Learn EV Charging

OCPP · VERSION 2.0.1

The OCPP 2.0.1 Field Guide

The message model, the transaction lifecycle, the device model, and the traps — the reference a CSMS or charger-firmware engineer actually needs.

A practitioner's reference to the Open Charge Point Protocol — how a charging station and its management system talk.

learnevcharging.com

FIELD GUIDE

What's inside

OCPP 2.0.1 is a big step up from 1.6 — a unified transaction message, a real device model, and security profiles. This guide is the map plus the traps, for the desk next to your editor.

HOW TO USE THIS

Everything here matches OCPP 2.0.1 (JSON over WebSocket). Where 2.0.1 differs sharply from 1.6, it's flagged. This is a reference, not a spec replacement — the normative spec lives with the Open Charge Alliance.

01 · THE MENTAL MODEL

Station ↔ CSMS, and 1.6 vs 2.0.1

OCPP (Open Charge Point Protocol, from the Open Charge Alliance) is the language between a **Charging Station** and the **CSMS** — the Charging Station Management System, the back-office platform that authorizes drivers, starts and stops sessions, and collects meter data. Both sides can initiate messages.

Charging Station

The physical unit. Contains one or more **EVSEs**, each with one or more **Connectors**. It reports state and meter data, and executes commands.

CSMS

The back office (ChargePoint, Driivz, AMPECO, Monta, EV Connect, and others). It authorizes tokens, configures the station, and pushes smart-charging and firmware.

The hierarchy

```
Charging Station
 └─ EVSE  (id 1, 2, …)     # a point that can charge one car
     └─ Connector (id 1, …) # a physical plug on that EVSE
```

EVSE id 0 is special: it addresses the *whole station* (used in the device model and some status reports). Real EVSEs are numbered from 1, connectors from 1.

What changed from 1.6

- **One transaction message.** 2.0.1 replaces 1.6's `StartTransaction`, `StopTransaction`, and `MeterValues` with a single `TransactionEvent`.
- **A real device model.** Flat key-value config becomes structured *components* and *variables*.
- **Security profiles.** TLS and certificate handling are now first-class.
- **ISO 15118 / Plug & Charge** support is built in.

> **KEY**
>
> 2.0.1 is **JSON over WebSocket** only (no SOAP). If you're carrying 1.6 habits over, most of the friction below comes from expecting 1.6 messages that no longer exist.

OCPP-J framing over WebSocket

The station opens a **WebSocket** (usually `wss://`) to the CSMS, negotiating the subprotocol `ocpp2.0.1`. The station's identity is in the URL path. Every message is a JSON **array** — not an object — of one of three shapes.

CALL — A REQUEST (EITHER SIDE MAY SEND)

```
[ 2, "19223201", "BootNotification", { …payload… } ]
  |   |             |                   |
  |   |             |                   └ payload object
  |   |             └ action name
  |   └ unique message id (echoed in the reply)
  └ MessageTypeId: 2 = CALL
```

CALLRESULT — A SUCCESS REPLY

```
[ 3, "19223201", { …payload… } ]      # same id as the CALL
```

CALLERROR — A FAILURE REPLY

```
[ 4, "19223201", "NotSupported", "…description…", { } ]
```

> **GOTCHA**
>
> The message is a positional *array*, and the reply must carry the *same* unique id as its request. Parsing it as an object, or generating a fresh id for the reply, breaks the correlation the whole protocol relies on.

Both sides act as client and server: the station sends `BootNotification` / `TransactionEvent`; the CSMS sends `SetChargingProfile` / `GetVariables` / `Reset`. Same three frame types in both directions.

Boot & status lifecycle

1. **BootNotification**
   On power-up the station announces itself with its `chargingStation` (model, vendorName) and a `reason` (PowerUp, FirmwareUpdate…). The CSMS replies with `status`, `currentTime`, and a heartbeat `interval`.

2. **Accepted / Pending / Rejected**
   `Accepted` = go. `Pending` = stay connected while the CSMS configures you (send nothing transactional yet). `Rejected` = wait and retry after the interval.

3. **Heartbeat**
   Every `interval` seconds the station sends `Heartbeat`; the CSMS returns `currentTime` so the station can keep its clock aligned.

4. **StatusNotification**
   Whenever a connector changes state, the station reports `connectorStatus` with its `evseId` + `connectorId`.

BOOTNOTIFICATION — REQUEST → RESPONSE

```
[2,"boot-1","BootNotification",{
  "reason": "PowerUp",
  "chargingStation": { "model": "CS-100", "vendorName": "Acme" }
}]
[3,"boot-1",{
  "currentTime": "2026-07-10T09:00:00Z",
  "interval": 300,
  "status": "Accepted"
}]
```

Connector statuses (2.0.1): Available · Occupied · Reserved · Unavailable · Faulted. Status lives at the connector; the operational state of a *session* is separate (see chargingState, next page).

04 · WORKED EXAMPLE

TransactionEvent — the unified message

The heart of 2.0.1. One message type carries a transaction's whole life via `eventType`: **Started**, **Updated**, **Ended**. Each event names the `triggerReason` that caused it and a per-transaction `seqNo`.

TRANSACTIONEVENT — A "STARTED" EVENT

```
[2,"txn-1","TransactionEvent",{
  "eventType": "Started",
  "timestamp": "2026-07-10T09:14:02Z",
  "triggerReason": "CablePluggedIn",
  "seqNo": 0,
  "transactionInfo": {
    "transactionId": "T-88291",
    "chargingState": "EVConnected"
  },
  "evse": { "id": 1, "connectorId": 1 },
  "idToken": { "idToken": "045A2B…", "type": "ISO14443" },
  "meterValue": [{
    "timestamp": "2026-07-10T09:14:02Z",
    "sampledValue": [{ "value": 0, "measurand": "Energy.Active.Import.Register" }]
  }]
}]
```

> **GOTCHA — USE CHARGINGSTATE**
>
> `transactionInfo.chargingState` (*Charging, EVConnected, SuspendedEV, SuspendedEVSE, Idle*) tells you the real session state directly. Re-deriving it from meter deltas is error-prone — read the field.

`seqNo` is monotonic per transaction (0, 1, 2…). The CSMS uses it to order events and detect gaps from an offline period.

Authorization & idTokens

Before a session bills, the driver is authorized. The station sends an `Authorize` request with an `idToken`; the CSMS returns an `idTokenInfo.status`. An `idToken` is always an object — a value *plus a type*, never a bare string.

AUTHORIZE — REQUEST → RESPONSE

```
[2,"auth-1","Authorize",{
  "idToken": { "idToken": "045A2B…", "type": "ISO14443" }
}]
[3,"auth-1",{
  "idTokenInfo": { "status": "Accepted" }
}]
```

idToken types

`ISO14443` (RFID card UID) · `ISO15693` · `KeyCode` (PIN) · `Local` · `MacAddress` · `eMAID` (ISO 15118 contract ID) · `Central` (CSMS-initiated) · `NoAuthorization`.

Status values

`Accepted` · `Blocked` · `Expired` · `Invalid` · `ConcurrentTx` · `NoCredit` · `NotAllowedTypeEVSE` · `NotAtThisLocation` · `NotAtThisTime`.

> **KEY — LOCAL AUTHORIZATION**
>
> The station can also authorize from a cached **Local Authorization List** when the CSMS is unreachable, so charging still starts offline. Reconcile with the CSMS on reconnect.

06 · CONFIGURATION

The Device Model

1.6's flat key-value configuration is gone. In 2.0.1 everything the CSMS can read or set is a **Variable** on a **Component** — a structured, discoverable model of the station.

SETVARIABLES — CSMS CONFIGURES THE STATION

```
[2,"set-1","SetVariables",{
  "setVariableData": [{
    "component": { "name": "SampledDataCtrlr" },
    "variable":  { "name": "TxUpdatedInterval" },
    "attributeValue": "60"
  }]
}]
```

- **GetVariables / SetVariables** — read and write individual variables.
- **GetBaseReport → NotifyReport** — the station streams its full component/variable inventory, so the CSMS can discover what a given model supports.
- Variables carry attributes (`Actual`, `Target`, `MinSet`, `MaxSet`) and characteristics (type, unit, range).

> **GOTCHA**
>
> Don't port 1.6 `GetConfiguration` / `ChangeConfiguration` key names over. A 2.0.1 setting is addressed by *(component, variable)* — e.g. *SampledDataCtrlr / TxUpdatedInterval* — not a single flat key.

07 · SECURITY

The three security profiles

2.0.1 defines three security profiles. Which one applies is a deployment decision; the station and CSMS must agree.

#	Profile	What it means
1	Unsecured transport + Basic Auth	Plain WebSocket with HTTP Basic Authentication. Lowest bar — for trusted networks only.
2	TLS + Basic Auth	`wss://` encrypts the link; the station authenticates with a Basic Auth password. Server certificate only.
3	TLS + client certificate	Mutual TLS — both sides present certificates. Strongest, and the basis for managed certificate rotation.

KEY

2.0.1 also standardizes **certificate management** (install/update/delete) and the **ISO 15118** certificate flows for Plug & Charge, so a station can obtain and rotate the certificates it needs over OCPP itself.

GOTCHA

Profiles are a ladder, not a menu to mix. Moving to Profile 3 means provisioning client certificates on every station and handling rotation — plan the PKI before you flip the switch.

Top 10 gotchas that reach production

#	The trap	What's actually true
1	Looking for `StartTransaction`	Gone in 2.0.1 — start/stop/meter are all `TransactionEvent`.
2	Re-deriving session state	Read `transactionInfo.chargingState` instead.
3	Parsing a message as an object	Every message is a JSON *array*: `[2,id,action,payload]`.
4	New id on the reply	The reply reuses the request's unique id — that's the correlation.
5	Treating EVSE 0 as a real EVSE	Id 0 = the whole station; chargeable EVSEs start at 1.
6	Flat 1.6 config keys	Settings are *(component, variable)* pairs via Get/SetVariables.
7	Ignoring `Pending` at boot	Stay connected and wait for CSMS config; don't start transacting.
8	Bare-string idToken	`idToken` is an object: a value *plus* a `type`.
9	Assuming real-time delivery	Offline, the station keeps charging and *queues* events; expect gaps + `seqNo` catch-up.
10	Mismatched security profile	Station and CSMS must agree on profile 1/2/3; Profile 3 needs client certs.

THE PATTERN BEHIND THE TRAPS

Almost every one is a 1.6 habit that 2.0.1 changed. When something feels missing, ask: *is this the 1.6 way?* The 2.0.1 answer is usually "one unified message" or "the device model."

09 · REFERENCE

One-page cheat sheet

Message types

2	CALL (request)
3	CALLRESULT (success)
4	CALLERROR (failure)

Key actions

`Station→`	BootNotification, Heartbeat, StatusNotification, Authorize, TransactionEvent, NotifyReport
`CSMS→`	SetVariables, GetVariables, GetBaseReport, RequestStartTransaction, SetChargingProfile, Reset, UpdateFirmware

Connector status

Available · Occupied · Reserved

Unavailable · Faulted

chargingState

Charging · EVConnected · Idle

SuspendedEV · SuspendedEVSE

TransactionEvent

```
eventType: Started | Updated | Ended
seqNo: 0,1,2… (per transaction)
```

TRANSPORT

```
wss://…/<stationId>   subprotocol: ocpp2.0.1
Security profile 1 (Basic) · 2 (TLS+Basic) · 3 (mTLS)
```

EVSE id 0 = whole station · EVSEs and connectors numbered from 1 · every message is a JSON array.

Learn EV Charging

KEEP GOING

You've got the map. Now go deeper.

Every part of this guide has a full walkthrough — messages, payloads, and edge cases — on the site (tap to read):

- What is OCPP — the protocol, end to end
- OCPP vs OCPI vs ISO 15118 — how the layers fit together
- Every OCPP guide — the full series as it publishes

Free, vendor-neutral guides and tools for the people building EV charging — engineers, PMs, and operators.

learnevcharging.com/blog/ocpp