

Outgoing Call Service API Documentation

version 1.0.4 API REST + WebSocket

A comprehensive solution for managing voice calls through REST endpoints and real-time WebSocket communication.

Quick Start • □ Documentation • % Examples • ○ Support

Table of Contents

- **&** Overview
- 🛱 Authentication
- REST API Endpoints
- <u>A</u> Error Handling
- Examples
- Past Practices

© Overview

The Outgoing Call API provides a comprehensive solution for managing voice calls through REST endpoints and real-time WebSocket communication. This service enables applications to initiate outbound calls, manage call sessions, and handle bi-directional audio streaming.

★ Key Features

Feature	Description
Outbound Calls	Initiate calls to any phone number
Real-time Status	Live call status updates via WebSocket
■ Audio Streaming	Bi-directional audio with multiple format support
& Session Management	Complete call lifecycle management
∠ Call Detail Records	Comprehensive CDR with filtering and pagination

Base URL

https://api-call.optimaccs.com

份 Authentication

All API requests require a Bearer token for authentication.



Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Parameter	Type	Required	Description
Authorization	String		Bearer token format

⚠ **Important**: Keep your authentication token secure and never expose it in client-side code.

REST API Endpoints

1. Initiate Outgoing Call

Initiates a new outbound call to the specified phone number.

Endpoint: POST /pbx/call

Request Parameters

Parameter	Type	Required	Description
phoneNumber	String	Yes	Target phone number to call

Request Example

```
{
    "phoneNumber": "089608675796"
}
```

Response Parameters

Parameter	Type	Description
status	Number	HTTP status code (200: success, 500: failed)
errors	String	Error message (empty if successful)
data	Object	Response data containing call details
data.phoneNumber	String	The dialed phone number
data.sessionId	String	Unique session identifier

Response Example

```
{
   "status": 200,
   "errors": "",
   "data": {
      "phoneNumber": "089608675796",
      "sessionId": "bb4ad603-d3d9-4729-929c-7497a617cb26"
   }
}
```

2. Terminate Call

Terminates an active call session.

Endpoint: POST /pbx/hangup

Request Parameters

Parameter	Type	Required	Description
sessionId	String	Yes	Session ID from the outgoing call response

Request Example

```
{
    "sessionId": "bb4ad603-d3d9-4729-929c-7497a617cb26"
}
```

Response Parameters

Parameter	Type	Description
status	Number	HTTP status code (200: success, 500: failed)
errors	String	Error message (empty if successful)
data	Object	Response data
data.sessionId	String	Terminated session identifier

Response Example

```
{
    "status": 200,
    "errors": "",
    "data": {
        "sessionId": "bb4ad603-d3d9-4729-929c-7497a617cb26"
```

```
}
}
```

3. API: Retrieve Call Detail Records (CDR)

Endpoint: GET /v1/cdr

Fetches a list of Call Detail Records (CDRs) with support for pagination and filtering.

Query Parameters

Parameter	Туре	Description
page	number	Page number, starting from 1.
limit	number	Number of records per page.
filter. <field></field>	string	Used to filter records by field using supported operators.

***** Supported Filters

▶ ♥ Destination Filters

• Operators: \$eq, \$contains

• Examples:

```
/v1/cdr?filter.destination=$eq:08123456789
/v1/cdr?filter.destination=$contains:0812
```

▶ [®] Duration Filters

- Operators: \$eq, \$gt, \$gte, \$1t, \$1te
- Examples:

```
/v1/cdr?filter.duration=$gte:60
/v1/cdr?filter.duration=$lt:300
```

▶ ③ Billable Seconds Filters

• Operators: \$eq, \$gt, \$gte, \$1t, \$1te

• Examples:

```
/v1/cdr?filter.billableSeconds=$gt:30
/v1/cdr?filter.billableSeconds=$lte:180
```

▶ ■ Disposition Filters

• Operators: \$eq, \$in

• Examples:

```
/v1/cdr?filter.disposition=$eq:ANSWERED
/v1/cdr?filter.disposition=$in:ANSWERED,NO ANSWER,FAILED
```

Example Request

```
GET /v1/cdr?
page=1&limit=10&filter.destination=$contains:0812&filter.duration=$gte:30
```

Response Example

```
"data": [
     "sessionId": "0198355b-f324-718b-95a4-625c43c654bc",
      "destination": "0811234678",
      "startTime": "2025-07-23T03:37:56.000Z",
      "answerTime": null,
      "endTime": "2025-07-23T03:38:20.000Z",
      "duration": 24,
      "billableSeconds": 0,
      "disposition": "NO ANSWER",
      "hangupBy": "CALLEE",
      "hangupCauseCode": 18,
      "hangupCauseText": "No response from destination",
      "createdAt": "2025-07-23T03:38:22.000Z",
      "updatedAt": "2025-07-23T03:38:22.000Z"
 },
],
  "meta": {
        "itemsPerPage": 100,
        "totalItems": 36,
        "currentPage": 1,
        "totalPages": 1,
        "sortBy": [
                "id",
                "DESC"
            ]
        ]
```

```
},
```

4. API: Get CDR Detail

Endpoint: GET /v1/cdr/{sessionId}

Retrieves detailed information for a specific Call Detail Record (CDR) by session ID.

Path Parameters

Parameter Type Description

sessionId string Unique session identifier for the CDR.

Example Request

```
GET /v1/cdr/0198355b-f324-718b-95a4-625c43c654bc
```

Example Response

```
"status": 200,
  "errors": "",
  "data": {
    "sessionId": "0198355b-f324-718b-95a4-625c43c654bc",
    "destination": "089608675796",
    "startTime": "2025-07-23T03:37:56.000Z",
    "answerTime": null,
    "endTime": "2025-07-23T03:38:20.000Z",
    "duration": 24,
    "billableSeconds": ∅,
    "disposition": "NO ANSWER",
    "hangupBy": "CALLEE",
    "hangupCauseCode": 18,
    "hangupCauseText": "No response from destination",
    "createdAt": "2025-07-23T03:38:22.000Z",
    "updatedAt": "2025-07-23T03:38:22.000Z"
 }
}
```

Response Fields

Field Type Description

Field	Type	Description
sessionId	string	Unique ID for the CDR session.
destination	string	Destination phone number.
startTime	datetime	Time the call was initiated.
answerTime	datetime	Time the call was answered (null if unanswered).
endTime	datetime	Time the call ended.
duration	number	Duration of the call in seconds.
billableSeconds	number	Billable time in seconds.
disposition	string	Final state of the call (e.g., ANSWERED, NO ANSWER).
hangupBy	string	Party that hung up (if available).
hangupCauseCode	number	Numeric hangup cause code.
hangupCauseText	string	Text explanation of hangup cause.
createdAt	datetime	Record creation time.
updatedAt	datetime	Last update time for the record.

Real-time communication using Socket.IO for live call management and audio streaming.

Connection Setup

```
const SOCKET_CONFIG = {
  transports: ["websocket"],
  reconnection: true,
  reconnectionAttempts: 5,
  reconnectionDelay: 1000,
  auth: {
    token: "your_bearer_token",
  },
};

const SERVER_URL = "https://api-call.optimaccs.com";
const socket = io(SERVER_URL, SOCKET_CONFIG);
```

■ Event Flow Diagram

```
sequenceDiagram

participant  Client

participant Server
```

```
Note over 🗏 Client, 🖳 Server: 🖉 Call Initialization
■ Client->>
■ Server:  POST /pbx/call
🖳 Server-->> 🖩 Client: 🔑 sessionId
Note over 🗏 Client, 🖳 Server: 🔗 Session Setup

■ Server-->> 
■ Client: 

↑ newSession
■ Client->>
■ Server:

□ joinRoom

■ Server-->> ■ Client: ✓ roomJoined

Note over 🗏 Client, 🗏 Server: 📞 Call Flow
🖳 Server-->> 🗏 Client: 📊 dialStatus

■ Client->>
■ Server: 

■ audio
🖳 Server-->> 🖩 Client: 🎜 audio
■ Client->>
■ Server: 
✓ checkPoint
Server-->> ■ Client: ✓ checkPoint
■ Client->> ■ Server: 3 dtmf
🖳 Server-->> 🖩 Client: 🔢 dtmf
Note over 🗏 Client, 🗏 Server: 🖀 Call Termination

■ Server-->> ■ Client: 
   hangup

🖳 Server-->> 🖩 Client: 🗐 cdr
```

ぬ WebSocket Events Reference

Server to Client Events

newSession

The newSession event is triggered when a new call session is initiated through the REST API. This allows clients to listen for the creation of both inbound and outbound call sessions in real-time.

```
socket.on("newSession", (data) => {
    /*
    data: {
        sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
        type: 'outbound' // 'outbound' or 'inbound'
    }
    */
});
```

Payload Parameters

Field	Type	Description
sessionId	String	Unique identifier for the new call session.
type	String	Type of call: "inbound" or "outbound".



The roomJoined event is emitted by the server to confirm whether the client has successfully joined the designated call room.

```
socket.on("roomJoined", (data) => {
    /*
    data: {
       sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
       status: 'success' // 'success' or 'failed'
    }
    */
});
```

Payload Parameters

Field	Type	Description
sessionId	String	The unique identifier of the call session (room).
status	String	Indicates if the client successfully joined the room.

dialStatus

The dialStatus event provides real-time updates about the current status of an ongoing call session. This helps the client track the call lifecycle (e.g., dialing, ringing, connected).

```
socket.on("dialStatus", (data) => {
    /*
    data: {
        sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
        status: 'Dialing' // 'Dialing', 'Ringing', 'Busy', 'Connected', 'Failed'
    }
    */
});
```

Payload Parameters

Field	Type	Description
sessionId	String	The unique identifier of the call session.
status	String	The current status of the call.

Status Values:

- END Dialing Call is being initiated
- **L** Ringing Destination is ringing

- **Susy** Destination is busy
- Connected Call successfully answered
- X Failed Call failed

12 dtmf

The dtmf event is triggered whenever a Dual-Tone Multi-Frequency (DTMF) digit is received during a call session. This is typically used for IVR (Interactive Voice Response) systems or capturing keypad input from the user.

```
socket.on("dtmf", (data) => {
    /*
    data: {
       sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
       digit: '1' // 1, 2, 3, 4 5, a, b etc
    }
    */
});
```

Payload Parameters

Field	Type	Description
sessionId	String	Unique identifier for the call session.
digit	String	The DTMF digit received (e.g., '1', '*', '#', 'A').

I audio

The audio event is triggered to deliver raw audio data from the ongoing call session. This audio stream can be used for real-time processing, such as transcription, speech analysis, or playback.

```
socket.on("audio", (data) => {
    /*
    data: {
      sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
      audioData: <Buffer 33 29 12 94> // Buffer type
      audioFormat: 'pcm16'
    }
    */
});
```

Audio Specifications:

• **G** Format: Linear PCM 16-bit

• 📊 Sample Rate: 8 kHz

• **Channels:** Mono

Payload Parameters

Field	Туре	Type Description	
sessionId	String	Unique identifier for the call session.	
audioData	Buffer	Audio buffer in 16-bit linear PCM format.	
audioFormat	String	type of audio pcm16	

✓ checkPoint

The checkPoint event notifies the client when a previously submitted checkpoint has finished playing its associated audio. This can be used to synchronize events or trigger actions precisely after specific audio segments have completed.

```
socket.on("checkPoint", (data) => {
    /*
    data: {
        sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
        name: your check point name
    }
    */
});
```

Payload Parameters

Field	Type	e Description	
sessionId	String	Unique identifier for the call session.	
name	String	The name of the checkpoint that has completed.	

hangup

The hangup event notifies the client that the call session has been terminated. This may occur due to user action, network issues, or a remote party disconnecting.

```
socket.on("hangup", (data) => {
    /*
    data: {
      sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f'
    }
    */
});
```

Payload Parameters

Field	Type	Description
sessionId	String	Unique identifier for the call session that has ended.



The cdr event is emitted after a call session has ended, typically upon call hangup. This event provides a Call Detail Record (CDR), which contains comprehensive metadata related to the completed call session. This event allows the system or client application to log, store, or process call-related data for billing, analytics, auditing, or reporting purposes.

```
socket.on("cdr", (cdr) => {
 /*
 cdr: {
   sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
   destination: '0811234567',
   startTime: 2025-07-23T05:17:28.000Z,
    answerTime: 2025-07-23T05:17:34.000Z,
    endTime: 2025-07-23T05:17:40.000Z,
    duration: 11,
    billableSeconds: 5,
    disposition: 'ANSWERED',
    hangupBy: 'CALLEE',
   hangupCauseCode: 16,
   hangupCauseText: 'Call completed successfully'
 }
});
```

Cdr field

Field	Description
sessionId	Unique identifier for the call session
destination	Destination number or party being called
startTime	Timestamp when the call was initiated
answerTime	Timestamp when the call was answered
endTime	Timestamp when the call ended (hangup occurred)
duration	Total call duration in seconds (from start to end)
billableSeconds	Duration considered billable, excluding ring time or early termination
disposition	Final status of the call (e.g., ANSWERED, NO ANSWER, BUSY, FAILED, CANCEL)
hangupBy	Indicates which party ended the call (CALLER or CALLEE)

Field	Description
hangupCauseCode	Numeric code indicating the reason for call termination
hangupCauseText	Textual description of the hangup cause

↓ Hangup Cause Codes

Code	Text	Description
0	? Call terminated - unknown reason	Cause is unknown or not configured
1	X Invalid destination number	Called number is not allocated/invalid
16	✓ Call completed successfully	Call terminated normally (usually by remote party)
17	Destination busy	User is busy
18	No response from destination	No response from user (not answering)
19	L Destination not answering	Phone is ringing but not answered
21	Call declined by user	Call rejected by recipient
27	Service unavailable	Destination number is not working or inactive
28		Wrong or incomplete number format
29	X Service not supported	Service or feature request rejected
34	Network congestion	No circuit/channel available to make the call
38	% Network failure	Network issues causing call failure
44	f Resource unavailable	Requested circuit/channel is not available
58	⚠ Service not implemented	Channel type not supported by recipient
102		Timeout while waiting for response, system performs recovery
127	❖ System error	Inter-network issues without specific explanation

Client to Server Events



Request to join a call session room.

```
socket.emit("joinRoom", {
   sessionId: "07475d2c-32c9-4f4b-8103-6db8c0dc741f",
});
```

J audio

The audio event allows the client to send raw or encoded audio data directly into an ongoing call session in real time. This enables advanced media control such as custom voice prompts, TTS injection, and audio overlays.

This interface supports multiple audio formats commonly used in telephony systems and requires precise timing and byte alignment for optimal performance.

```
socket.emit('audio', {
   audioData: <Buffer 33 29 12 94>,
   sessionId: '07475d2c-32c9-4f4b-8103-6db8c0dc741f',
   audioFormat: 'mp3', // 'mp3', 'ulaw', 'alaw', 'pcm16'
}, (acknowledgment) => {
   /*
   acknowledgment: {
     success: true,
     error: null
   }
   */
});
```

Payload Parameters:

Field	Туре	Description
audioData	Buffer	The binary audio payload. Must conform to the format and timing expectations.
sessionId	string	The unique identifier of the call session currently in progress.
audioFormat	string	Format of the audio data. Accepted values: 'mp3', 'ulaw', 'alaw', 'pcm16'.

Audio Timing and Chunk Size

To ensure smooth playback, audio data must be chunked in real-time intervals. A standard telephony sampling rate of 8000 Hz is used, where each 320 bytes of pcm16 audio represents 20 ms of sound.

8 For uncompressed pcm16, chunk size = 2 bytes per sample \times 8000 samples/sec \times duration.

Audio Packet Timing Table

This table shows how chunk sizes affect transmission timing and efficiency over the network:

% 8000 Hz, 320 bytes per packet (20 ms audio)

Bytes	Packets	Duration (ms)	Playback Timing	Network Efficiency
1920	6×	120 ms	120 ms	Good efficiency

Bytes	Packets	Duration (ms)	Playback Timing	Network Efficiency
2240	7×	140 ms	140 ms	Good efficiency
2560	8×	160 ms	160 ms	★ OPTIMAL
2880	9×	180 ms	180 ms	High efficiency

Recommendation: 2560-byte packets (160 ms) provide the best in condition network latency range 0 - 1ms trade-off between latency and transmission overhead in most VoIP or streaming scenarios.

Best Practice

Packet size optimal: 2560 bytes (160 ms)

i Important Note:

The chunk duration of 160 ms refers to the amount of audio contained within each packet, not the total latency experienced by the user. The actual total latency is calculated as: Total Latency = Chunk Duration + Network Latency (client ↔ server)

⚠ If the network latency is 50 ms, (160ms - 50ms) the maximum timing duration should be approximately 100 ms.

✓ Acknowledgment Response:

Field	Туре	Description
success	boolean	Indicates whether the audio chunk was successfully processed.
error	string or null	Returns error message if transmission failed.

✓ checkPoint

After sending audio data to the call session using the audio event, the client may emit a checkPoint event to indicate a significant moment during audio playback—such as a marker, timestamp, or synchronization point.

This can be useful for logging, analytics, or coordinating client-side logic in real-time.

```
socket.emit("checkPoint", {
  sessionId: "07475d2c-32c9-4f4b-8103-6db8c0dc741f",
  name: "mark audio playback",
});
```

Parameters

Field	Type	Description
sessionId	string	The unique identifier for the call session.

Field	Type	Description
name	string	A descriptive label or name for the checkpoint event.

Behavior

Upon receiving the checkPoint event, the server may:

- Record the current playback timestamp.
- Log the event for audit or debug purposes.
- Acknowledge the checkpoint to the sender (if implemented).

Best Practice

It is recommended to emit checkPoint immediately after the final or key audio emit to ensure temporal alignment with audio playback on the server side. For example, use it to denote:

- Start or end of an audio segment.
- Specific spoken phrase boundaries.
- Transcription markers or cue points.

12 dtmf

The dtmf event is used to send Dual-Tone Multi-Frequency (DTMF) signals into an active audio session. This is typically used for transmitting keypresses (such as digits, *, or #) during a phone call, IVR navigation, or interactive voice response systems.

```
socket.emit("dtmf", {
    sessionId: "07475d2c-32c9-4f4b-8103-6db8c0dc741f",
    digit: "1",
    duration: 200, // in milisecond
});
```

Parameters

• sessionId (string)

The unique identifier for the active audio session.

• digit (string)

The DTMF digit to be sent. Acceptable values include '0'-'9', '*', and '#'.

• duration (number)

The length of the tone in milliseconds (commonly between 100-500ms).

Behavior

Upon receiving the dtmf event, the server:

- Injects the specified DTMF tone into the media stream for the given session.
- May log or audit the DTMF tone for traceability or feature triggers.

Best Practice

- Keep DTMF tone durations within a standard range (e.g., 150–300ms) to ensure compatibility across different endpoints and devices.
- Avoid sending multiple DTMF digits in rapid succession without appropriate inter-digit delay (~50ms).

interruption

The interruption event is used to immediately stop any audio that is currently being played and clear the audio queue for a given session.

This is typically used when you need to cancel ongoing audio playback—for example, when the user takes an action that invalidates the current audio flow or when a new message needs to take priority.

```
socket.emit("interruption", {
    sessionId: "07475d2c-32c9-4f4b-8103-6db8c0dc741f",
});
```

Parameters

Field	Type	Required	Description
sessionId	String	Yes	The unique session ID of the active connection whose audio should be stopped.

Behavior

- Current audio playback is stopped immediately.
- All queued audio files are discarded.
- No new audio will be played unless a new playAudio or similar event is emitted.

hangup

The hangup event is used to request termination of an active call session. Once emitted, the server will process the request and initiate the call hangup procedure associated with the specified session.

```
socket.emit("hangup", {
   sessionId: "07475d2c-32c9-4f4b-8103-6db8c0dc741f",
});
```

Parameters

Field	Type	Required	Description
sessionId	String	Yes	The unique identifier of the call session to be terminated.

Behavior

- Ends the call session associated with the given sessionId.
- Triggers internal cleanup such as stopping media, closing connections, and generating call records (CDRs).
- May notify other connected clients or systems that the session has ended.

♠ Error Handling

HTTP Status Codes

Code	Status	Description
☑ 200	Success	Request completed successfully
X 400	Bad Request	Invalid parameters
1 401	Unauthorized	Invalid or missing token
Q 404	Not Found	Resource not found
≯ 500	Server Error	Internal server error

Common Error Responses

```
{
   "status": 400,
   "errors": "Invalid phone number format",
   "data": null
}
```

```
{
   "status": 401,
   "errors": "Authentication token is required",
   "data": null
}
```

Code Examples

Quick Start Example

```
// 1 Initialize connection
const socket = io("https://api-call.optimaccs.com", {
 transports: ["websocket"],
 auth: { token: "your_bearer_token" },
});
// 2 Set up event listeners
socket.on("newSession", (data) => {
 console.log("NEW New session:", data.sessionId);
  socket.emit("joinRoom", { sessionId: data.sessionId });
});
socket.on("dialStatus", (data) => {
 console.log(" \ Status:", data.status);
 if (data.status === "Connected") {
   // I Send audio when connected
   sendAudio(data.sessionId);
});
// [3] Initiate call
async function makeCall(phoneNumber) {
 const response = await fetch("https://api-call.optimaccs.com/pbx/call", {
   method: "POST",
   headers: {
      "Content-Type": "application/json",
     Authorization: "Bearer your_token",
   },
   body: JSON.stringify({ phoneNumber }),
 });
 const result = await response.json();
 return result.data.sessionId;
}
// 🔊 Usage
makeCall("089608675796");
```

Audio Streaming Example

```
function sendAudioStream(sessionId, audioFile) {
  const chunkSize = 2560; // Optimal chunk size for 160ms
  const reader = new FileReader();

  reader.onload = function (e) {
    const buffer = new Uint8Array(e.target.result);

  for (let i = 0; i < buffer.length; i += chunkSize) {
    const chunk = buffer.slice(i, i + chunkSize);
}</pre>
```

```
socket.emit(
        "audio",
          audioData: chunk,
          sessionId: sessionId,
          audioFormat: "mp3",
        },
        (ack) \Rightarrow {
          if (!ack.success) {
            console.error("X Audio send failed:", ack.error);
          }
        }
      );
      // Add proper timing delay
      setTimeout(() => {}, 160); // 160ms for optimal chunks
   }
 };
 reader.readAsArrayBuffer(audioFile);
}
```



```
async function getCDRWithFilters() {
 const filters = new URLSearchParams({
    page: 1,
   limit: 50,
    "filter.destination": "$contains:0812",
    "filter.duration": "$gte:30",
    "filter.billableSeconds": "$gt:0",
 });
 const response = await fetch(
    `https://api-call.optimaccs.com/v1/cdr?${filters}`,
   {
      headers: {
        Authorization: "Bearer your_token",
      },
   }
 );
 const data = await response.json();
 return data;
```

Best Practices

♠ Security

- Store authentication tokens securely
- Use HTTPS for all API requests
- X Never expose tokens in client-side code

Connection Management

- Implement reconnection logic for WebSocket
- Mandle connection timeouts gracefully
- Monitor connection health
- Disconnect socket.io with timeout after hangup or cdr event

Audio Handling

- Use optimal chunk sizes (2560 bytes recommended)
- Implement proper buffering
- Handle audio format conversions
- Add error handling for audio transmission

Session Management

- Track active sessions
- Implement proper error recovery

Support

♦ Need Help?

Resource	Link
™ Email Support	noc@atlasat.co.id
Documentation	Version 1.0.4
⊕ Base URL	https://api-call.optimaccs.com
℘ Status Page	[Check API Status]

Made with **♥** for developers