Protocols and APIs

Gateway Protocol

The Gateway WS protocol is the **single control plane + node transport** for OpenClaw. All clients (CLI, web UI, macOS app, iOS/Android nodes, headless nodes) connect over WebSocket and declare their **role** + **scope** at handshake time.

Transport

- WebSocket, text frames with JSON payloads.
- First frame **must** be a `connect` request.

Handshake (connect)

Gateway → Client (pre-connect challenge):

```
{
  "type": "event",
  "event": "connect.challenge",
  "payload": { "nonce": "…", "ts": 1737264000000 }
}
```

Client → Gateway:

```json
{
  "type": "req",
  "id": "…",
  "method": "connect",
  "params": {
    "minProtocol": 3,
    "maxProtocol": 3,
    "client": {
      "id": "cli",
      "version": "1.2.3",
      "platform": "macos",
      "mode": "operator"
    },
    "role": "operator",
    "scopes": ["operator.read", "operator.write"],
    "caps": [],
    "commands": [],
    "permissions": {},
    "auth": { "token": "…" },
    "locale": "en-US",
    "userAgent": "openclaw-cli/1.2.3",
    "device": {
      "id": "device_fingerprint",
      "publicKey": "…",
      "signature": "…",
      "signedAt": 1737264000000,
      "nonce": "…"
    }
  }
}
```

`Gateway → Client:`

```
{
  "type": "res",
  "id": "…",
  "ok": true,
  "payload": { "type": "hello-ok", "protocol": 3, "policy": { "tickIntervalMs": 1
}
```

When a device token is issued, `hello-ok` also includes:

```
{
  "auth": {
    "deviceToken": "…",
    "role": "operator",
    "scopes": ["operator.read", "operator.write"]
  }
}
```

Node example

```
{
  "type": "req",
  "id": "…",
  "method": "connect",
  "params": {
    "minProtocol": 3,
    "maxProtocol": 3,
    "client": {
      "id": "ios-node",
      "version": "1.2.3",
      "platform": "ios",
      "mode": "node"
    },
    "role": "node",
    "scopes": [],
    "caps": ["camera", "canvas", "screen", "location", "voice"],
    "commands": ["camera.snap", "canvas.navigate", "screen.record", "location.get'
    "permissions": { "camera.capture": true, "screen.record": false },
    "auth": { "token": "…" },
    "locale": "en-US",
    "userAgent": "openclaw-ios/1.2.3",
    "device": {
      "id": "device_fingerprint",
      "publicKey": "…",
      "signature": "…",
      "signedAt": 1737264000000,
      "nonce": "…"
    }
  }
}
```

Framing

- **Request**: `{type:"req", id, method, params}`
- **Response**: `{type:"res", id, ok, payload|error}`
- **Event**: `{type:"event", event, payload, seq?, stateVersion?}`

Side-effecting methods require **idempotency keys** (see schema).

Roles + scopes

Roles

- `operator` = control plane client (CLI/UI/automation).
- `node` = capability host (camera/screen/canvas/system.run).

Scopes (operator)

Common scopes:

- `operator.read`
- `operator.write`
- `operator.admin`
- `operator.approvals`
- `operator.pairing`

Caps/commands/permissions (node)

Nodes declare capability claims at connect time:

- `caps` : high-level capability categories.
- `commands` : command allowlist for invoke.
- `permissions` : granular toggles (e.g. `screen.record` , `camera.capture`).

The Gateway treats these as **claims** and enforces server-side allowlists.

Presence

- `system-presence` returns entries keyed by device identity.

Presence entries include `deviceId`, `roles`, and `scopes` so UIs can show a single row per device even when it connects as both **operator** and **node**.

›

Node helper methods

Nodes may call `skills.bins` to fetch the current list of skill executables for auto-allow checks.

Exec approvals

When an exec request needs approval, the gateway broadcasts `exec.approval.requested`.

Operator clients resolve by calling `exec.approval.resolve` (requires `operator.approvals` scope).

Versioning

`PROTOCOL_VERSION` lives in `src/gateway/protocol/schema.ts`.

Clients send `minProtocol` + `maxProtocol`; the server rejects mismatches.

Schemas + models are generated from TypeBox definitions:

- `pnpm protocol:gen`
- `pnpm protocol:gen:swift`
- `pnpm protocol:check`

Auth

If `OPENCLAW_GATEWAY_TOKEN` (or `--token`) is set, `connect.params.auth.token` must match or the socket is closed.

After pairing, the Gateway issues a **device token** scoped to the connection role + scopes. It is returned in `hello-ok.auth.deviceToken` and should be persisted by the client for future connects.

Device tokens can be rotated/revoked via `device.token.rotate` and `device.token.revoke` (requires `operator.pairing` scope).

Device identity + pairing

Nodes should include a stable device identity (`device.id`) derived from a keypair fingerprint.

Gateways issue tokens per device + role.

Pairing approvals are required for new device IDs unless local auto-approval is enabled.

Local connects include loopback and the gateway host's own tailnet address (so same-host tailnet binds can still auto-approve).

All WS clients must include `device` identity during `connect` (operator + node). Control UI can omit it **only** when `gateway.controlUi.allowInsecureAuth` is enabled (or `gateway.controlUi.dangerouslyDisableDeviceAuth` for break-glass use).

Non-local connections must sign the server-provided `connect.challenge` nonce.

TLS + pinning

TLS is supported for WS connections.

Clients may optionally pin the gateway cert fingerprint (see `gateway.tls` config plus `gateway.remote.tlsFingerprint` or CLI `--tls-fingerprint`).

Scope

This protocol exposes the **full gateway API** (status, channels, models, chat, agent, sessions, nodes, approvals, etc.). The exact surface is defined by the TypeBox schemas in `src/gateway/protocol/schema.ts`.

‹ Sandbox vs Tool Policy vs Elevated   Bridge Protocol ›

Powered by mintlify