OCPP for Developers

Version 2.0.1 / 2.1 · verified 2026-11-02

OCPP 1.6J, 2.0.1 and 2.1 for backend developers — frames, security profiles, boot, transactions, smart charging, DER and V2X.

Contents

Mental model 2

Versions 2

The message frame 3

Connection and security 5

Boot, heartbeat and status 6

Authorization 8

Transactions with TransactionEvent 9

Smart charging 10

Firmware and diagnostics 12

OCPP 2.1 additions and certification 13

Further reading 15

Mental model

OCPP is JSON-RPC over a single WebSocket that the **charging station opens** to your Charging Station Management System (CSMS). Once the socket is up, either side can send requests, so your backend is both a server for the station's reports and a client issuing commands down the same pipe. Every charging session is a state machine the station drives with a handful of messages; your job is to persist those events in order and answer them fast.

Versions

Version	Status	Subprotocol
1.6J	Large installed base	`ocpp1.6`
2.0.1	IEC 63584 (2024)	`ocpp2.0.1`
2.1	IEC 63584-210:2025	`ocpp2.1`

2.0.1 is a redesign, not an upgrade of 1.6: transactions, device management and security all changed, so a 1.6 handler does not carry over. 2.1 was released in January

2025, builds on 2.0.1 and stays compatible with it — every 2.0.1 message still exists in 2.1. The version is negotiated in the WebSocket handshake.

```
GET /ocpp/CS-0042 HTTP/1.1
Upgrade: websocket
Sec-WebSocket-Protocol: ocpp2.0.1, ocpp1.6
```

> **Gotcha:** One CSMS endpoint often serves 1.6 and 2.0.1 stations. Dispatch on the negotiated subprotocol per connection, never on a config flag, or a mixed fleet silently mis-parses half its traffic.

The message frame

Type id	Frame	Shape
`2`	CALL	`[2, id, action, payload]`
`3`	CALLRESULT	`[3, id, payload]`
`4`	CALLERROR	`[4, id, code, text, details]`

Every message is a JSON array. The message id is chosen by the sender and echoed in the reply — that is the only link between a request and its response, because a CALLRESULT carries no action name. Payload validation failures go back as a CALLERROR with codes such as `FormatViolation`, `TypeConstraintViolation` or `NotImplemented`.

```
[2, "19223201", "Heartbeat", {}]
```

```
[3, "19223201",
  {"currentTime": "2026-11-02T10:15:00Z"}]
```

> **Tip:** Keep the pending-request map keyed by message id per connection. A CALLRESULT is meaningless without it, since you must know which action it answers to validate its payload.

Connection and security

Profile	Transport	Station proves identity by
1	Plain WebSocket	HTTP Basic password
2	TLS	HTTP Basic password
3	TLS	Client certificate

The station's identity is the last segment of the connection URL, so the URL alone tells you which charger is calling before any message arrives. OCA certification for 2.0.1 requires Security Profile 2; profile 3 is the optional Advanced Security certification profile. The OCA Security Operations Guide requires a Basic password of at least 16 characters and recommends a unique random password per station.

```
wss://csms.example.com/ocpp/CS-0042
Authorization: Basic <base64 credentials>
```

> **Warning:** Profile 1 sends the password unencrypted, and the OCA guidance is that a station shall not use it on untrusted networks. Treat it as a lab-only setting, never a production default.

Boot, heartbeat and status

Message	Direction	Purpose
`BootNotification`	Station to CSMS	Register after start
`Heartbeat`	Station to CSMS	Liveness, clock sync
`StatusNotification`	Station to CSMS	Connector state

The first message after connecting is `BootNotification`. Your response's `status` is `Accepted`, `Pending` or `Rejected`, and its `interval` sets the heartbeat period in seconds. `StatusNotification` reports each connector as `Available`, `Occupied`, `Reserved`, `Unavailable` or `Faulted`.

```
[2, "1", "BootNotification", {
  "reason": "PowerUp",
  "chargingStation": {
    "model": "AC-22",
    "vendorName": "Acme"
  }
}]
```

```
[3, "1", {
  "currentTime": "2026-11-02T10:00:00Z",
  "interval": 300,
  "status": "Accepted"
}]
```

> **Tip:** Answer `BootNotification` immediately. If the station needs configuring first, reply `Pending`, push your settings, and accept it afterwards — rather than holding the boot request open while you do the work.

Authorization

`idToken.type`	Typical source
`ISO14443`	RFID card UID
`eMAID`	ISO 15118 contract (Plug and Charge)
`Central`	Remote start from an app
`NoAuthorization`	Free-vend charger

`Authorize` asks whether a token may charge. The answer is `idTokenInfo.status`: `Accepted`, `Blocked`, `Expired`, `Invalid`, `NoCredit`, `ConcurrentTx` and a few location-specific values. The optional `cacheExpiryDateTime` controls how long the station may trust the result offline.

```
[2, "7", "Authorize", {
  "idToken": {
    "idToken": "04A2B3C4D5E6F7",
    "type": "ISO14443"
```

```
  }
}]
```

Transactions with TransactionEvent

`eventType`	Sent when
`Started`	Transaction begins
`Updated`	State change or meter sample
`Ended`	Transaction ends, with `stoppedReason`

2.0.1 replaces 1.6's `StartTransaction`, `StopTransaction` and `MeterValues` with one message. The **station** generates `transactionId`, unlike 1.6 where the CSMS assigned it in the `StartTransaction` response. Every event carries `triggerReason`, a `seqNo` and, while charging, `chargingState` such as `Charging` or `SuspendedEV`.

```
[2, "42", "TransactionEvent", {
  "eventType": "Updated",
  "timestamp": "2026-11-02T10:20:00Z",
```

```
  "triggerReason": "MeterValuePeriodic",
  "seqNo": 3,
  "transactionInfo": {
    "transactionId": "tx-8f1c",
    "chargingState": "Charging"
  }
}]
```

> **Gotcha:** Events queued while the station was offline arrive late, in a burst, with `offline: true`. Order by `seqNo` and the event `timestamp`, not by arrival, or an `Ended` can land before its last `Updated` and your billing reads the wrong final meter value.

Smart charging

`chargingProfilePurpose`	Scope
`ChargingStationMaxProfile`	Whole-station ceiling
`TxDefaultProfile`	Default for new transactions

chargingProfilePurpose	Scope
TxProfile	One running transaction
ChargingStationExternalConstraints	Set by an external system

`SetChargingProfile` sends a schedule of `chargingSchedulePeriod` entries, each a `startPeriod` in seconds and a `limit` in the `chargingRateUnit` (`W` or `A`). Profiles of the same purpose stack: the highest `stackLevel` valid at a given time wins.

```
{"evseId": 1, "chargingProfile": {
  "id": 10, "stackLevel": 0,
  "chargingProfilePurpose": "TxDefaultProfile",
  "chargingProfileKind": "Absolute",
  "chargingSchedule": [{
    "id": 1, "chargingRateUnit": "A",
    "chargingSchedulePeriod": [
      {"startPeriod": 0, "limit": 16},
      {"startPeriod": 3600, "limit": 32}
    ]}]}}
```

> **Gotcha:** A limit in `A` is per phase. 16 A on three phases at 230 V is about 11 kW, not 3.7 kW — check `numberPhases` before you convert a grid budget in watts into an amps schedule.

Firmware and diagnostics

Message	Purpose
`UpdateFirmware`	Download and install a signed image
`FirmwareStatusNotification`	Progress: `Downloading` … `Installed`
`GetLog`	Upload `DiagnosticsLog` or `SecurityLog`
`LogStatusNotification`	Upload progress

Both operations are asynchronous: the CSMS request only starts the job, carrying a `requestId`, and the station reports progress in separate notifications tagged with the same id. `UpdateFirmware` requires a download

`location` and a `retrieveDateTime`, and accepts a `signingCertificate` and `signature` so the station can verify the image.

```
[2, "88", "UpdateFirmware", {
  "requestId": 501,
  "firmware": {
    "location": "https://fw.example.com/v3.bin",
    "retrieveDateTime": "2026-11-03T02:00:00Z"
  }
}]
```

OCPP 2.1 additions and certification

Area	New in 2.1
DER control	`SetDERControl`, `NotifyDERAlarm`
V2X	`operationMode`, `dischargeLimit` in periods
Charging profiles	`PriorityCharging`, `LocalGeneration`, `Dynamic`

Area	New in 2.1
Tariffs and payment	`SetDefaultTariff`, `NotifySettlement`
Battery swap	`BatterySwap`, `RequestBatterySwap`

2.1 adds ISO 15118-20 bidirectional power transfer, so the EV can feed energy back, plus DER control: `SetDERControl` carries grid-code curves such as `VoltVar`, `FreqDroop` or `FixedPFInject`, and the station reports grid faults back with `NotifyDERAlarm`. Certification is run by the OCA: for 2.0.1, Core is mandatory and Advanced Security, Smart Charging, ISO 15118 Support and Local Authorization are optional profiles. Check the OCA site for the current 2.1 certification scope.

```
[2, "93", "NotifyDERAlarm", {
  "controlType": "HVMustTrip",
  "gridEventFault": "OverVoltage",
```

```
  "timestamp": "2026-11-02T13:05:00Z"
}]
```

> **Note:** A 2.1 CSMS still speaks to 2.0.1 stations; negotiate `ocpp2.1` first and fall back. New actions such as `SetDERControl` exist only on `ocpp2.1` connections.

Further reading

- Open Charge Alliance — OCPP downloads and overview
- OCPP 2.1 published as IEC 63584-210:2025
- OCPP Security Operations Guide
- OCPP 2.0.1 certification profiles
- mobilityhouse/ocpp — Python OCPP library with the JSON schemas