OpenAI Platform

Webhooks and server-side controls

Use webhooks and server-side controls with the Realtime API.

Copy page

The Realtime API allows clients to connect directly to the API server via WebRTC or SIP. However, you'll most likely want tool use and other business logic to reside on your application server to keep this logic private and client-agnostic.

Keep tool use, business logic, and other details secure on the server side by connecting over a “sideband” control channel. We now have sideband options for both SIP and WebRTC connections.

With WebRTC

When establishing a peer connection, you receive an SDP response from the Realtime API to configure the connection. If you used the sample code from the WebRTC guide, that looks something like this:

```
const baseUrl = "https://api.openai.com
const model = "gpt-realtime";
const sdpResponse = await fetch(`${base
    method: "POST",
    body: offer.sdp,
    headers: {
        Authorization: `Bearer ${EPHEME
        "Content-Type": "application/sd
    },
});
```

The SDP response will contain a `Location` header that has a unique call ID that can be used on the server to establish a WebSocket connection to that same Realtime session.

```
const location = sdpResponse.headers.get
const callId = location?.split("/").pop(
console.log(callId);
// rtc_u1_9c6574da8b8a41a18da9308f4ad974
```

On a server, you can then listen for events and configure the session just as you would from the client, using the ID from this URL:

```
import WebSocket from "ws";

// You'll need to get the call ID from
// server somehow:
const callId = "rtc_u1_9c6574da8b8a41a1

// Connect to a WebSocket for the in-pr
const url = "wss://api.openai.com/v1/re
const ws = new WebSocket(url, {
    headers: {
        Authorization: "Bearer " + proc
    },
});

ws.on("open", function open() {
    console.log("Connected to server.")

    // Send client events over the WebS
    ws.send(
        JSON.stringify({
            type: "session.update",
            session: {
                type: "realtime",
                instructions: "Be extra
            },
        })
    );
});

// Listen for and parse server events
ws.on("message", function incoming(mess
    console.log(JSON.parse(message.toSt
});
```

In this way, you are able to add tools, monitor sessions, and carry out business logic on the server instead of needing to configure those actions on the client.

With SIP

1. A user connects to OpenAI via phone over SIP.
2. OpenAI sends a webhook to your application's backend webhook URL, notifying your app of the state of the session.

```
POST https://my_website.com/webhook_enc
user-agent: OpenAI/1.0 (+https://platfo
content-type: application/json
webhook-id: wh_685342e6c53c8190a1be43f0
webhook-timestamp: 1750287078 # timesta
webhook-signature: v1,K5oZfzN95Z9UVu1Es

{
"object": "event",
"id": "evt_685343a1381c819085d44c354e1b
"type": "realtime.call.incoming",
"created_at": 1750287018, // Unix times
"data": {
"call_id": "some_unique_id",
"sip_headers": [
{ "name": "From", "value": "sip:+142555
{ "name": "To", "value": "sip:+18005551
{ "name": "Call-ID", "value": "03782086
],
}
}
```

1. The application server opens a WebSocket connection to the Realtime API using the `call_id` value provided in the webhook. This `call_id` looks like this: `wss://api.openai.com/v1/realtime?call_id={callId}`. The WebSocket connection will live for the life of the SIP call.