Concept internals

# TypeBox

Last updated: 2026-01-10

TypeBox is a TypeScript-first schema library. We use it to define the **Gateway WebSocket protocol** (handshake, request/response, server events). Those schemas drive **runtime validation**, **JSON Schema export**, and **Swift codegen** for the macOS app. One source of truth; everything else is generated.

If you want the higher-level protocol context, start with **Gateway architecture**.

## Mental model (30 seconds)

Every Gateway WS message is one of three frames:

**Request**: `{ type: "req", id, method, params }`

**Response**: `{ type: "res", id, ok, payload | error }`

**Event**: `{ type: "event", event, payload, seq?, stateVersion? }`

The first frame **must** be a `connect` request. After that, clients can call methods (e.g. `health`, `send`, `chat.send`) and subscribe to events (e.g. `presence`, `tick`, `agent`).

Connection flow (minimal):

```
Client                    Gateway
   |---- req:connect -------->|
   |<---- res:hello-ok --------|
   |<---- event:tick ----------|
   |---- req:health ---------->|
   |<---- res:health ----------|
```

Common methods + events:

| Category | Examples | Notes |
| --- | --- | --- |
| Core | `connect` , `health` , `status` | `connect` must be first |
| Messaging | `send` , `poll` , `agent` , `agent.wait` | side-effects need `idempotencyKey` |
| Chat | `chat.history` , `chat.send` , `chat.abort` , `chat.inject` | WebChat uses these |
| Sessions | `sessions.list` , `sessions.patch` , `sessions.delete` | session admin |
| Nodes | `node.list` , `node.invoke` , `node.pair.*` | Gateway WS + node actions |
| Events | `tick` , `presence` , `agent` , `chat` , `health` , `shutdown` | server push |

Authoritative list lives in `src/gateway/server.ts` ( `METHODS` , `EVENTS` ).

## Where the schemas live

Source: `src/gateway/protocol/schema.ts`

Runtime validators (AJV): `src/gateway/protocol/index.ts`

Server handshake + method dispatch: `src/gateway/server.ts`

Node client: `src/gateway/client.ts`

Generated JSON Schema: `dist/protocol.schema.json`

Generated Swift models:

`apps/macos/Sources/OpenClawProtocol/GatewayModels.swift`

›

## Current pipeline

`pnpm protocol:gen`

writes JSON Schema (draft-07) to `dist/protocol.schema.json`

`pnpm protocol:gen:swift`

generates Swift gateway models

`pnpm protocol:check`

runs both generators and verifies the output is committed

## How the schemas are used at runtime

**Server side**: every inbound frame is validated with AJV. The handshake only accepts a `connect` request whose params match `ConnectParams`.

**Client side**: the JS client validates event and response frames before using them.

**Method surface**: the Gateway advertises the supported `methods` and `events` in `hello-ok`.

## Example frames

Connect (first message):

```
{
  "type": "req",
  "id": "c1",
  "method": "connect",
  "params": {
    "minProtocol": 2,
    "maxProtocol": 2,
    "client": {
      "id": "openclaw-macos",
      "displayName": "macos",
      "version": "1.0.0",
      "platform": "macos 15.1",
      "mode": "ui",
      "instanceId": "A1B2"
    }
  }
}
```

Hello-ok response:

```
  "type": "res",
  "id": "c1",
  "ok": true,
  "payload": {
    "type": "hello-ok",
    "protocol": 2,
    "server": { "version": "dev", "connId": "ws-1" },
    "features": { "methods": ["health"], "events": ["tick"] },
    "snapshot": {
      "presence": [],
      "health": {},
      "stateVersion": { "presence": 0, "health": 0 },
      "uptimeMs": 0
    },
    "policy": { "maxPayload": 1048576, "maxBufferedBytes": 1048576, "tickIntervalI
  }
}
```

Request + response:

```
{ "type": "req", "id": "r1", "method": "health" }
```

```
{ "type": "res", "id": "r1", "ok": true, "payload": { "ok": true } }
```

Event:

```
{ "type": "event", "event": "tick", "payload": { "ts": 1730000000 }, '
```

## Minimal client (Node.js)

Smallest useful flow: connect + health.

```
import { WebSocket } from "ws";

const ws = new WebSocket("ws://127.0.0.1:18789");

ws.on("open", () => {
  ws.send(
    JSON.stringify({
      type: "req",
      id: "c1",
      method: "connect",
      params: {
        minProtocol: 3,
        maxProtocol: 3,
        client: {
          id: "cli",
          displayName: "example",
          version: "dev",
          platform: "node",
          mode: "cli",
        },
      },
    }),
  );
});

ws.on("message", (data) => {
  const msg = JSON.parse(String(data));
  if (msg.type === "res" && msg.id === "c1" && msg.ok) {
    ws.send(JSON.stringify({ type: "req", id: "h1", method: "health" }));
  }
  if (msg.type === "res" && msg.id === "h1") {
    console.log("health:", msg.payload);
    ws.close();
  }
});
```

## Worked example: add a method end-to-end

Example: add a new `system.echo` request that returns `{ ok: true, text }`.

1. **Schema (source of truth)**

Add to `src/gateway/protocol/schema.ts`:

```
export const SystemEchoParamsSchema = Type.Object(
  { text: NonEmptyString },
  { additionalProperties: false },
);

export const SystemEchoResultSchema = Type.Object(
  { ok: Type.Boolean(), text: NonEmptyString },
  { additionalProperties: false },
);
```

Add both to `ProtocolSchemas` and export types:

```
  SystemEchoParams: SystemEchoParamsSchema,
  SystemEchoResult: SystemEchoResultSchema,
```

```
export type SystemEchoParams = Static<typeof SystemEchoParamsSchema>;
export type SystemEchoResult = Static<typeof SystemEchoResultSchema>;
```

2. **Validation**

In `src/gateway/protocol/index.ts`, export an AJV validator:

```
export const validateSystemEchoParams = ajv.compile<SystemEchoParams>(
```

3. **Server behavior**

Add a handler in `src/gateway/server-methods/system.ts`:

```
export const systemHandlers: GatewayRequestHandlers = {
  "system.echo": ({ params, respond }) => {
    const text = String(params.text ?? "");
    respond(true, { ok: true, text });
  },
};
```

Register it in `src/gateway/server-methods.ts` (already merges `systemHandlers`), then add `"system.echo"` to `METHODS` in `src/gateway/server.ts`.

4. **Regenerate**

```
pnpm protocol:check
```

5. **Tests + docs**

Add a server test in `src/gateway/server.*.test.ts` and note the method in docs.

## Swift codegen behavior

The Swift generator emits:

- `GatewayFrame` enum with `req`, `res`, `event`, and `unknown` cases
- Strongly typed payload structs/enums
- `ErrorCode` values and `GATEWAY_PROTOCOL_VERSION`

Unknown frame types are preserved as raw payloads for forward compatibility.

## Versioning + compatibility

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

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

The Swift models keep unknown frame types to avoid breaking older clients.

## Schema patterns and conventions

Most objects use `additionalProperties: false` for strict payloads.

`NonEmptyString` is the default for IDs and method/event names.

The top-level `GatewayFrame` uses a **`discriminator`** on `type`.

Methods with side effects usually require an `idempotencyKey` in params (example: `send`, `poll`, `agent`, `chat.send`).

## Live schema JSON

Generated JSON Schema is in the repo at `dist/protocol.schema.json`. The published raw file is typically available at:

https://raw.githubusercontent.com/openclaw/openclaw/main/dist/protocol.schema.json

## When you change schemas

1. Update the TypeBox schemas.
2. Run `pnpm protocol:check`.
3. Commit the regenerated schema + Swift models.

‹ grammY

Markdown Formatting ›