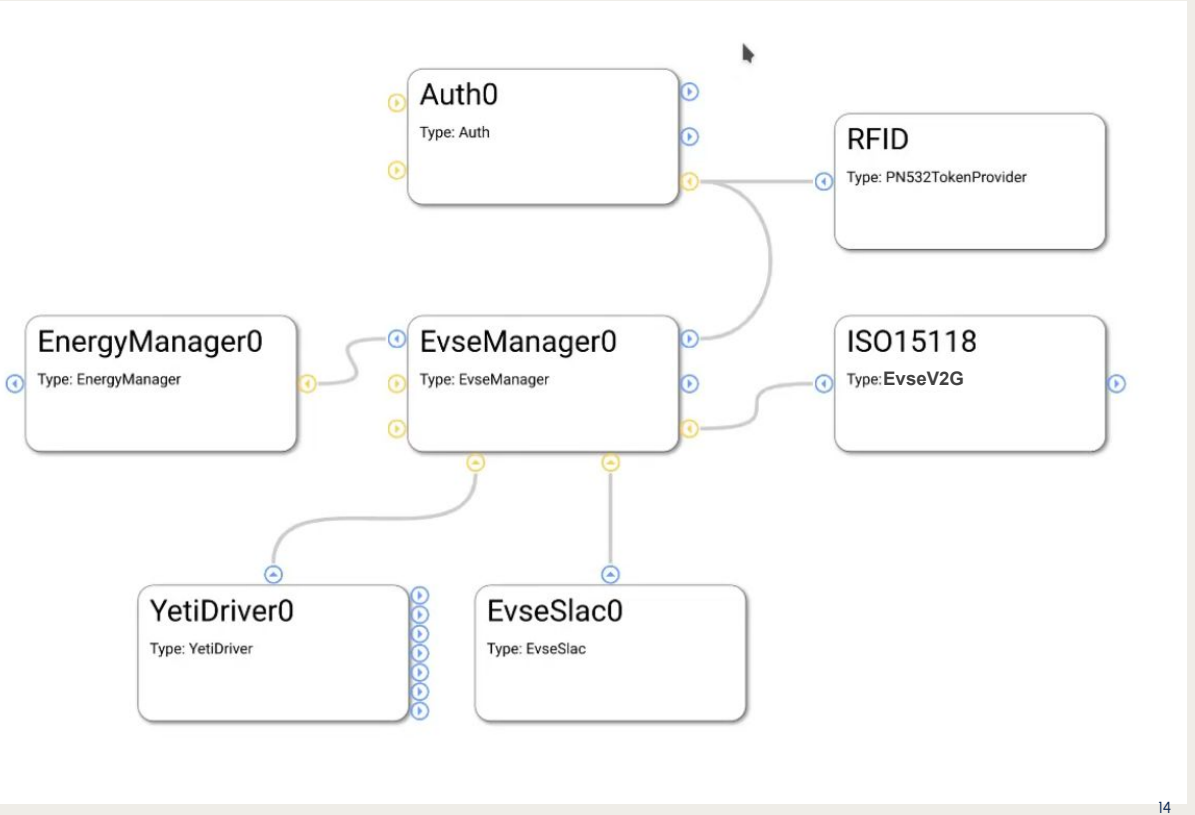
- SLAC (EVerest C++ implementation)



Build a simple AC Wallbox
Step 5 of 7

Auth manager needs

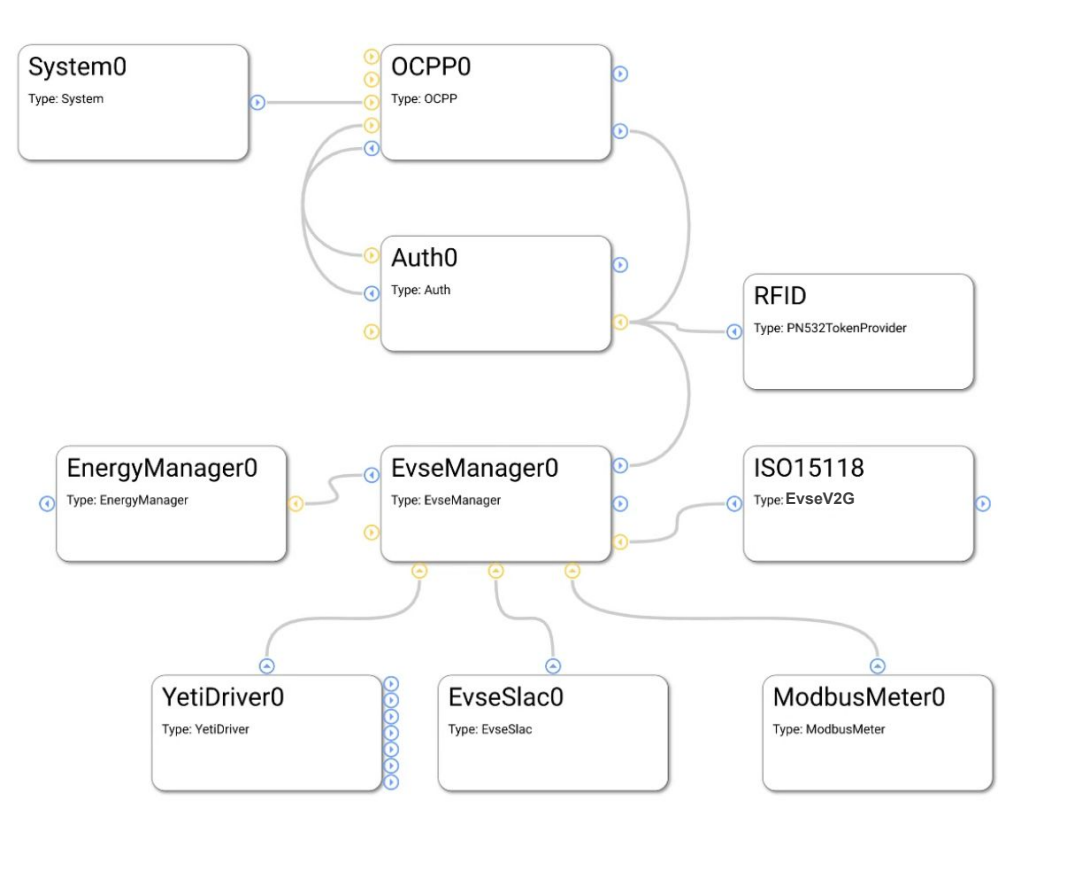
- Token providers (output tokens) and token validators (can check if token is valid)
- We add two token providers here:
- RFID (new module)
- Autocharge (EvseManager also has a token provider interface for EVCCID)



Build a simple AC Wallbox
Step 6 of 7

Add Cloud backend

- OCPP 2.0.1+1.6J module
- Powermeter (may also support German Eichrecht)
- System module supports reboot/firmware update etc via OCPP

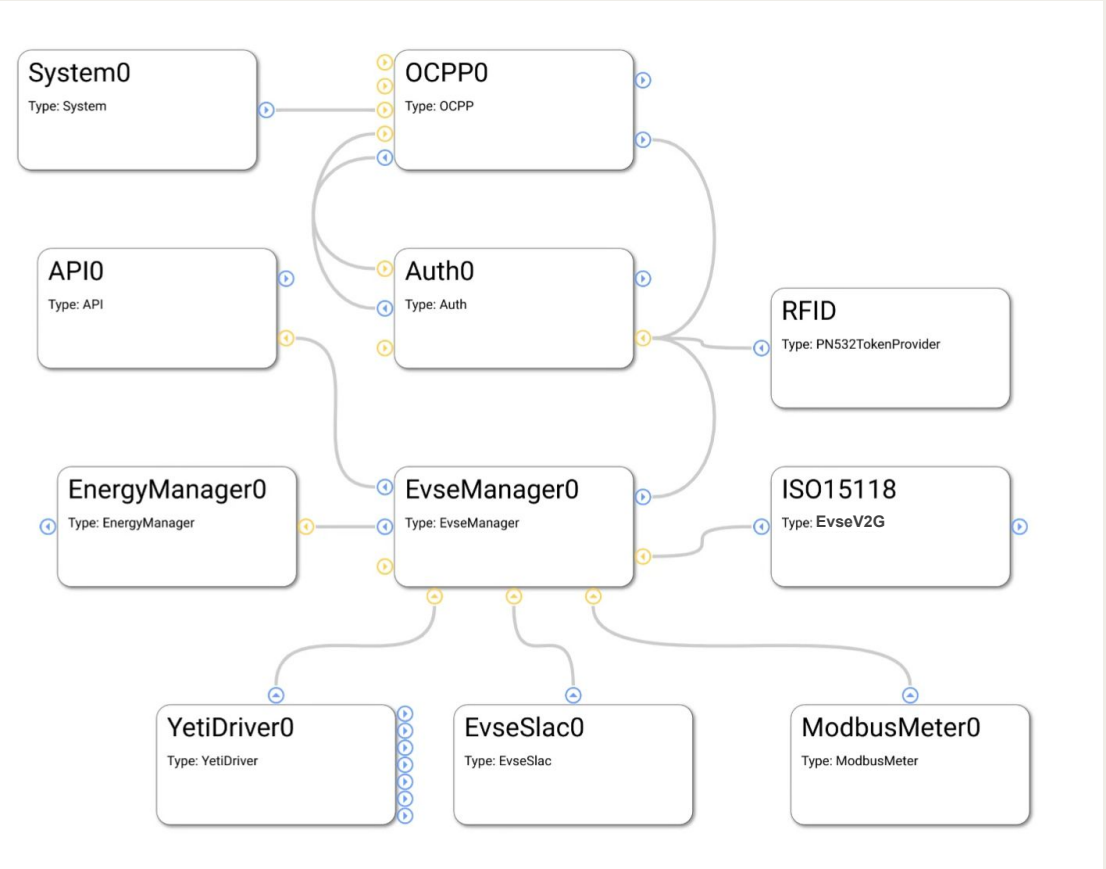


PIONIX

Build a simple AC Wallbox
Step 7 of 7

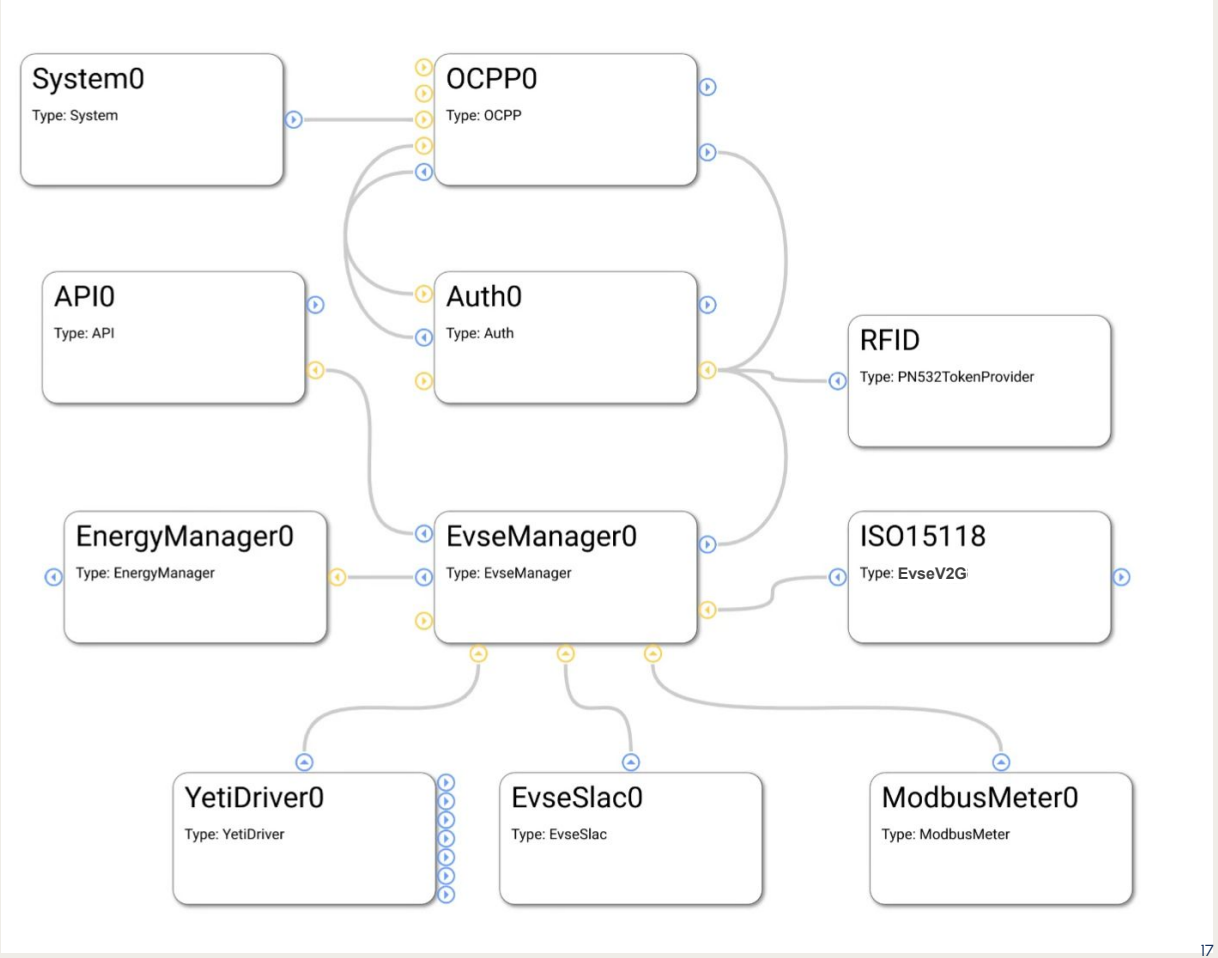
Add API for ext. applications

- (display app, mobile phone app)



Build a simple AC Wallbox **Completed!**

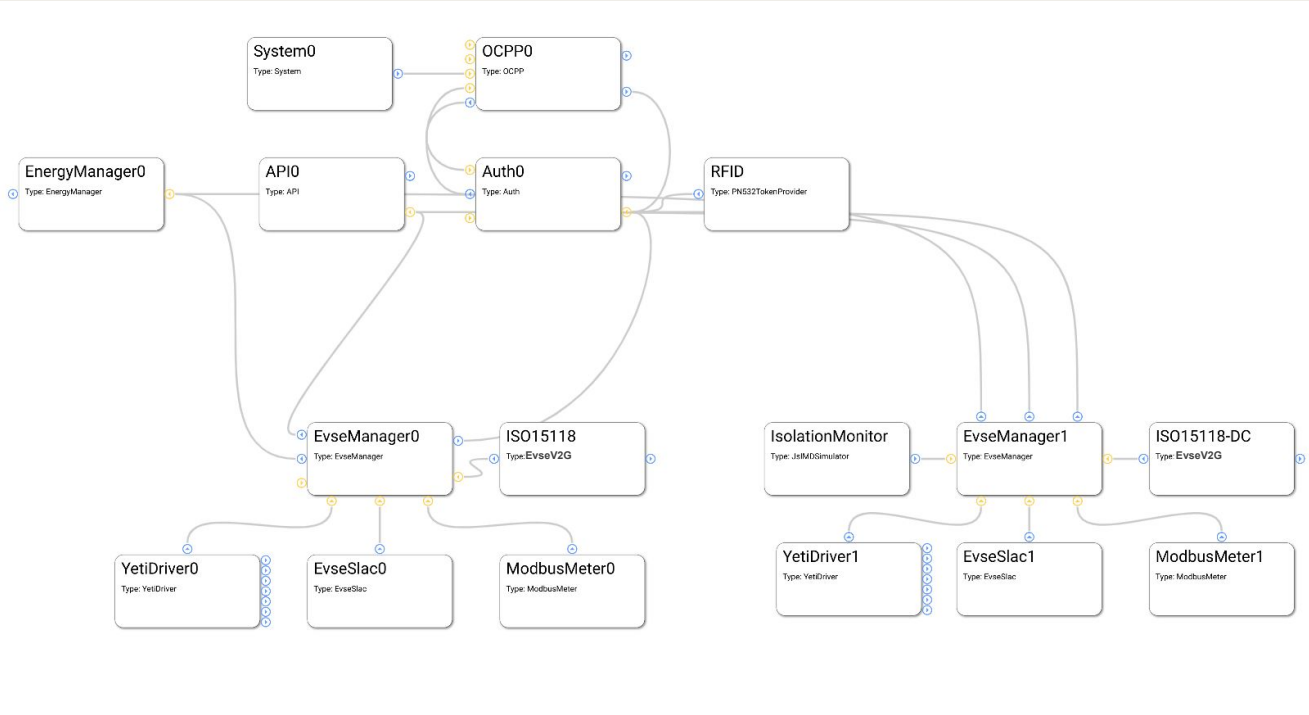
Now EVerest is ready to run for your ISO15118 / OCPP ready AC wallbox!



Advanced config
DC Charging

Adding a second DC port to the charging station

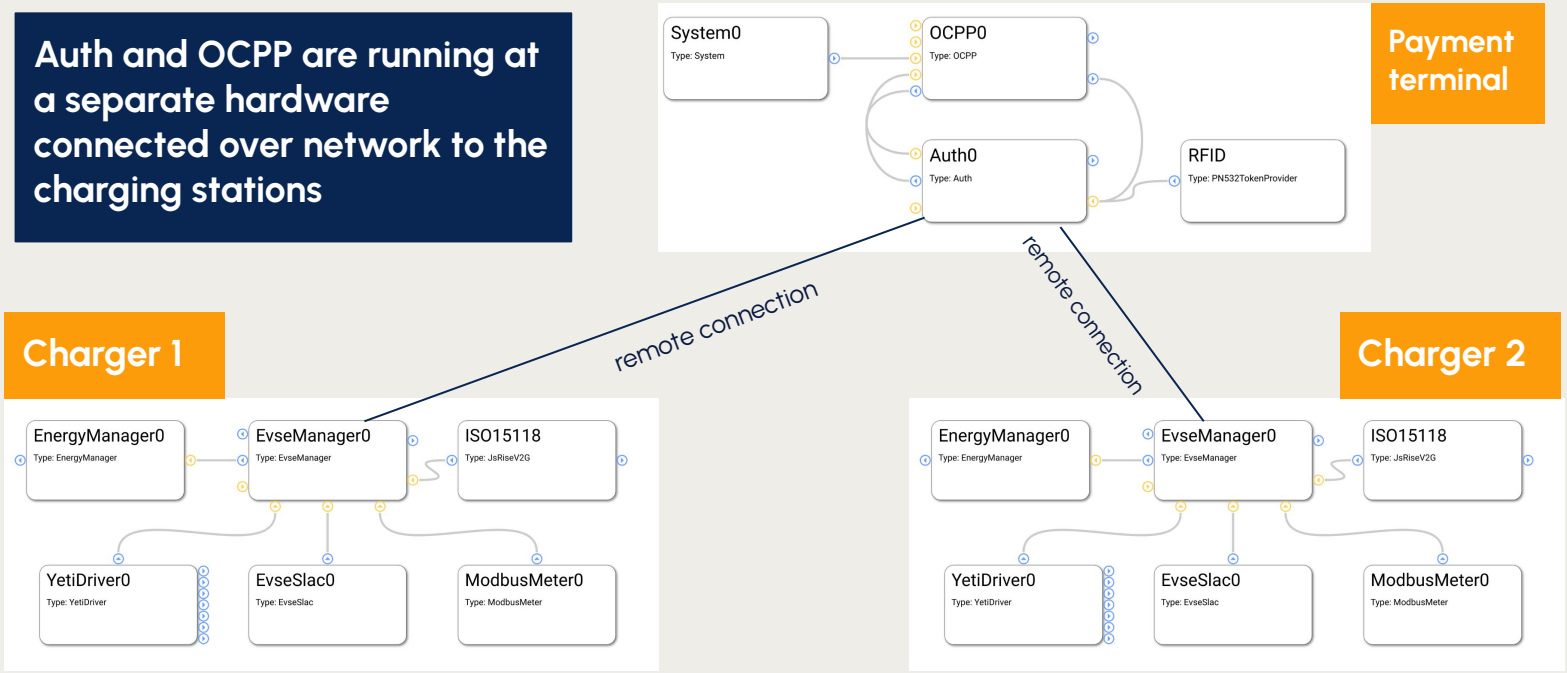
- The two ports share:
- EnergyManager (load balancing)
- OCPP (shown as two connectors in backend)
- API (App will see two ports)



PIONIX

Advanced configuration
Central payment and OCPP terminal

Auth and OCPP are running at a separate hardware connected over network to the charging stations



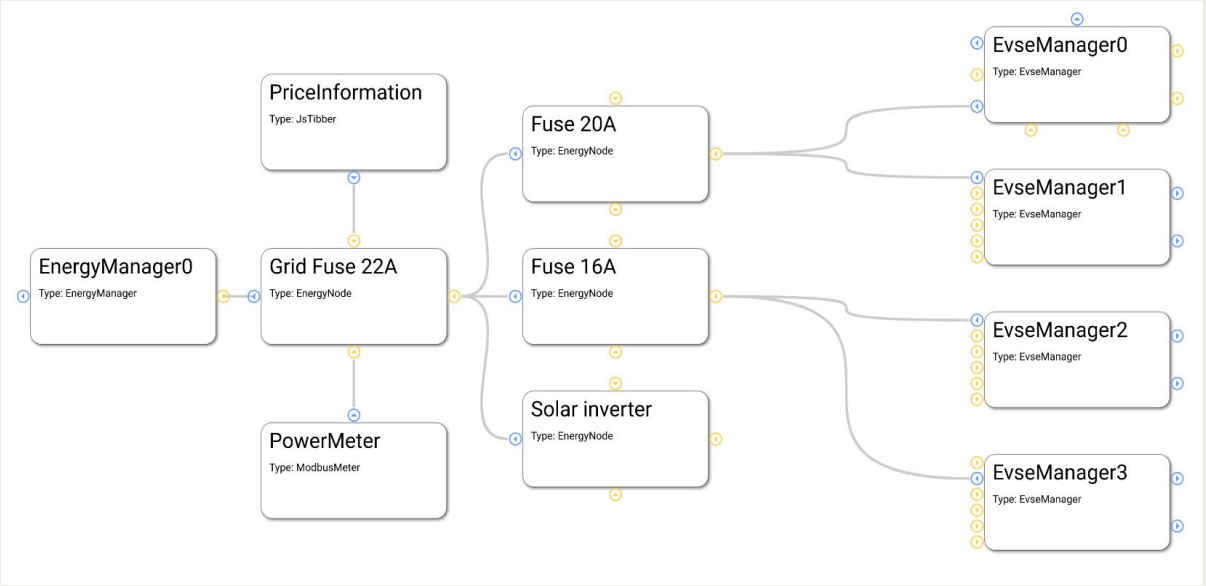
PIONIX

Advanced configuration
Central energy management

Energy manager

Complex energy distribution trees can be represented to **load balance multiple charging stations**.

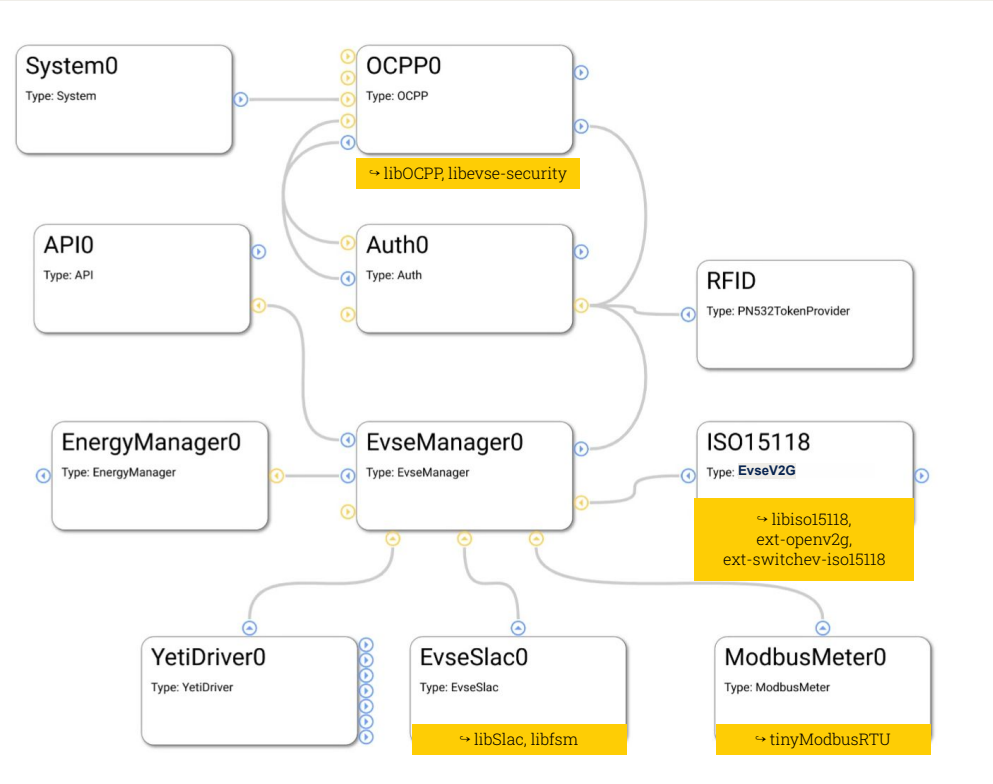
Load sharing will work across the tree with **different optimizer targets** for each car.



EVerest is a module system AND a huge set of supporting libraries

https://github.com/EVerest

- 32 public modules
- 34 Repositories
- ~720k lines of Code
- easily extendable



PIONIX

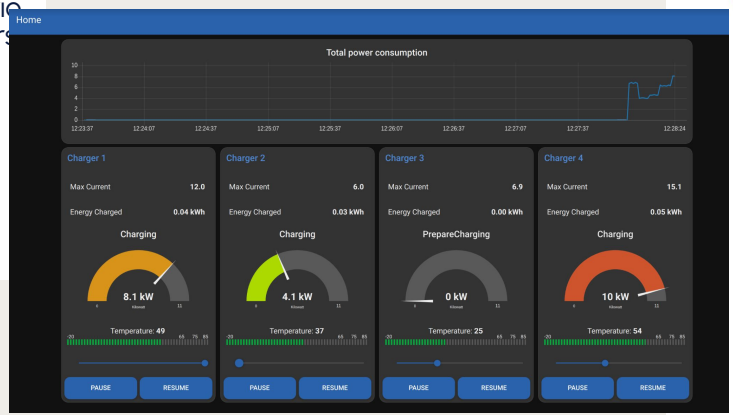
4) Testing!
SIL - develop on single laptop

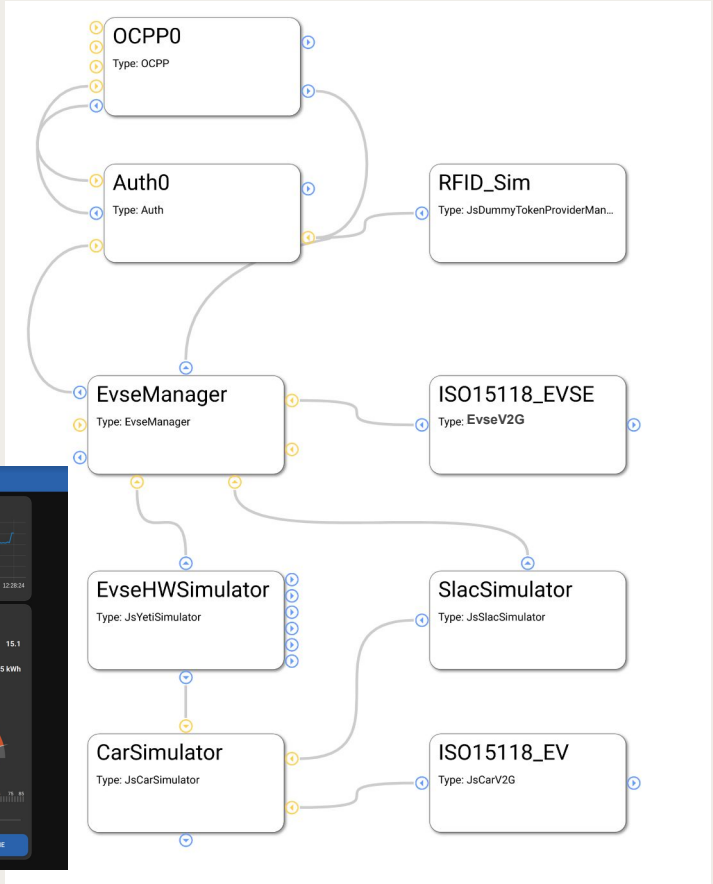
Reference HW samples

Software in the loop simulation (SIL)
Running EVerest without HW or a car

Run complete EVerest on your laptop - load simulated EVSE hardware as well as car simulation

- Simulates Control Pilot signal and ISO15118 between simulated car and evse
- Test complex scenarios with multiple chargers etc.
- Use Node Red for quick UIs for development purposes





Get started with your development kit

PIONIX BelayBox

The PIONIX BelayBox is a complete 11/22kW AC charging station designed for Developers. It comes with EVerest pre-loaded and gets you started within minutes:

Yak High Level Board (Compute):

- Raspberry Pi CM4 compute module
- 5" IPS display, 1000 nits sunlight readable, capacitive touch
- QCA7005 PLC GreenPHY modem
- NFC/RFID Reader
- USB, RS232/485, Ethernet, WiFi, Bluetooth, CAN, ...

Yeti Power Board:

- safety critical code runs on separate STM32 MCU
- 1ph/3ph automatic switching during charging
- GPS (on request)
- optional metering Display
- 6mA DC RCD module
- Integrated power meter (non MID)
  3ph full waveform U/I

All in a wall-mountable IP44 case with metal cable holder.
Full spec sheet available here

all open HW resources: https://github.com/PionixPublic/reference-hardware

µMWC - Micro Megawatt Charger

- perfect for full communication and charging session testing incl. isolation monitor etc.
- CCS: up to 1250V, up to 0.8mA, up to 1W
- Local OCPP backend
- Battery Powered
- also AC protocol testing (without power delivery)
- Ping us if you want one ;-)
  contact@pionix.de

PIONIX LFENERGY EVerest

PIONIX

LF ENERGY EVerest

PIONIX

Minimal EVerest providing OCPP

Bridge to EVerest 1

Charger Stack -> EVerest

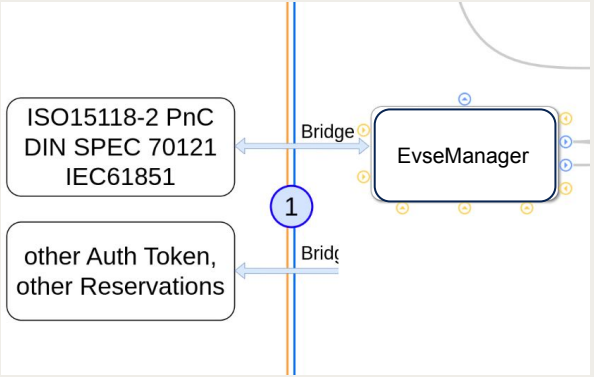
- Events (Car plugged in, transaction started(5kWh), ...)
- Power meter readings
- ...

EVerest -> Charger stack

- Control PWM duty cycle
- Allow power on
- ...

optional things (System module):

- Trigger firmware updates/reboots via OCPP etc



Complete yaml interface descriptions here (not everything has to be implemented)

evse_manager:

interfaces/evse_manager.yaml
types/evse_manager.yaml

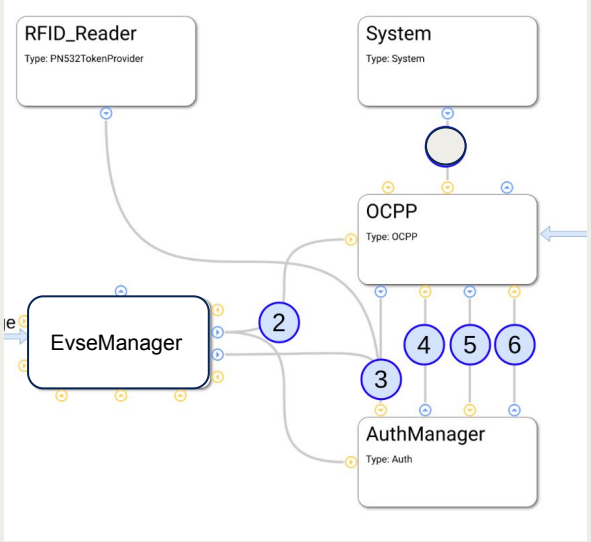
token_provider (for PnC):

interfaces/auth_token_provider.yaml
types/authorization.yaml

EVerest OCPP subsystem business logic

Example logical flow (car plug in before auth):

- **EvseManager** stack starts session, throws "CarPluggedIn/SessionStarted" event (2)
  - OCPP notifies CSMS (StatusNotification.req(occupied))
  - Auth manager internally tracks session
- **AuthManager** gets Auth token (3) (from EvseManager/PnC, RFID, OCPP remote start)
  - Checks against reservations. New reservations come in on (4)
  - Sends out to all connected token validators (5) (could be more)
- **OCPP** receives auth_token_validator request
  - asks AuthCache/AuthList/CSMS whether token is valid, responds ACCEPTED (5)
- **AuthManager** informs EvseManager that it is authorized (2)
- **EvseManager** starts charging, throws event "Transaction started" (2)
  - OCPP starts transaction at CSMS
  - AuthManager also tracks transaction
- **EvseManager** stack stops charging (full, local stop, OCPP remote stop), throws "Transaction finished" event (2)
  - OCPP stops transaction at CSMS and reports charged amount of kWh
- **EvseManager** stack throws event "Car unplugged/SessionFinished" (2)
  - OCPP notifies CSMS (StatusNotification.req(available))
  - Authmanager switches port to available and delete Auth token



Reason for AuthManager and OCPP split: EVerest can be used without OCPP (local auth whitelists, direct card payment, …)

EvseManager drives Auth and OCPP state machines

PIONIX

OCPP2.0.1 functional block overview

A. Security

- Websocket TLS Connection,
- Certificate Management (interaction w secure storage)

B. Provisioning

- DeviceModel covered by SQLite
- DeviceModel DB can be written by other modules
- Reset driven by System module

C. Authorization / H. Reservation

- OCPP is AuthProvider: RemoteStartTransaction.req
- OCPP is AuthValidator (Authorize.req, AuthCache, AuthList)
- Auth module coordinates providers and validators and provides and withdraws authorization to EvseManager
- Plug&Charge: (Interaction OCPP + Auth + EvseManager)

D. LocalAuthListManagement

E. Transactions

- Main state machine in EvseManager
- State machine of EvseManager drives libocpp state machine and respective transaction messages

F. RemoteControl

G. Availability

I. TariffAndCost

J. MeterValues

K. SmartCharging

- libocpp contains interval merging algorithms
- libocpp can share composite schedules with EvseManager

M. ISO15118 Certificate Management

O. DisplayMessage

P. DataTransfer

L. FirmwareManagement / N. Diagnostics

- OCPP initiates update / diagnostics upload
- System updates / uploads and notifies OCPP about current status

Color indicates which module is driving the business logic

- OCPP
- Auth
- EvseManager
- System

Advantages of running a minimal EVerest over using libocpp

- Easy switch between OCPP1.6 / 2.0.1 / 2.1: bridge can remain the same
- Internal OCPP logic is completely hidden,
  would work even with another protocol instead of OCPP
- Potential to use more EVerest modules later on
  (e.g. ISO15118-20 V2X, German Eichrecht, Energy management, …)
- Avoids integration work for Authorization, Reservation and System functions
  (Reset, Diagnostics)

PIONIX

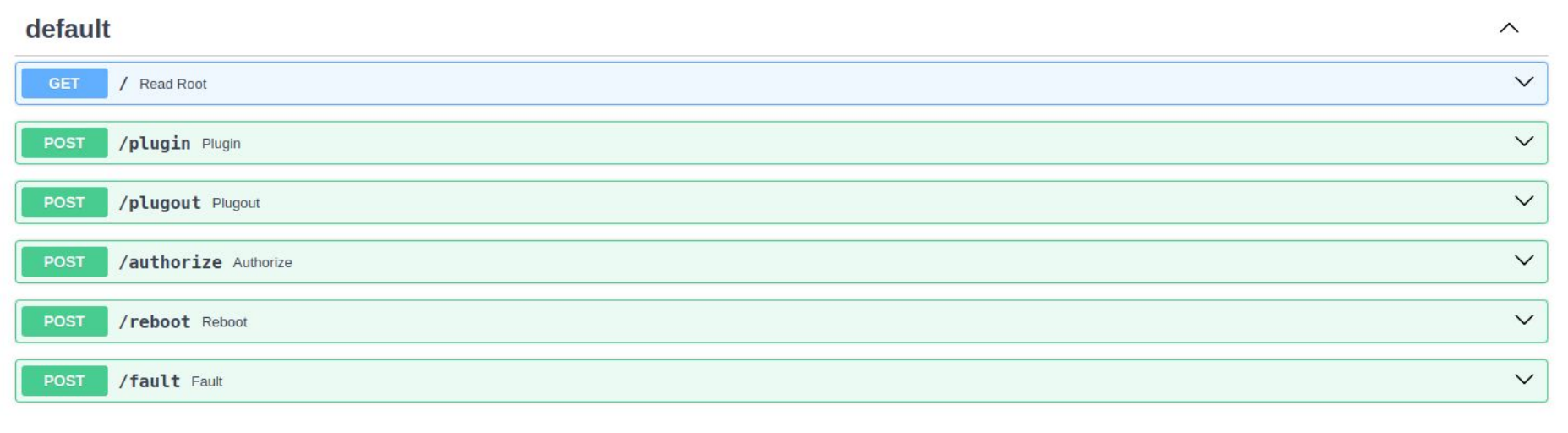
OETZI

OETZI - Open EVerest Testing Zone Instance

- Enables Automated Testing / Simulation
- Events as “plugin” or “authorize” can be triggered via HTTP API automatically
- OETZI can be deployed fully remote on a server
- Targeting:
  - CSMS makers to continuously testing their implementation against a fully simulated and remote controllable EVSE+EV
  - pre testing new OCTT releases

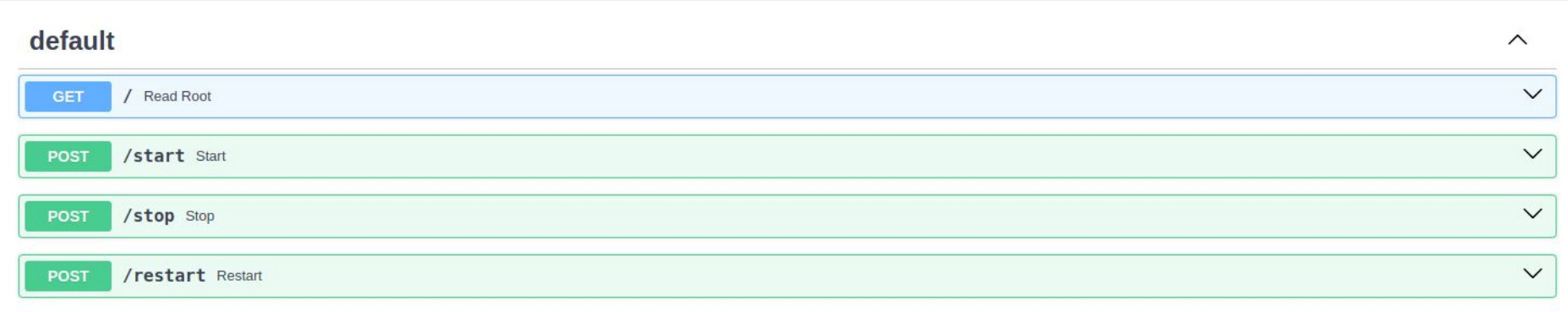
PIONIX

OETZI - CP API



PIONIX

OETZI - Argali API



PIONIX

Q & A

PIONIX

Appendix

PIONIX

Timelines

EVerest - Roadmaps
...depending on Community priorities

Short Term: (next quarter)

- **ChargeX (error code harmonization)**
  First coverage done!
- **Rust & ZVT**
  First coverage done!
- **CHAdeMO library stump**
- **⇒ OCPP/ ISO 15118 Details next slides**

Continuously:

- Adding new drivers for components and adding cars and clouds to compatibility list

Mid Term: (~12 months+)

- **OCPP 2.1 complete implementation**
- **Remote connections**
- **CHAdeMO / GB/T / ChaoJi**
- **Advanced Energy Management**
  - **load balancing**
  - **solar integration**
  - **dynamic pricing**
- **EEBus**
- **IEEE 2030.5**
- **openADR / USEF**
- **OCPP server-client (local energy mgmt in the middle)**

PIONIX

OCPP2.0.1 / 2.1 Timeline

...depending on Community priorities

Today	**OCPP 1.6J fully** implemented & tested with all whitepapers, OCPP 2.0.1 Core implementation & advanced security **OCPP 2.0.1 ready for core certification**: Authorization, Configuration, Transactions, RemoteControl, Security
Q1/24	OCPP2.1 BPT Support: NotifyEvChargingNeeds.req, SetChargingProfile.req OCPP error code harmonization: US National Charging Experience Consortium
Q1/24	SmartCharging, ISO15118 Certificate Management, Plug&Charge, FirmwareManagement, Reservation, LocalAuthList, DataTransfer
Q2/24	Full and stable OCPP2.0.1 Implementation* TariffAndCost, DisplayMessage Minimum required error codes (MREC)
Q3-Q4/24	OCPP 2.1 full implementation

*TBD with US JOET

ISO 15118-20 **Timeline**

...depending on Community priorities

Today	• Fully tested and operational ISO 15118-2 / DIN SPEC 70121 implementation, bidirectional PoC based on C/C++ implementation by Chargebyte/Pionix • ISO 15118-2 DC BPT (SAE J2847/2) • C++ based ISO 15118-20 for EVSE side (EXI + state machine) for DC & DC BPT
Q1/24	• stable release DC & DC BPT (native C++) • beta release AC & AC BPT (native C++) • car simulator for ISO 15118-20 (based on JOSEV community edition)
Q2/24	• stable release AC & AC BPT • car simulator natively in EVerest
Q3/24	• beta release missing ISO15118-20 components: schedule renegotiation, pause/resume, dynamic mode, etc.
TBD, work started	*ISO 15118-8 (Wifi communication)*

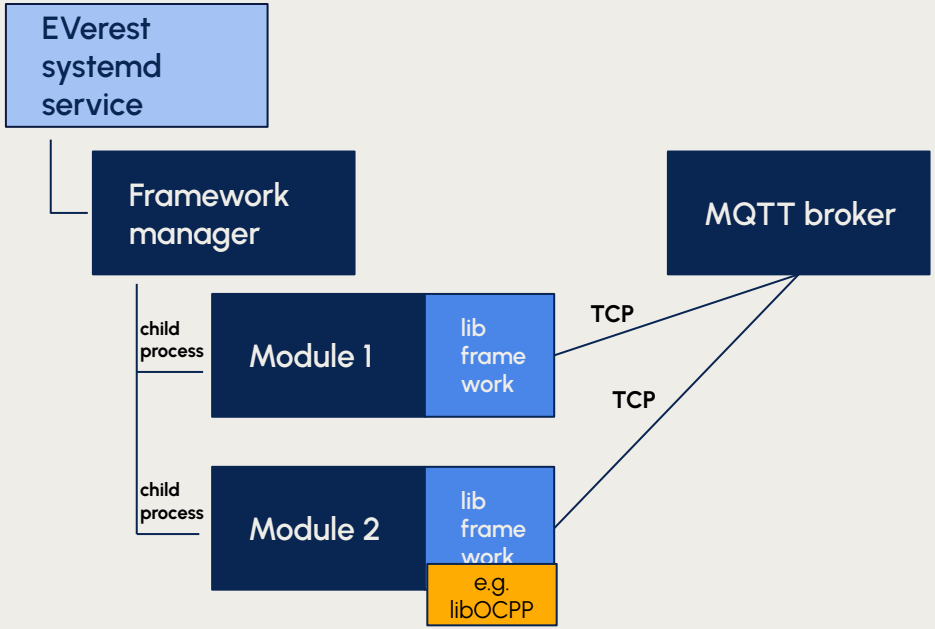
PIONIX

EVerest framework Infrastructure - Integrating OCPP

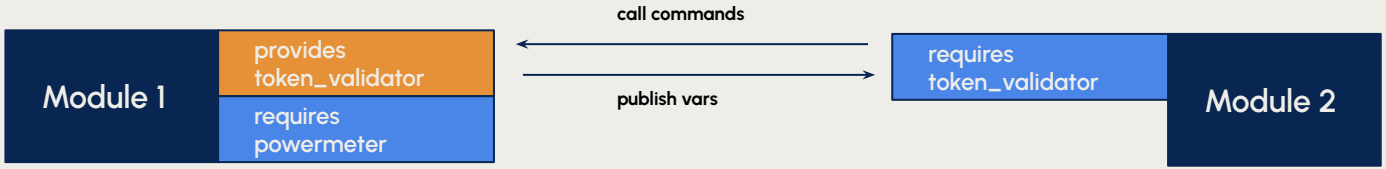
Infrastructure

Everest framework provides two things:

1) manager process:
   - reads config file for this EVerest instance
   - checks dependencies between modules
   - spawns/monitors modules as child processes
2) library linked to each module:
   - abstracts inter-module communication between requirements and providers
   - provides configuration variables
   - logging



Logical view



Infrastructure

- EVerest can interface directly with the HW (CAN bus etc)
- **To connect to existing SW outside of EVerest we use software bridge modules**
- BSP module:
  - Adapts outside protocol to internal EVerest interfaces
  - Translates outside world logic to internal EVerest logic