# Amazon Chime

## Developer Guide

# Amazon Chime: Developer Guide

# Table of Contents

# What is Amazon Chime?

Amazon Chime is a communications service that transforms online meetings with an application that is secure and comprehensive. Amazon Chime works across your devices so that you can stay connected. You can use Amazon Chime for online meetings, video conferencing, calls, and chat. You can also share content inside and outside of your organization. Amazon Chime helps you to work productively from anywhere.

Developers can use Amazon Chime with the AWS SDK or AWS Command Line Interface (AWS CLI) to perform such tasks as managing users, or integrating webhooks and chat bots with Amazon Chime. They can also use the Amazon Chime SDK to build real-time media applications that can send and receive audio and video and allow content sharing. For detailed information about Amazon Chime API actions, see the *Amazon Chime API Reference*.

Amazon Chime permissions are controlled using AWS Identity and Access Management (IAM). For more information, see Identity and access management for Amazon Chime in the *Amazon Chime Administrator Guide*.

## Pricing

Amazon Chime provides usage-based pricing. You pay only for the users with Pro permissions that host meetings, and only on the days that those meetings are hosted. For more information, see Amazon Chime pricing.

## Resources

The following related resources can help you as you work with this service.

- **Classes & Workshops** – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- **AWS Developer Tools** – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- **AWS Whitepapers** – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- **AWS Support Center** – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- **AWS Support** – The primary webpage for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- **Contact Us** – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- **AWS Site Terms** – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

# Extending the Amazon Chime desktop client

Developers can extend the capabilities of the Amazon Chime desktop client by adding chat bots, proxy phone sessions, and webhooks. Chat bots enable users to perform tasks such as querying internal systems for information. Proxy phone sessions allow users to call and send texts without revealing their phone numbers. Webhooks can automatically send messages to chat rooms. For example, a webhook can send meeting reminders to a team, along with a link to the meeting.

**Topics**



# User management

The following code snippets can help developers manage Amazon Chime users. All of the examples in this topic use Java.

**Content**



## Invite multiple users

The following example shows how to invite multiple users to an Amazon Chime `Team` account.

```
List<String> emails = new ArrayList<>();
emails.add("janedoe@example.com");
emails.add("richardroe@example.net");
InviteUsersRequest inviteUsersRequest = new InviteUsersRequest()
    .withAccountId("chimeAccountId")
    .withUserEmailList(emails);

chime.inviteUsers(inviteUsersRequest);
```

## Download user list

The following example shows how to download a list of users associated with your Amazon Chime administrative account in `.csv` format.

```
BufferedWriter writer = Files.newBufferedWriter(Paths.get("/path/to/csv"));
CSVPrinter printer = new CSVPrinter(writer, CSVFormat.DEFAULT.withHeader("userId",
 "email"));

ListUsersRequest listUsersRequest = new ListUsersRequest()
```

```
    .withAccountId(accountId)
    .withMaxResults(1);

boolean done = false;
while (!done) {
    ListUsersResult listUsersResult = chime.listUsers(listUsersRequest);
    for (User user: listUsersResult.getUsers()) {
        printer.printRecord(user.getUserId(), user.getPrimaryEmail());
    }

    if (listUsersResult.getNextToken() == null) {
        done = true;
    }

    listUsersRequest = new ListUsersRequest()
        .withAccountId(accountId)
        .withNextToken(listUsersResult.getNextToken());
}

printer.close();
```

## Log out multiple users

The following example shows how to log out multiple users from your Amazon Chime administrative account.

```
ListUsersRequest listUsersRequest = new ListUsersRequest()
    .withAccountId("chimeAccountId");
ListUsersResult listUsersResult = chime.listUsers(listUsersRequest);

for (User user: listUsersResult.getUsers()) {
    LogoutUserRequest logoutUserRequest = new LogoutUserRequest()
        .withAccountId(user.getAccountId())
        .withUserId(user.getUserId());

    chime.logoutUser(logoutUserRequest);
}
```

## Update user personal PINs

The following example shows how to reset the personal meeting PIN for a specified Amazon Chime user.

```
ResetPersonalPINRequest request = new ResetPersonalPINRequest()
    .withAccountId("chimeAccountId")
    .withUserId("userId");

ResetPersonalPINResult result = chime.resetPersonalPIN(request);

User user = result.getUser();
user.getPersonalPIN()
```

# Integrating chatbots into the Amazon Chime desktop client

Developers can use the AWS Command Line Interface (AWS CLI), Amazon Chime API, or AWS SDK to integrate chatbots with Amazon Chime. Chatbots let you use the power of Amazon Lex, AWS Lambda,

and other AWS services to streamline common tasks with intelligent conversational interfaces that are accessible to users in Amazon Chime chat rooms.

If you're an Amazon Chime Enterprise account administrator, you can use chatbots to allow users to perform such tasks as:

- Querying their internal systems for information.
- Automating tasks.
- Receiving notifications for critical issues.
- Creating support tickets.

For more information about Amazon Chime Enterprise accounts, see Managing your Amazon Chime accounts in the *Amazon Chime Administrator Guide*.

If you administer an Amazon Chime Enterprise account, you can create up to 10 chatbots for integration with Amazon Chime. Chatbots can be used only in chat rooms created by members of your account. Only chat room administrators can add chatbots to a chat room. After a chatbot is added to a chat room, members of the chat room can interact with the bot using commands provided by the bot creator. For more information, see Using chatbots in the *Amazon Chime User Guide*.

Linux and macOS users can build a sample custom chatbot. For more information, see Build custom chatbots for Amazon Chime.

**Content**



# Use chatbots with Amazon Chime

If you administer an Amazon Chime Enterprise account, you can create up to 10 chatbots for integration with Amazon Chime. Chatbots can only be used in chat rooms created by members of your account. Only chat room administrators can add chatbots to a chat room. After a chatbot is added to a chat room, members of the chat room can interact with the bot using commands provided by the bot creator. For more information, see Using chatbots in the *Amazon Chime User Guide*.

You can also use the Amazon Chime API operation to enable or stop chatbots for your Amazon Chime account. For more information, see Update chatbots (p. 10).

> **Note**
> Chatbots cannot be deleted. To stop a chatbot from being used in your account, use the Amazon Chime UpdateBot API operation in the *Amazon Chime API Reference*. When you stop a chatbot, chat room administrators can remove it from a chat room, but they cannot add it to a chat room. Users who @mention a stopped chatbot in a chat room receive an error message.

## Prerequisites

Before you start the procedure to integrate chatbots with Amazon Chime, complete the following prerequisites:

- Create a chatbot.
- Create the outbound endpoint for Amazon Chime to send events to your bot. Choose from an AWS Lambda function ARN or an HTTPS endpoint. For more information about Lambda, see the *AWS Lambda Developer Guide*.

## DNS best practices for HTTPS endpoints

We recommend the following best practices when assigning DNS for your HTTPS endpoint:

- Use a DNS subdomain that is dedicated to the bot endpoint.
- Use only A-records to point to the bot endpoint.
- Protect your DNS servers and DNS registrar account to prevent domain hijacking.
- Use publicly valid TLS intermediate certificates that are dedicated to the bot endpoint.
- Cryptographically verify the bot message signature before acting on a bot message.

After creating your chatbot, use the AWS Command Line Interface (AWS CLI) or the Amazon Chime API operation to complete the tasks described in the following sections.

**Tasks**



# Step 1: Integrate a chatbot with Amazon Chime

After you complete the prerequisites (p. 4), integrate your chatbot with Amazon Chime using the AWS CLI or Amazon Chime API.

> **Note**
> These procedures create a name and email address for your chatbot. Chatbot names and email addresses cannot be changed after creation.

## AWS CLI

### To integrate a chatbot using the AWS CLI

1. To integrate your chatbot with Amazon Chime, use the **create-bot** command in the AWS CLI.

```
aws chime create-bot --account-id 12a3456b-7c89-012d-3456-78901e23fg45 --display-
name exampleBot --domain example.com
```

   a. Enter a chatbot display name of up to 55 alphanumeric or special characters (such as +, -, %).
   b. Enter the registered domain name for your Amazon Chime Enterprise account.

2. Amazon Chime returns a response that includes the bot ID.

```
"Bot": {
        "CreatedTimestamp": "timeStamp",
        "DisplayName": "exampleBot",
        "Disabled": exampleBotFlag,
        "UserId": "1ab2345c-67de-8901-f23g-45h678901j2k",
        "BotId": "botId",
        "UpdatedTimestamp": "timeStamp",
        "BotType": "ChatBot",
        "SecurityToken": "securityToken",
        "BotEmail": "displayName-chimebot@example.com"
```

```
        }
```

3. Copy and save the bot ID and bot email address to use in the following procedures.

### Amazon Chime API

**To integrate a chatbot using the Amazon Chime API**

1. To integrate your chatbot with Amazon Chime, use the CreateBot API operation in the *Amazon Chime API Reference*.
   a. Enter a chatbot display name of up to 55 alphanumeric or special characters (such as +, -, %).
   b. Enter the registered domain name for your Amazon Chime Enterprise account.
2. Amazon Chime returns a response that includes the bot ID. Copy and save the bot ID and email address. The bot email address looks like this: *exampleBot*-chimebot@*example.com*.

### AWS SDK for Java

The following sample code demonstrates how to integrate a chatbot using the AWS SDK for Java.

```
CreateBotRequest createBotRequest = new CreateBotRequest()
    .withAccountId("chimeAccountId")
    .withDisplayName("exampleBot")
    .withDomain("example.com");

chime.createBot(createBotRequest);
```

Amazon Chime returns a response that includes the bot ID. Copy and save the bot ID and email address. The bot email address looks like this: *exampleBot*-chimebot@*example.com*.

## Step 2: Configure the outbound endpoint for an Amazon Chime chatbot

After you create a chatbot ID for your Amazon Chime Enterprise account, configure your outbound endpoint for Amazon Chime to use to send messages to your bot. The outbound endpoint can be an AWS Lambda function ARN or an HTTPS endpoint that you created as part of the prerequisites (p. 4). For more information about Lambda, see the *AWS Lambda Developer Guide*.

> **Note**
> If the outbound HTTPS endpoint for your bot is not configured or is empty, chat room administrators cannot add the bot to a chat room. Also, chat room users cannot interact with the bot.

### AWS CLI

To configure an outbound endpoint for your chatbot, use the **put-events-configuration** command in the AWS CLI. Configure a Lambda function ARN or an outbound HTTPS endpoint.

Lambda ARN

```
aws chime put-events-configuration --account-id 12a3456b-7c89-012d-3456-78901e23fg45
 --bot-id botId --lambda-function-arn arn:aws:lambda:us-
east-1:111122223333:function:function-name
```

HTTPS endpoint

```
aws chime put-events-configuration --account-id 12a3456b-7c89-012d-3456-78901e23fg45 --
bot-id botId --outbound-events-https-endpoint https://example.com:8000
```

Amazon Chime responds with the bot ID and HTTPS endpoint.

```
{
    "EventsConfiguration": {
        "BotId": "BotId",
        "OutboundEventsHTTPSEndpoint": "https://example.com:8000"
    }
}
```

## Amazon Chime API

To configure the outbound endpoint for your chatbot, use the Amazon Chime PutEventsConfiguration API operation in the *Amazon Chime API Reference*. Configure either a Lambda function ARN or an outbound HTTPS endpoint.

- **If you configure a Lambda function ARN** – Amazon Chime calls Lambda to add permission to allow the Amazon Chime administrator's AWS account to invoke the provided Lambda function ARN. This is followed by a dry run invocation to verify that Amazon Chime has permission to invoke the function. If adding permissions fails, or if the dry run invocation fails, then the `PutEventsConfiguration` request returns an HTTP 4xx error.
- **If you configure an outbound HTTPS endpoint** – Amazon Chime verifies your endpoint by sending an HTTP Post request with a Challenge JSON payload to the outbound HTTPS endpoint that you provided in the previous step. Your outbound HTTPS endpoint must respond by echoing back the Challenge parameter in JSON format. The following examples show the request and a valid response.

  Request

  ```
  HTTPS POST

  JSON Payload:
  {
     "Challenge":"0000000000000000000",
     "EventType" : "HTTPSEndpointVerification"
  }
  ```

  Response

  ```
  HTTP/1.1 200 OK
  Content-type: application/json

  {
      "Challenge":"0000000000000000000"
  }
  ```

  If the challenge handshake fails, then the `PutEventsConfiguration` request returns an HTTP 4xx error.

### AWS SDK for Java

The following sample code demonstrates how to configure an endpoint using the AWS SDK for Java.

```
PutEventsConfigurationRequest putEventsConfigurationRequest = new
 PutEventsConfigurationRequest()
     .withAccountId("chimeAccountId")
     .withBotId("botId")
     .withOutboundEventsHTTPSEndpoint("https://www.example.com")
     .withLambdaFunctionArn("arn:aws:lambda:region:account-id:function:function-name");

 chime.putEventsConfiguration(putEventsConfigurationRequest):
```

## Step 3: Add the chatbot to an Amazon Chime chat room

Only a chat room administrator can add a chatbot to a chat room. They use the chatbot email address created in Step 1 (p. 5).

**To add a chatbot to a chat room**

1. Open the Amazon Chime desktop client or web application.
2. Choose the gear icon in the upper-right corner, and choose **Manage webhooks and bots**.
3. Choose **Add bot**.
4. For **Email address**, enter the bot email address.
5. Choose **Add**.

The bot name appears in the chat room roster. If there are additional actions necessary to add a chatbot to a chat room, provide the actions to the chat room administrator.

After the chatbot is added to the chat room, provide the chatbot commands to your chat room users. One way to do this is to program your chatbot to send command help to the chat room when it receives the chat room invite. AWS also recommends creating a help command for your chatbot users to use.

## Authenticate chatbot requests

You can authenticate requests sent to your chatbot from an Amazon Chime chat room. To do this, compute a signature based on the request. Then, validate that the computed signature matches the one on the request header. Amazon Chime uses the HMAC SHA256 hash to generate the signature.

If your chatbot is configured for Amazon Chime using an outbound HTTPS endpoint, use the following authentication steps.

**To validate a signed request from Amazon Chime for a chatbot with a outbound HTTPS endpoint configured**

1. Get the **Chime-Signature** header from the HTTP request.
2. Get the **Chime-Request-Timestamp** header and the **body** of the request. Then, use a vertical bar as the delimiter between the two elements to form a string.
3. Use the **SecurityToken** from the CreateBot response as the initial key of **HMAC_SHA_256**, and hash the string that you created in step 2.
4. Encode the hashed byte with Base64 encoder to a signature string.
5. Compare this computed signature to the one in the **Chime-Signature** header.

The following code sample demonstrates how to generate a signature using Java.

```
       private final String DELIMITER = "|";
       private final String HMAC_SHA_256 = "HmacSHA256";

       private String generateSignature(String securityToken, String requestTime, String
requestBody)
       {
           try {
               final Mac mac = Mac.getInstance(HMAC_SHA_256);
               SecretKeySpec key = new SecretKeySpec(securityToken.getBytes(UTF_8),
HMAC_SHA_256);
               mac.init(key);
               String data = requestTime + DELIMITER + requestBody;
               byte[] rawHmac = mac.doFinal(data.getBytes(UTF_8));

               return Base64.getEncoder().encodeToString(rawHmac);
               }
           catch (Exception e) {
               throw e;
               }
        }
```

The outbound HTTPS endpoint must respond to the Amazon Chime request with `200 OK` within 2 seconds. Otherwise, the request fails. If the outbound HTTPS endpoint is unavailable after 2 seconds, possibly because of a Connection or Read timeout, , or if Amazon Chime receives a 5xx response code, Amazon Chime retries the request two times. The first retry is sent 200 milliseconds after the initial request fails. The second retry is sent 400 milliseconds after the previous retry fails. If the outbound HTTPS endpoint is still unavailable after the second retry, the request fails.

**Note**
The **Chime-Request-Timestamp** changes each time the request is retried.

If your chatbot is configured for Amazon Chime using a Lambda function ARN, use the following authentication steps.

### To validate a signed request from Amazon Chime for a chatbot with a Lambda function ARN configured

1. Get the **Chime-Signature** and **Chime-Request-Timestamp** from the Lambda request **ClientContext**, in Base64 encoded JSON format.

   ```
   {
   "Chime-Signature" : "1234567890",
   "Chime-Request-Timestamp" : "2019-04-04T21:30:43.181Z"
   }
   ```

2. Get the **body** of the request from the request payload.
3. Use the **SecurityToken** from the `CreateBot` response as the initial key of **HMAC_SHA_256**, and hash the string that you created.
4. Encode the hashed byte with Base64 encoder to a signature string.
5. Compare this computed signature to the one in the **Chime-Signature** header.

If a `com.amazonaws.SdkClientException` occurs during the Lambda invocation, Amazon Chime retries the request two times.

## Update chatbots

As the Amazon Chime account administrator, you can use the Amazon Chime API with the AWS SDK or AWS CLI to view your chatbot details. You can also enable or stop your chatbots from being used in your account. You can also regenerate security tokens for your chatbot.

For more information, see the following topics in the *Amazon Chime API Reference*:

- GetBot – Gets your chatbot details, such as bot email address and bot type.
- UpdateBot – Enables or stops a chatbot from being used in your account.
- RegenerateSecurityToken – Regenerates the security token for your chatbot.

You can also change the `PutEventsConfiguration` for your chatbot. For example, if your chatbot was initially configured to use an outbound HTTPS endpoint, you can delete the previous events configuration and put a new events configuration for a Lambda function ARN.

For more information, see the following topics in the *Amazon Chime API Reference*:

- DeleteEventsConfiguration
- PutEventsConfiguration

# Amazon Chime events sent to chatbots

The following events are sent to your chatbot from Amazon Chime:

- **Invite** – Sent when your chatbot is added to an Amazon Chime chat room
- **Mention** – Sent when a user in a chat room @mentions your chatbot
- **Remove** – Sent when your chatbot is removed from an Amazon Chime chat room

The following examples show the JSON payload sent to your chatbot for each of these events.

**Example : Invite event**

```
            {
              "Sender": {
                            "SenderId": "user@example.com",
                            "SenderIdType": "EmailId"
                          },
              "Discussion": {
                            "DiscussionId": "abcdef12-g34h-56i7-j8kl-mn9opqr012st",
                            "DiscussionType": "Room"
                            },
              "EventType": "Invite",
              "InboundHttpsEndpoint": {
                                        "EndpointType": "Persistent",
                                        "Url": "https://
hooks.a.chime.aws/incomingwebhooks/a1b2c34d-5678-90e1-f23g-h45i67j8901k?
token=ABCDefGHiJK1LMnoP2Q3RST4uvwxYZAbC56DeFghIJkLM7N8OP9QRsTuV0WXYZABcdefgHiJ"
                                      },
              "EventTimestamp": "2019-04-04T21:27:52.736Z"
            }
```

**Example : Mention event**

```
            {
                "Sender": {
                            "SenderId": "user@example.com",
                            "SenderIdType": "EmailId"
                          },
                "Discussion": {
                                "DiscussionId": "abcdef12-g34h-56i7-j8kl-mn9opqr012st",
                                "DiscussionType": "Room"
                              },
                "EventType": "Mention",
                "InboundHttpsEndpoint": {
                                            "EndpointType": "ShortLived",
                                            "Url": "https://
hooks.a.chime.aws/incomingwebhooks/a1b2c34d-5678-90e1-f23g-h45i67j8901k?
token=ABCDefGHiJK1LMnoP2Q3RST4uvwxYZAbC56DeFghIJkLM7N8OP9QRsTuV0WXYZABcdefgHiJ"
                                          },
                "EventTimestamp": "2019-04-04T21:30:43.181Z",
                "Message": "@botDisplayName@example.com Hello Chatbot"
            }
```

**Note**
The `InboundHttpsEndpoint` URL for a Mention event expires 2 minutes after it is sent.

**Example : Remove event**

```
            {
                "Sender": {
                            "SenderId": "user@example.com",
                            "SenderIdType": "EmailId"
                          },
                "Discussion": {
                                "DiscussionId": "abcdef12-g34h-56i7-j8kl-mn9opqr012st",
                                "DiscussionType": "Room"
                              },
                "EventType": "Remove",
                "EventTimestamp": "2019-04-04T21:27:29.626Z"
            }
```

# Proxy phone sessions

Developers can use the AWS Command Line Interface (AWS CLI), Amazon Chime API, or AWS SDK to create proxy phone sessions for use with Amazon Chime Voice Connectors. Proxy phone sessions allow participants to call or send text messages to each other without revealing private phone numbers.

Creating proxy phone sessions requires the following:

- The ability to program.
- An AWS account.
- An AWS Identity and Access Management (IAM) role that grants permission to access the Amazon Chime API actions used to create proxy phone sessions, such as the following:
  - `chime:CreateProxySession`

- `chime:DeleteProxySession`
- `chime:DeleteVoiceConnectorProxy`
- `chime:GetProxySession`
- `chime:GetVoiceConnectorProxy`
- `chime:ListProxySessions`
- `chime:PutVoiceConnectorProxy`
- `chime:UpdateProxySession`

  For more information, see Amazon Chime identity-based policies in the *Amazon Chime Administrator Guide*.
- An Amazon Chime Voice Connector created by an Amazon Chime account administrator. For more information, see Managing Amazon Chime Voice Connectors in the *Amazon Chime Administrator Guide*.

The following procedure demonstrates how to create a proxy phone session.

**To create a proxy phone session**

1. Use the PutVoiceConnectorProxy action in the *Amazon Chime API Reference* to configure the Amazon Chime Voice Connector for the proxy phone session.
2. Use the CreateProxySession action in the *Amazon Chime API Reference* to create the proxy phone session.

For more information about the available Amazon Chime API actions for proxy phone sessions, see the *Amazon Chime API Reference*.

# Webhooks for Amazon Chime

Incoming webhooks that you create can programmatically send messages to Amazon Chime chat rooms. For example, a webhook can notify a customer service team about the creation of a new high-priority ticket, and add a link to the ticket in the chat room.

Webhooks messages can be formatted with markdown and can include emojis. HTTP links and email addresses render as active links. Messages can also include @All and @Present annotations to alert all members and present members of a chat room, respectively. To directly @mention a chat room participant, use their alias or full email address. For example, `@alias` or `@alias@domain.com`.

Webhooks can only be part of a chat room and can't be shared. Amazon Chime chat room administrators can add up to 10 webhooks for each chat room.

After you create a webhook, you can integrate it with an Amazon Chime chat room, as shown in the following procedure.

**To integrate a webhook with a chat room**

1. Get the webhook URL from the chat room administrator. For more information, see Adding webhooks to chat rooms in the *Amazon Chime User Guide*.
2. Use the webhook URL in the script or application that you created to send messages to the chat room:

   a. The URL accepts an HTTP POST request.

   b. Amazon Chime webhooks accept a JSON payload with a single key **Content**. The following is a sample curl command with a sample payload:

```
curl -X POST "<Insert your webhook URL here>" -H "Content-Type:application/json"
 --data '{"Content":"Message Body emoji test: :) :+1: link test: http://sample.com
 email test: marymajor@example.com All member callout: @All All Present member
 callout: @Present"}'
```

The following is a sample PowerShell command for Windows users:

```
Invoke-WebRequest -Uri '<Insert your webhook URL here>' -Method 'Post' -ContentType
 'application/JSON' -Body '{"Content":"Message Body emoji test: :) :+1: link test:
 http://sample.com email test: marymajor@example.com All member callout: @All All
 Present member callout: @Present"}'
```

After the external program sends the HTTP POST to the webhook URL, the server validates that the webhook is valid and has an assigned chat room. The webhook appears in the chat room roster with a webhook icon next to its name. Chat room messages sent by the webhook appear in the chat room under the webhook name followed by **(Webhook)**.

**Note**
CORS is not currently enabled for webhooks.

# Troubleshooting webhook errors

The following is a list of webhook-related errors:

- The incoming webhook rate limit for each webhook is 1 TPS per chat room. Throttling results in an HTTP 429 error.
- Messages posted by a webhook must be 4 KB or less. A bigger message payload results in an HTTP 413 error.
- Messages posted by a webhook with @All and @Present annotations work only for chat rooms with 50 or fewer members. More than 50 members results in an HTTP 400 error.
- If the webhook URL is regenerated, using the old URL results in an HTTP 404 error.
- If the webhook in a room is deleted, using the old URL results in an HTTP 404 error.
- Invalid webhook URLs result in HTTP 403 errors.
- If the service is unavailable, the user receives an HTTP 503 error in the response.

# Using the Amazon Chime SDK

You use the Amazon Chime SDK to build real-time media applications that can send and receive audio and video and allow content sharing. The Amazon Chime SDK works independently of any Amazon Chime administrator accounts, and it does not affect meetings hosted on Amazon Chime. Instead, the Amazon Chime SDK provides builder tools for developers to use to build their own meeting applications.

**Topics**



## Amazon Chime SDK prerequisites

Using the Amazon Chime SDK requires the following:

- The ability to program.
- An AWS account.
- An IAM role with a policy that grants permission to access Amazon Chime API actions used by the Amazon Chime SDK, such as the AWS managed **AmazonChimeSDK** policy. For more information, see How Amazon Chime works with IAM and Allow users to access Amazon Chime SDK actions in the *Amazon Chime Administrator Guide*.
- For the majority of use cases, you also need the following:
  - A **server application** – Manages meeting and attendee resources, and serves those resources to the client application. The server application is created in the AWS account and must have access to the IAM role mentioned previously.
  - A **client application** – Receives meeting and attendee information from the server application, and uses that information to make media connections.

## Amazon Chime SDK concepts

The following terminology and concepts are central to understanding how to use the Amazon Chime SDK.

**meeting**

An ephemeral resource identified by a unique `MeetingId`. The `MeetingId` is placed onto a group of media services that host the active meeting.

**media service group**

The group of media services that hosts an active meeting.

**media placement**

A set of regionalized URLs that represents a media service group. Attendees connect to the media service group with their clients to send and receive real-time audio and video, and share their screens.

**attendee**

A meeting participant that is identified by a unique `AttendeeId`. Attendees may freely join and leave meetings using a client application built with an Amazon Chime SDK client library.

**join token**

A unique token assigned to each attendee. Attendees use the join token to authenticate with the media service group.

# Amazon Chime SDK architecture

The following list describes how the different components of the Amazon Chime SDK architecture work together to support meetings and attendees, audio, video, and content sharing.

**Meetings and attendees**

When the server application creates a meeting using the Amazon Chime SDK, the meeting is assigned to a region-specific media service group. The hosts in this group are responsible for securely transferring real-time media between attendee clients. Each created attendee is assigned a unique join token, an opaque secret key that your server application must securely transfer to the client authorized to join the meeting on behalf of an attendee. Each client uses a join token to authenticate with the media service group. Clients use a combination of secure WebSockets and Datagram Transport Layer Security (DTLS) to securely signal the media service group, and to send and receive media to and from other attendees through the media service group.

**Audio**

The media service group mixes audio together from each attendee and sends the mix to each recipient, after subtracting their own audio from the mix. The Amazon Chime SDK for JavaScript samples audio at the highest sample rate supported by the device and browser, up to a maximum of 48kHz. Audio is encoded using the Opus codec, with a default bitrate of 32kbps, which can be increased to up to 64kbps. The Amazon Chime SDKs for iOS and Android sample audio at a rate of 16kHz and encodes using the Opus codec at 32kbps.

**Video**

The media service group acts as a Selective Forwarding Unit (SFU) using a publish and subscribe model. Each attendee can publish one video source, up to a total of 16 simultaneous videos per meeting. The Amazon Chime SDK for JavaScript supports video resolutions up to 1280x720 at 30 frames per second without simulcast, and 15 frames per second with simulcast. The Amazon Chime SDK for iOS and Android support video resolutions up to 1280x720 and 15 frames per second, however the actual framerate and resolution is automatically managed by the Amazon Chime SDK.

When active, video simulcast sends each video stream in two different resolutions and bitrates. Clients which are bandwidth constrained automatically subscribe to the lower bitrate stream. Video encoding and decoding uses hardware acceleration where available to improve performance.

**Data messages**

In addition to audio and video content, meeting attendees can send each other real-time data messages of up to 2 KB each. Developers can use messages to implement custom meeting features such as whiteboarding, chat, real-time emoji reactions, and application-specific floor control signaling.

**Content sharing**

The client application can share audio and video content, such as screen captures or media files. Content sharing supports pre-recorded content video up to 1280x720 at 15 frames per second, and audio up to 48kHz at 64kbps. Screen capture for content sharing is supported up to 15 frames per second, but may be limited by the capabilities of the device and browser.

# Amazon Chime SDK quotas

| Quota | Default | Adjustable |
| --- | --- | --- |
| Active Meetings | 250 | Yes |
| Attendees per meeting | 250 | Yes |
| Audio streams per meeting | 250 | Yes |
| Video tiles per meeting | 16 | Yes |
| Content shares per meeting | 2 | No |
| API Requests per second (RPS) with a burst of 20 RPS. | 10 | Yes |

# Amazon Chime SDK system requirements

The following system requirements apply to applications created with the Amazon Chime SDK.

**Amazon Chime SDK for JavaScript – Supported browsers**

- Mozilla Firefox (version 75 and later), for macOS and Windows
- Google Chrome (version 78 and later), for macOS, Windows, and Ubuntu LTS 16.04 and later
  - Google Chrome for Android also supported for audio and video only (no content sharing)

- Chromium-based Edge version 79 and later for Windows and macOS.
- Chromium-based Electron version 7 and later, with Chromium version 78 and later.
- Safari version 12 for macOS, audio and video only, no content sharing.
- Safari version 12.1.1 and later for iOS, audio and video only, no content sharing.
- Safari version 13 and later for macOS. Content sharing with screen capture requires turning on the **Develop**, **Experimental Features**, **Screen Capture** feature in the browser.
- Opera version 66 and later for macOS and Windows.
- Samsung Internet version 12 and later for Android, audio and video only, no content sharing.

#### Amazon Chime SDK for iOS

- iOS version 10.0 and later

#### Amazon Chime SDK for Android

- Android OS version 5.0 and later, ARM and ARM64 architecture

# Integrating with a client library

Before you can build real-time meeting clients with the Amazon Chime SDK, you must integrate your client application with an Amazon Chime SDK client library. The following client libraries are available:

- Amazon Chime SDK client library for JavaScript (NPM) – A JavaScript library with TypeScript type definitions that helps you build Amazon Chime SDK applications in WebRTC-enabled browsers.
- Amazon Chime SDK client library for iOS – A Swift library that helps you build Amazon Chime SDK applications on supported iOS devices.
- Amazon Chime SDK client library for Android – A Kotlin library that helps you build Amazon Chime SDK applications on supported Android devices.

To learn how to integrate your client application with the Amazon Chime SDK, see the actions in the client library `README.md`. Use the demos to learn how to build specific media components for your application.

# Creating meetings with the Amazon Chime SDK

The following procedure demonstrates how to create a meeting with audio and video for your server and client applications. Before you begin, you must integrate your client application with an Amazon Chime SDK client library. For more information, see Integrating with a client library (p. 17).

#### To create a meeting with audio and video

1. Complete the following steps from your server application:

    a. Use the CreateMeeting API action in the *Amazon Chime API Reference* to create a meeting. Optionally, specify an AWS Region using the `MediaRegion` parameter. For more information about choosing a media Region, see Amazon Chime SDK media Regions (p. 18).

    b. Add attendees to the meeting using the CreateAttendee API action or the BatchCreateAttendee API action. Securely transfer the meeting and attendee from your server application to the client authorized as the respective attendee. For more information about meetings and attendees, see Meeting and Attendee in the *Amazon Chime API Reference*.

2. Complete the following steps from your client application:

    a. Use an Amazon Chime SDK client library to construct a `MeetingSessionConfiguration` object. Use the meeting and attendee information from the previous steps.
    b. Implement the `AudioVideoObserver` interface.
    c. Create a `MeetingSession` using the `MeetingSessionConfiguration`.
    d. Use the `AudioVideoFacade` from the `MeetingSession` to control real-time media.

        i. Register an instance of the `AudioVideoObserver` interface. This lets you receive events when the meeting state changes.
        ii. Select initial devices for the audio input, audio output, and video input.
        iii. Start the audiovisual session.
        iv. Start local video capture when the user wants to share video.
        v. To show video tiles, manage video tile events, and bind the tiles to video surfaces in the client application.
        vi. Manage other user interactions such as muting and unmuting, or starting and stopping local video capture.
        vii. To leave the meeting, stop the audiovisual session.
    e. (Optional) Use the `AudioVideoFacade` from the `MeetingSession` to share media content, such as screen captures, with other clients.

        i. Start the screen share session. The content joins the meeting as an additional attendee.
        ii. To view the shared content, manage video tile events and bind the tiles to surfaces in the client application.
        iii. Manage other interactions, such as pausing, restarting, or stopping the content share.

The meetings end when you run the DeleteMeeting API action. A meeting automatically ends after a period of inactivity, such as the following:

- No audio connections are present in the meeting for more than five minutes.
- Less than two audio connections are present in the meeting for more than 30 minutes.
- Screen share viewer connections are inactive for more than 30 minutes.
- The meeting time exceeds 24 hours.

# Amazon Chime SDK media Regions

We recommend specifying an AWS Region for your Amazon Chime SDK meeting using the `MediaRegion` parameter in the `CreateMeeting` API action. Available media Regions for Amazon Chime SDK meetings include the following:

- US East (Ohio) (us-east-2)
- US East (N. Virginia) (us-east-1)
- US West (N. California) (us-west-1)
- US West (Oregon) (us-west-2)
- Africa (Cape Town) (af-south-1)
- Asia Pacific (Mumbai) (ap-south-1)
- Asia Pacific (Seoul) (ap-northeast-2)
- Asia Pacific (Singapore) (ap-southeast-1)
- Asia Pacific (Sydney) (ap-southeast-2)
- Asia Pacific (Tokyo) (ap-northeast-1)

- Canada (Central) (ca-central-1)
- Europe (Frankfurt) (eu-central-1)
- Europe (Ireland) (eu-west-1)
- Europe (London) (eu-west-2)
- Europe (Milan) (eu-south-1)
- Europe (Paris) (eu-west-3)
- Europe (Stockholm) (eu-north-1)
- South America (São Paulo) (sa-east-1)

For more information about available media Regions, see the `MediaRegion` definition for CreateMeeting in the *Amazon Chime API Reference*.

> **Note**
> The preceding media Regions are used for hosting Amazon Chime SDK meetings. This differs from the Amazon Chime API, which has a single endpoint. For more information, see Amazon Chime endpoints and quotas in the *Amazon Web Services General Reference*.

## Choosing a media Region

When choosing a media Region to use for your Amazon Chime SDK meeting, common factors to consider include the following:

**Regulatory requirements**

If your Amazon Chime SDK meetings are subject to regulations requiring them to be hosted within a geopolitical border, consider hardcoding the meeting Region based on fixed application logic.

For example, a telemedicine application might require all meetings to be hosted within the medical practitioner's jurisdiction. If the application supports clinics located in both Europe and the United States, you can use each clinic's address to select a Region within its jurisdiction.

**Meeting quality**

When an Amazon Chime SDK meeting is hosted in a media Region, each attendee's audio and video is sent and received from that Region. As the distance between the attendee and the Region increases, meeting quality can be affected by network latency. Specifying a Region for your Amazon Chime SDK meeting can help enhance the meeting quality for your attendees, whether they are located near each other or distributed geographically.

You can use one of the following methods to choose a media Region for your Amazon Chime SDK meeting:

**Hardcode a media Region**

Recommended if your Amazon Chime SDK meetings are all hosted within a specific AWS Region.

**Choose the nearest media Region**

Recommended if your Amazon Chime SDK meeting attendees are located in the same AWS Region, but your meetings are hosted in different Regions.

For more details about choosing the nearest media Region, see the following topic.

### Choosing the nearest media Region

Call `https://nearest-media-region.l.chime.aws` to identify the nearest media Region that can host your Amazon Chime SDK meeting. Make the call from the client application, not the server

application. Pass the call to the application before your attendees need to join the meeting, such as at the time that the application starts up.

Your request returns a JSON object showing the AWS Region that is nearest to you.

**Example : Call to `https://nearest-media-region.l.chime.aws`**

The following example shows the contents of a request sent to `https://nearest-media-region.l.chime.aws` to identify the nearest media Region.

```
async getNearestMediaRegion(): Promise<string> {
    var nearestMediaRegion = '';
    const defaultMediaRegion = 'us-east-1';
    try {
      const nearestMediaRegionResponse = await fetch(
        `https://nearest-media-region.l.chime.aws`,
        {
          method: 'GET',
        }
      );
      const nearestMediaRegionJSON = await nearestMediaRegionResponse.json();
      this.log(nearestMediaRegionJSON.region);
      nearestMediaRegion = nearestMediaRegionJSON.region;
    } catch (error) {
      nearestMediaRegion = defaultMediaRegion;
      this.log('Default media region ' + defaultMediaRegion + ' selected: ' +
 error.message);
    } finally {
      return nearestMediaRegion;
    }
  }
```

# SIP integration using an Amazon Chime Voice Connector

Integrate your SIP-compatible voice infrastructure with an Amazon Chime Voice Connector to make SIP voice calls. You must have an IP Private Branch Exchange (PBX), Session Border Controller (SBC), or other voice infrastructure with internet access that supports Session Initiation Protocol (SIP). For more information, see Before you begin in the *Amazon Chime Administrator Guide*.

**To integrate your voice infrastructure with an Amazon Chime Voice Connector**

1. Create an Amazon Chime Voice Connector under your AWS account. For more information, see Creating an Amazon Chime Voice Connector in the *Amazon Chime Administrator Guide*.
2. Edit your Amazon Chime Voice Connector settings to allow calling from your voice infrastructure to AWS. For more information, see Editing Amazon Chime Voice Connector settings in the *Amazon Chime Administrator Guide*.
    a. For **Termination settings**, select **Enabled**.
    b. For **Allowlist**, choose **New**.
    c. Enter the CIDR notations of the IP addresses for your internal SIP infrastructure. This allows your infrastructure to access the Amazon Chime Voice Connector. For example, to allow traffic from IP address `10.24.34.0`, allowlist the CIDR notation `10.24.34.0/32`.
    d. Choose **Add**.
    e. For **Calling plan**, select the country or countries to add to your calling plan.

f. Edit any other settings as needed, and choose **Save**.

3. In the Amazon Chime console, under **Voice connectors**, view the **Outbound host name** for your Amazon Chime Voice Connector. For example, *abcdef1ghij2klmno3pqr4*.voiceconnector.chime.aws.
4. To join a meeting using the Amazon Chime SDK, use a SIP URI to make a SIP request to the **Outbound host name** of your Amazon Chime Voice Connector. Use phone number **+17035550122** in the SIP URI. Set the `transport` parameter to use the TLS protocol. Finally, use the unique join token generated by calling the CreateAttendee API action. For more information, see the following example.

### Example Example: SIP request

The following example shows the contents of a SIP URI used to make a SIP request to an Amazon Chime Voice Connector.

```
sip:+17035550122@abcdef1ghij2klmno3pqr4.voiceconnector.chime.aws;transport=tls;X-chime-
join-token=join-token
```

The following example shows a sample SIP INVITE message to join an Amazon Chime SDK meeting.

```
INVITE sip:+17035550122@abcdef1ghij2klmno3pqr4.voiceconnector.chime.aws;transport=tls;X-
chime-join-token=join-token SIP/2.0
Via: SIP/2.0/TLS IPaddress:12345;rport;branch=branch;alias
Max-Forwards: 70
From: sip:+12065550100@IPaddress;tag=tag
To: sip:+17035550122@abcdef1ghij2klmno3pqr4.voiceconnector.chime.aws;X-chime-join-
token=join-token
Contact: <sip:+12065550100@IPaddress:54321;transport=TLS;ob>
Call-ID: a1234567-89b0-1c2d-e34f-5gh678j9k2lm
CSeq: 6214 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE,
 OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
Content-Type: application/sdp
Content-Length:   991

v=0
o=- 3775321410 3775321410 IN IP4 IPaddress
s=pjmedia
b=AS:117
t=0 0
a=X-nat:0
m=audio 4000 RTP/SAVP 0 3 8 9 125 101
c=IN IP4 IPaddress
b=TIAS:96000
a=rtcp:4001 IN IP4 IPaddress
a=sendrecv
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:9 G722/8000
a=rtpmap:125 opus/48000/2
a=fmtp:125 useinbandfec=1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=crypto:1 AEAD_AES_256_GCM inline:EXAMPLE
a=crypto:2 AEAD_AES_256_GCM_8 inline:EXAMPLE
a=crypto:3 AES_256_CM_HMAC_SHA1_80 inline:EXAMPLE
```

```
a=crypto:4 AES_256_CM_HMAC_SHA1_32 inline:EXAMPLE
a=crypto:5 AES_CM_128_HMAC_SHA1_80 inline:EXAMPLE
a=crypto:6 AES_CM_128_HMAC_SHA1_32 inline:EXAMPLE
```

**Note**
Amazon Chime recognizes phone numbers only in E.164 format. Make sure that an E.164 phone number is in your `From` header.

# Amazon Chime SDK event notifications

The Amazon Chime SDK supports sending meeting event notifications to Amazon EventBridge, Amazon Simple Queue Service (Amazon SQS), and Amazon Simple Notification Service (Amazon SNS).

**Note**
The services listed here can go down. As a best practice, app builders should subscribe to multiple notification targets in order to enable higher availability for Chime meeting events.

## Sending notifications to EventBridge

We recommend sending Amazon Chime SDK Event notifications to EventBridge. Events are emitted on a best-effort basis for Amazon Chime SDK events. For detailed information about using the Amazon Chime SDK with EventBridge, see Automating the Amazon Chime SDK with EventBridge in the *Amazon Chime Administrator Guide*. For information about EventBridge, see the Amazon EventBridge User Guide.

## Sending notifications to Amazon SQS and Amazon SNS

You can use the CreateMeeting API in the *Amazon Chime API Reference* to send Amazon Chime SDK meeting event notifications to one Amazon SQS queue and one Amazon SNS topic per meeting. This can help reduce notification latency. For more information about Amazon SQS, see the Amazon Simple Queue Service Developer Guide. For more information about Amazon SNS, see the Amazon Simple Notification Service Developer Guide.

The notifications sent to Amazon SQS and Amazon SNS contain the same information as the notifications that Amazon Chime sends to EventBridge. The Amazon Chime SDK supports sending meeting event notifications to queues and topics in the US East (N. Virginia) (**us-east-1**) AWS Region. Event notifications might be delivered out of order of occurrence.

## Granting the Amazon Chime SDK access to Amazon SQS and Amazon SNS

If you have an Amazon SQS queue or Amazon SNS topic configured in the **us-east-1** Region and you want to send Amazon Chime SDK events to it, you must grant the Amazon Chime SDK permission to publish messages to the Amazon Resource Name (ARN) of the queue or topic. To do this, attach an AWS Identity and Access Management (IAM) policy to the queue or topic that grants the appropriate permissions to the Amazon Chime SDK. For more information, see Identity and access management in Amazon SQS in the *Amazon Simple Queue Service Developer Guide* and Example cases for Amazon SNS access control in the *Amazon Simple Notification Service Developer Guide*.

**Example : Allow the Amazon Chime SDK to publish events to an Amazon SQS queue**

The following example IAM policy grants the Amazon Chime SDK permission to publish meeting event notifications to the specified Amazon SQS queue.

```
{
 "Version": "2008-10-17",
 "Id": "example-ID",
 "Statement": [
  {
   "Sid": "example-statement-ID",
   "Effect": "Allow",
   "Principal": {
     "Service": "chime.amazonaws.com"
   },
   "Action": [
    "sqs:SendMessage",
    "sqs:GetQueueUrl"
   ],
   "Resource": "arn:aws:sqs:us-east-1:111122223333:queueName",
  }
 ]
}
```

If the Amazon SQS queue is enabled for server-side encryption (SSE), you must take an additional step. Attach an IAM policy to the associated AWS KMS key that grants the Amazon Chime SDK permission to the AWS KMS actions needed to encrypt data added to the queue.

```
{
    "Version": "2012-10-17",
    "Id": "example-ID",
    "Statement": [
        {
            "Sid": "example-statement-ID",
            "Effect": "Allow",
            "Principal": {
                "Service": "chime.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": ""
        }
    ]
}
```

### Example : Allow the Amazon Chime SDK to publish events to an Amazon SNS topic

The following example IAM policy grants the Amazon Chime SDK permission to publish meeting event notifications to the specified Amazon SNS topic.

```
{
 "Version": "2008-10-17",
 "Id": "example-ID",
 "Statement": [
  {
   "Sid": "allow-chime-sdk-access-statement-id",
   "Effect": "Allow",
   "Principal": {
     "Service": "chime.amazonaws.com"
   },
   "Action": [
    "SNS:Publish"
   ],
   "Resource": "arn:aws:sns:us-east-1:111122223333:topicName"
  }
```

```
  ]
}
```

# Using Amazon Chime SDK messaging

You use this SDK to help create messaging applications that run on the Amazon Chime service. This SDK provides the conceptual and practical information needed to create a basic messaging app.

**Topics**



## Messaging prerequisites

You need the following to use Amazon Chime SDK messaging.

- The ability to program.
- An AWS account.
- An IAM role with a policy that grants permission to access the API actions used by the Amazon Chime SDK. For more information, see How Amazon Chime works with IAM and Allow users to access Amazon Chime SDK actions in the *Amazon Chime Administrator Guide*.

For the majority of cases, you also need:

- **A server application** – Manages identity and users, and serves those resources to the client application. The server application is created in the AWS account and must have access to the IAM role mentioned previously.
- **A client application** – Displays messaging UI, connects to web sockets using the Amazon Chime Client SDKs, manages state.

## Messaging concepts

To use Amazon Chime SDK messaging effectively, you must understand the following terminology and concepts.

**AppInstance**

To use Amazon Chime SDK messaging, you must first create an `AppInstance`. Each `AppInstance` is identified by a unique `AppInstanceARN`. You enable settings, such as message retention and streaming configuration for message export, at the `AppInstance` level. App instances contain at least one `AppInstanceUser`, plus channels.

**AppInstanceUser**

App instance users are resources that represent your users, and they're typically associated with users in your identity provider. App instance users are represented with unique IDs and ARNs. An app instance user can be promoted to `ChannelModerator` for elevated privileges in individual channels, or `AppInstanceAdmin` for elevated privileges across all the channels in an app instance.

**Channel**

When you add an app instance user to a channel, that user becomes a member and can send and receive messages. Channels can be public, which allows any user to add themselves as a member, or private, which allows only channel moderators to add members. You can also hide channel members. Hidden members can observe conversations but not send message, and they aren't added to channel membership

**ChannelMessage**

ChannelMessages can be either `STANDARD` or `CONTROL` messages. `STANDARD` messages can contain 4KB of data and the 1KB of metadata. `CONTROL` messages can contain only 30 bytes of data.

**System Message**

Amazon Chime generates system messages in response to events such as members joining or leaving a channel.

# Messaging architecture

The Amazon Chime SDK messaging uses the following architecture.

**App instances, app instance users, and channels**

App instances contain app instance users and channels. You configure settings such as message retention policies and streaming export of message data at the app instance level. You also do the same for most limits. App instance users can only communicate with other users in the same app instance.

**Messages**

Messages are sent in channels, and can be `STANDARD`, `CONTROL`, or `SYSTEM` messages.

- `STANDARD` messages can be up to 4KB in size and contain metadata, which can contain a link to an attachment.
- `CONTROL` messages are limited to 30 bytes and do not contain metadata.
- `STANDARD` and `CONTROL` messages can be persistent or not persistent.
- `SYSTEM` messages are automated messages sent by Amazon Chime for events such as members joining or leaving a channel

# Messaging quotas

The Amazon Chime SDK messaging enforces the following quotas.

| Resource | Limit | Eligible for increase |
|---|---|---|
| App Instances Per AWS Account | 100 | Yes |
| Users per app instance | 100,000 | Yes |
| App instance admins per app instance | 100 | Yes |
| Channels Per AppInstance | 10,000,000 | Yes |
| Memberships Per Channel | 10,000 | Yes |
| Moderators per channel | 1,000 | Yes |
| Max concurrent connections per user | 10 | Yes |

# Getting started

The topics in this section explain how to start building an Amazon Chime messaging application.

**Topics**



## Creating an app instance

To use Amazon Chime SDK messaging, you must first create an Amazon Chime app instance within your AWS account.

**To create an app instance for messaging**

1. Create an Amazon Chime app instance. `aws chime create-app-instance --name >NameOfAppInstance<`
2. In the create response, make note of the `AppInstanceArn`. You use this to create an IAM policy for your users.

## Connecting to your identity provider

Amazon Chime SDK messaging integrates natively with AWS Identity and Access Management (IAM) policies to authenticate incoming requests. The IAM policy defines what an individual user can do. IAM policies can be crafted to provide scoped-down limited credentials for your use case. For more information on creating policies for Amazon Chime SDK messaging users, see Defining IAM roles and policies for AppInstance users (p. 30).

If you have an existing identity provider, you have the following options for integrating your existing identity with Amazon Chime SDK messaging.

- You can use your existing identity provider to authenticate users and then integrate the authentication service with AWS STS to create your own credential vending service for clients. The AWS Security Token Service (STS) provides APIs to assume IAM Roles.
- If you already have a SAML or OpenID compatible identity provider, Amazon Chime recommends using Amazon Cognito Identity Pools, which abstract away calls to AWS STS AssumeRoleWithSAML and AssumeRoleWithWebIdentity, respectively. Amazon Cognito integrates with OpenID, SAML, and public identity providers such as Facebook, Login with Amazon, Google, and Sign in with Apple.

If you do not have an identity provider, you can get started quickly with Amazon Cognito User Pools. For an example of how to use Amazon Cognito with the Amazon Chime SDK messaging features, see Build chat features into your application with Amazon Chime SDK messaging.

Alternately, you can use the AWS STS to create your own credential vending service or build your own identity provider.

**Using STS to vend credentials**

If you already have an IDP such as ActiveDirectory LDAP, and you want to implement a custom credential vending service, or grant access to chat for non-authenticated meeting attendees, you can use the AWS STS AssumeRole API. To do this, you first create a Chime Messaging SDK Role. For more information about creating that role, see Creating a role to delegate permissions to an IAM user .

The IAM Role would have permissions to the Chime Messaging SDK action your application would use, similar to the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": [
                "chime:GetMessagingSessionEndpoint"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "chime:SendChannelMessage",
                "chime:ListChannelMessages",
                "chime:CreateChannelMembership",
                "chime:ListChannelMemberships",
                "chime:DeleteChannelMembership",
                "chime:CreateChannelModerator",
                "chime:ListChannelModerators",
                "chime:DescribeChannelModerator",
                "chime:CreateChannel",
                "chime:DescribeChannel",
                "chime:ListChannels",
                "chime:DeleteChannel",
                "chime:RedactChannelMessage",
                "chime:UpdateChannelMessage",
                "chime:Connect",
                "chime:ListChannelBans",
                "chime:CreateChannelBan",
                "chime:DeleteChannelBan",
                "chime:ListChannelMembershipsForAppInstanceUser"
```

```
            ],
            "Resource": [
                "{Chime_App_Instance_Arn}/user/${my_applications_user_id}",
                "{Chime_App_Instance_Arn}/channel/*"
            ]
        }
    ]
}
```

For this example, call this role the *ChimeMessagingSampleAppUserRole*.

Note the session tag in the *ChimeMessagingSampleAppUserRole* policy `${my_application_user_id}` in the user ARN resource. This session tag is parameterized in the AssumeRole API call to limit the credentials returned to permissions for a single user.

The AssumeRole API call is called using an already credentialed IAM entity such as an IAM user. It can also be called by a different IAM role such as an AWS Lambda execution role. That IAM identity must have permissions to call AssumeRole on the *ChimeMessagingSampleAppUserRole*.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource":
 "arn:aws:iam::<MY_AWS_ACCOUNT_ID>:role/ChimeMessagingSampleAppUserRole"
        }
    ]
}
```

For this example, call this role the *ChimeSampleAppServerRole*.

You need to set up the *ChimeMessagingSampleAppUserRole* with a trust policy that allows the *ChimeMessagingSampleAppServerRole* to call the STS AssumeRole API on it. For more information about using trust policies with IAM roles, see How to use trust policies with IAM roles . You can use the AWS IAM Roles Console to add this policy to the *ChimeMessagingSampleAppUserRole*. The following example shows a typical trust relationship.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Principal": {

 "AWS":"arn:aws:iam::<MY_AWS_ACCOUNT_ID>:role/ChimeMessagingSampleAppServerRole"
            }
            "Action": "sts:AssumeRole"
        }
    ]
}
```

In a sample deployment, an Amazon EC2 instance, or an AWS Lambda, is launched with the `ChimeMessagingSampleAppServerRole`. The server then:

1. Performs any application specific authorization on a client's requests to receive credentials.
2. Calls STS `AssumeRole` on `ChimeMessagingSampleAppUserRole`, with a tag parameterizing the `${my_applications_user_guid}`.
3. Forwards the credentials returned in the `AssumeRole` call to the user.

The following example shows CLI command for assuming a role for step 2:

```
aws sts assume-role --role-arn
arn:aws:iam::<MY_AWS_ACCOUNT_ID>:role/ChimeMessagingSampleAppUserRole --role-
session-name demo --tags Key=my_applications_user_guid,Value=123456789
```

# Defining IAM roles and policies for AppInstance users

For users to access the Amazon Chime SDK messaging features, you must define an IAM role and policy to vend credentials to users when they sign in. The IAM policy defines the resources that users can access.

This example shows a policy for developers building applications using Amazon Chime SDK messaging.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "chime:CreateAppInstance",
                "chime:DescribeAppInstance",
                "chime:ListAppInstances",
                "chime:UpdateAppInstance",
                "chime:DeleteAppInstance",
                "chime:CreateAppInstanceUser",
                "chime:DeleteAppInstanceUser",
                "chime:ListAppInstanceUsers",
                "chime:UpdateAppInstanceUser",
                "chime:DescribeAppInstanceUser",
                "chime:CreateAppInstanceAdmin",
                "chime:DescribeAppInstanceAdmin",
                "chime:ListAppInstanceAdmins",
                "chime:DeleteAppInstanceAdmin",
                "chime:PutAppInstanceRetentionSettings",
                "chime:GetAppInstanceRetentionSettings",
                "chime:PutAppInstanceStreamingConfigurations",
                "chime:GetAppInstanceStreamingConfigurations",
                "chime:DeleteAppInstanceStreamingConfigurations",
                "chime:TagResource",
                "chime:UntagResource",
                "chime:ListTagsForResource"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

This example shows a policy that allows users to access the Amazon Chime SDK user actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "chime:GetMessagingSessionEndpoint",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "chime:CreateChannel",
                "chime:DescribeChannel",
                "chime:DeleteChannel",
```

```
                "chime:UpdateChannel",
                "chime:ListChannels",
                "chime:ListChannelMembershipsForAppInstanceUser",
                "chime:DescribeChannelMembershipForAppInstanceUser",
                "chime:ListChannelsModeratedByAppInstanceUser",
                "chime:DescribeChannelModeratedByAppInstanceUser",
                "chime:UpdateChannelReadMarker",
                "chime:CreateChannelModerator",
                "chime:DescribeChannelModerator",
                "chime:ListChannelModerators",
                "chime:DeleteChannelModerator",
                "chime:SendChannelMessage",
                "chime:GetChannelMessage",
                "chime:DeleteChannelMessage",
                "chime:UpdateChannelMessage",
                "chime:RedactChannelMessage",
                "chime:ListChannelMessages",
                "chime:CreateChannelMembership",
                "chime:DescribeChannelMembership",
                "chime:DeleteChannelMembership",
                "chime:ListChannelMemberships",
                "chime:CreateChannelBan",
                "chime:DeleteChannelBan",
                "chime:ListChannelBans",
                "chime:DescribeChannelBan",
                "chime:Connect"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:chime:us-east-1:{awsAccountId}:app-instance/{appInstanceId}/user/
{appInstanceUserId}",
                "arn:aws:chime:us-east-1:{awsAccountId}:app-instance/{appInstanceId}/
channel/*"
            ]
        }
    ]
}
```

This example shows a policy that gives users minimal access to Amazon Chime SDK user actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "chime:GetMessagingSessionEndpoint",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "chime:ListChannels",
                "chime:DescribeChannel",
                "chime:ListChannelMembershipsForAppInstanceUser",
                "chime:DescribeChannelMembershipForAppInstanceUser",
                "chime:ListChannelsModeratedByAppInstanceUser",
                "chime:DescribeChannelModeratedByAppInstanceUser",
                "chime:SendChannelMessage",
                "chime:GetChannelMessage",
                "chime:ListChannelMessages",
                "chime:Connect"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:chime:us-east-1:{awsAccountId}:app-instance/{appInstanceId}/user/
{appInstanceUserId}",
```

```
                "arn:aws:chime:us-east-1:{awsAccountId}:app-instance/{appInstanceId}/
channel/*"
            ]
        }
    ]
}
```

This example shows a policy for establishing a websocket connection for an `AppInstanceUser`. For more information about websocket connections, see Using websockets to receive messages (p. 32).

```
{
  "Version": "2012-10-17",
  "Statement": [
   {
     "Effect": "Allow",
     "Action": [
             "chime:Connect"
            ],
     "Resource": [
              "arn:aws:chime:us-east-1:{awsAccountId}:app-instance/{appInstanceId}/user/
{appInstanceUserId}"
         ]
      }
   ]
}
```

# Creating channels

You and your end users can create channels. Once created, you or your end users also need to add members to the channel. Sample code for creating channels is available in the sample application.

For more information on creating channels and adding members, see:

- CreateChannel
- CreateChannelMembership

# Sending messages

Use the SendChannelMessage API to send messages. Sample code is available in the sample application.

# Using websockets to receive messages

You can use the Amazon Chime JS SDK to receive messages using websockets, or you can also use another websocket client library of your choice.

**Topics**



## Establishing a connection

Follow these steps to establish a websocket connection:

1. Define an IAM policy that gives you permission to establish a websocket connection. For an example, see the last policy in Defining IAM roles and policies for AppInstance users (p. 30).
2. Use the GetMessagingSessionEndpoint API to retrieve the websocket endpoint.
3. Use the Connect API (p. 34) to connect to the websocket endpoint.

## Receiving messages

In order for an AppInstanceUser to start receiving messages after they successfully establish a connection, they must be added to a Channel. You can add AppInstanceUsers to a Channel via the CreateChannelMembership API.

> **Note**
> An AppInstanceUser always receives messages for all channels that they are a member of as long as they have established a connection successfully.

AppInstanceAdmins and ChannelModerators do not receive messages on a channel unless you use the CreateChannelMembership API to explicitly add them.

## Message structures

Every WebSocket message that you receive adheres to this format:

```
{
   "Headers": {"Key": "Value"},
   "Payload": "{\"Key\": \"Value\"}"
}
```

**Headers**

This map shows different possible header keys and their respective values:

1. **x-amz-chime-message-type**
   a. **STANDARD** – These are standard messages up to 4kb in size sent via the SendChannelMessage API.
   b. **CONTROL** – These are messages up to 30 bytes in size, and sent using the SendChannelMessage API.
   c. **SYSTEM** – These are messages generated by Amazon Chime. For more information, see Understanding system messages (p. 35).
2. **x-amz-chime-event-type** – For more information about the values, see Understanding system messages (p. 35).
3. **x-amz-chime-membership-type** – This header is only returned for SYSTEM messages related to membership updates. For more information about the values, see the ChannelMembership types.

**Payload**

Websocket messages return JSON strings. The structure of the JSON string depends on the `x-amz-event-type` headers.The following table lists the possible `x-amz-chime-event-type` values and payloads:

| EventType | Payload format | |
|---|---|---|
| CREATE_CHANNEL_MESSAGE | Channel message | |
| REDACT_CHANNEL_MESSAGE | | |

| EventType | Payload format | |
|---|---|---|
| UPDATE_CHANNEL_MESSAGE | | |
| DELETE_CHANNEL_MESSAGE | | |
| | | |
| UPDATE_CHANNEL | Channel | |
| DELETE_CHANNEL | | |
| | | |
| CREATE_CHANNEL_MEMBERSHIP | ChannelMembership | |
| DELETE_CHANNEL_MEMBERSHIP | | |

## Connect API

Establishes a websocket connection to the back-end server to receive messages for an `AppInstanceUser`. The request has to be signed using AWS Signature Version 4. For more information about signing a request, see Signing AWS Requests with Signature Version 4 .

**Note**
To retrieve the endpoint, you can invoke the GetMessagingSessionEndpoint API. You can use the websocket client library of your choice to connect to the endpoint.

**Request Syntax**

```
GET /connect
?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIARALLEXAMPLE%2F20201214%2Fus-east-1%2Fchime%2Faws4_request
&X-Amz-Date=20201214T171359Z
&X-Amz-Expires=10
&X-Amz-SignedHeaders=host
&sessionId={sessionId}
&userArn={appInstanceUserArn}
&X-Amz-Signature=db75397d79583EXAMPLE
```

**URI Request Parameters**

All URI Request Query Parameters must be URL Encoded.

**X-Amz-Algorithm**

Identifies the version of AWS Signature and the algorithm that you used to calculate the signature. Amazon Chime supports only AWS Signature Version 4 authentication, so the value of this is `AWS4-HMAC-SHA256`.

**X-Amz-Credential**

In addition to your access key ID, this parameter also provides the AWS region and service—the scope—for which the signature is valid. This value must match the scope you use in signature calculations. The general form for this parameter value is:

`<your-access-key-id>/<date>/<AWS-region>/<AWS-service>/aws4_request`

For example:

`AKIAIOSFODNN7EXAMPLE/20201214/us-east-1/chime/aws4_request`

**X-Amz-Date**

The date and time format must follow the ISO 8601 standard, and must be formatted as `yyyyMMddTHHmmssZ`. For example if the date and time was **08/01/2020 15:32:41.982-700**, then you must first convert it to UTC (Coordinated Universal Time) and then submitted as `20200801T083241Z`.

**X-Amz-Signed-Headers**

Lists the headers that you used to calculate the signature. The following headers are required in the signature calculations:

- The HTTP host header.
- Any x-amz-* headers that you plan to add to the request.

> **Note**
> For added security, sign all the request headers that you plan to include in your request.

**X-Amz-Signatures**

Provides the signature to authenticate your request. This signature must match the signature that Amazon Chime calculates. If it doesn't, Amazon Chime denies the request. For example, `733255ef022bec3f2a8701cd61d4b371f3f28c9f19EXAMPLEd48d5193d7`.

**X-Amz-Security-Token**

Optional credential parameter if using credentials sourced from the STS service.

**SessionId**

Indicates a unique Id for the websocket connection being established.

**UserArn**

Indicates the identity of the AppInstanceUser that is trying to establish a connection. The value should be the ARN of the `AppInstanceUser`. For example, `arn:aws:chime:us-east-1:123456789012:app-instance/694d2099-cb1e-463e-9d64-697ff5b8950e/user/johndoe`

## Configuring attachments

The Amazon Chime SDK allows you to use your own storage for message attachments, and include them as message metadata. Amazon Simple Storage Service (S3) is the easiest way to get started with attachments.

**To use S3 for attachments**

1. Create an S3 bucket to store attachments.
2. Create an IAM policy for the bucket that allows Amazon Chime SDK users to upload, download, and delete attachments from your S3 bucket.
3. Create an IAM role for use by your Identity provider to vend credentials to users for attachments.

The sample application provides an example of how to do this with Amazon S3, Amazon Cognito, and the Amazon Chime SDK.

# Understanding system messages

Amazon Chime sends system messages to all connected clients for events that take place in channels. Events include:

- `UPDATE_CHANNEL` – This event signifies any update made to the channel details, such as the name or metadata.
- `DELETE_CHANNEL` – This event signifies that the channel and all of its data, including messages, memberships, moderators and bans, will be deleted
- `CREATE_CHANNEL_MEMBERSHIP` – This event signifies that a particular `AppInstanceUser` has been added as a member to the channel. The event also contains details of the new `AppInstanceUser`.
- `DELETE_CHANNEL_MEMBERSHIP` – This event signifies that an `AppInstanceUser` has been removed from the channel. The event also contains the removed `AppInstanceUser` details.

# Understanding authorization by role

The tables in this topic list the actions that app instance users can execute, depending on their role.

### Legend

- **Allowed** – If the correct Action/Resource context is specified in the IAM Policy, then it can be successfully executed.
- **Allowed with restrictions** – If correct Action/Resource context is specified in the IAM Policy then certain conditions have to be met to successfully execute the action.
- **Denied** – Even if correct Action/Resource context is specified in the IAM Policy, it will still be blocked by the back end.

### Topics



## AppInstanceAdmin

App instance administrators can perform actions on a channels within the app instance for which they are an admin.

| API name | Allowed or denied | Notes |
|---|---|---|
| `UpdateChannel` | Allowed | |
| `DeleteChannel` | Allowed | |
| `DescribeChannel` | Allowed | |
| `ListChannel` | Allowed | |
| `ListChannelMembershipsForAppInstanceUser` | Allowed | You can also populate AppInstanceUserArn with another AppInstanceUser. |
| `DescribeChannelMembershipForAppInstanceUser` | Allowed | You can also populate AppInstanceUserArn with another AppInstanceUser. |

| API name | Allowed or denied | Notes |
| --- | --- | --- |
| `ListChannelsModeratedByAppInstanceUser` | Allowed | You can also populate AppInstanceUserArn with another AppInstanceUser. |
| `DescribeChannelModeratedByAppInstanceUser` | Allowed | You can also populate AppInstanceUserArn with another AppInstanceUser. |
| `CreateChannelMembership` | Allowed | |
| `DescribeChannelMembership` | Allowed | |
| `ListChannelMembership` | Allowed | |
| `DeleteChannelMembership` | Allowed | |
| `SendChannelMessage` | Allowed with restriction | You first need to use CreateChannelMembership to create a membership for yourself, and then call the API. |
| `GetChannelMessage` | Allowed | |
| `ListChannelMessage` | Allowed | |
| `DeleteChannelMessage` | Allowed | |
| `RedactChannelMessage` | Allowed | |
| `UpdateChannelMessage` | Allowed with restriction | You can only edit your own messages. |
| `CreateChannelModerator` | Allowed | |
| `DeleteChannelModerator` | Allowed | |
| `DescribeChannelModerator` | Allowed | |
| `ListChannelModerator` | Allowed | |
| `CreateChannelBan` | Allowed with restriction | The `AppInstanceUser` that you ban cannot be an `AppInstanceAdmin` or the moderator of that channel. |
| `DeleteChannelBan` | Allowed with restriction | |
| `DescribeChannelBan` | Allowed | |
| `ListChannelBan` | Allowed | |
| `UpdateChannelReadMarker` | Allowed with restriction | You need to use CreateChannelMembership to create a membership for yourself first, and then call the API. |

# ChannelModerator

Channel moderators can perform actions only on channels for which they have the moderator role.

**Note**
A moderator who is an `AppInstanceAdmin` can perform actions on channels allowed by that role.

| API name | Allowed or denied | Notes |
|---|---|---|
| `UpdateChannel` | Allowed | |
| `DeleteChannel` | Allowed | |
| `DescribeChannel` | Allowed with restriction | You can only get details for public channels. |
| `ListChannel` | Allowed with restriction | You can only get details for public channels. |
| `ListChannelMembershipsForAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `DescribeChannelMembershipForAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `ListChannelsModeratedByAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `DescribeChannelModeratedByAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `CreateChannelMembership` | Allowed | |
| `DescribeChannelMembership` | Allowed | |
| `ListChannelMembership` | Allowed | |
| `DeleteChannelMembership` | Allowed | |
| `SendChannelMessage` | Allowed with restriction | You need to use CreateChannelMembership to create a membership for yourself first, and then call the API. |
| `GetChannelMessage` | Allowed | |
| `ListChannelMessage` | Allowed | |
| `DeleteChannelMessage` | Denied | |
| `RedactChannelMessage` | Allowed | |
| `UpdateChannelMessage` | Allowed with restriction | You can only update your own messages. |
| `CreateChannelModerator` | Allowed | You need to use CreateChannelMembership to create a membership for |

| API name | Allowed or denied | Notes |
|---|---|---|
| | | yourself first, and then call the API. |
| `DeleteChannelModerator` | Allowed | |
| `DescribeChannelModerator` | Allowed | |
| `ListChannelModerator` | Allowed | |
| `CreateChannelBan` | Allowed with restriction | The `AppInstanceUser` you are banning cannot be an `AppInstanceAdmin` or the moderator of that channel. |
| `DeleteChannelBan` | Allowed with restriction | |
| `DescribeChannelBan` | Allowed | |
| `ListChannelBan` | Allowed | |
| `UpdateChannelReadMarker` | Allowed with restriction | You need to use CreateChannelMembership to create a membership for yourself first, and then call the API. |

# Member

An `AppInstanceUser` becomes a member of a channel if they are added to the channel via the CreateChannelMembership API.

Members can perform actions only on channels to which they belong.

**Note**
A member who is an `AppInstanceAdmin` or `ChannelModerator` can perform actions on Channels allowed by those two roles.

| API name | Allowed or denied | Notes |
|---|---|---|
| `UpdateChannel` | Denied | |
| `DeleteChannel` | Denied | |
| `DescribeChannel` | Allowed with restriction | You can only get details for public channels. |
| `ListChannel` | Allowed with restriction | You can only get details for public channels. |
| `ListChannelMembershipsForAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `DescribeChannelMembership` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| `ListChannelsModeratedByAppInstanceUser` | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |

| API name | Allowed or denied | Notes |
| --- | --- | --- |
| DescribeChannelModeratedByAppInstanceUser | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| CreateChannelMembership | Allowed with restriction | You can only add other members for an UNRESTRICTED channel. |
| DescribeChannelMembership | Allowed | |
| ListChannelMembership | Allowed | |
| DeleteChannelMembership | Allowed | |
| SendChannelMessage | Allowed | |
| GetChannelMessage | Allowed | |
| ListChannelMessage | Allowed | |
| DeleteChannelMessage | Denied | |
| RedactChannelMessage | Denied | |
| UpdateChannelMessage | Allowed with restriction | You can only update your own messages. |
| CreateChannelModerator | Denied | |
| DeleteChannelModerator | Denied | |
| DescribeChannelModerator | Denied | |
| ListChannelModerator | Denied | |
| CreateChannelBan | Denied | |
| DeleteChannelBan | Denied | |
| DescribeChannelBan | Denied | |
| ListChannelBan | Denied | |
| UpdateChannelReadMarker | Allowed | |

# Non-member

Non-members are a regular AppInstanceUser and they cannot perform any channel related actions unless you use the CreateChannelMembership API to add them.

**Note**
A non-member who is an AppInstanceAdmin or ChannelModerator can perform channel related actions allowed by those two roles.

| API name | Allowed or denied | Notes |
| --- | --- | --- |
| UpdateChannel | Denied | |
| DeleteChannel | Denied | |

| API name | Allowed or denied | Notes |
| --- | --- | --- |
| DescribeChannel | Allowed with restriction | You can only get details for public channels. |
| ListChannel | Allowed with restriction | You can only get details for public channels. |
| ListChannelMembershipsForAppInstanceUser | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| DescribeChannelMembershipForAppInstanceUser | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| ListChannelsModeratedByAppInstanceUser | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| DescribeChannelModeratedByAppInstanceUser | Allowed with restriction | You can only use your ARN as the AppInstanceUserArn value. |
| CreateChannelMembership | Denied | |
| DescribeChannelMembership | Allowed with restriction | You can only get details for public channels. |
| ListChannelMembership | Allowed with restriction | You can only get details for public channels. |
| DeleteChannelMembership | Denied | |
| SendChannelMessage | Denied | |
| GetChannelMessage | Allowed with restriction | You can only get details for public channels. |
| ListChannelMessage | Allowed with restriction | You can only get details for public channels. |
| DeleteChannelMessage | Denied | |
| RedactChannelMessage | Denied | |
| UpdateChannelMessage | Denied | |
| CreateChannelModerator | Denied | |
| DeleteChannelModerator | Denied | |
| DescribeChannelModerator | Denied | |
| ListChannelModerator | Denied | |
| CreateChannelBan | Denied | |
| DeleteChannelBan | Denied | |
| DescribeChannelBan | Denied | |
| ListChannelBan | Denied | |
| UpdateChannelReadMarker | Denied | |

# Streaming export of messaging data

You can configure your AWS Chime Messaging SDK AppInstance to receive data—messages, channel events, etc.—in the form of a stream. Currently, Amazon Chime only accepts Kinesis streams as stream destinations. Here are the following prerequisites for using your Kinesis streams with this feature:

- A stream must be in the same region as the AppInstance.
- Stream names have a prefix that starts with “chime-messaging-”.
- You must configure at least two shards. Each shard can receive data up to 1MB per second, so scale your stream accordingly.
- You must enable server-side encryption (SSE).

### To configure a Kinesis stream

1. Create one or more Kinesis streams using the prerequisites in the previous section.
2. Call the PutAppInstanceStreamingConfigurations API.

   You can configure two app instance data types. You can configure one or both types, and you can choose the same stream or separate streams for them. The data types have the following scopes:

| Event types generated | `AppInstanceDataType` | |
|---|---|---|
| CREATE_CHANNEL_MESSAGE | Channel message | |
| REDACT_CHANNEL_MESSAGE | | |
| UPDATE_CHANNEL_MESSAGE | | |
| DELETE_CHANNEL_MESSAGE | | |
| | | |
| CREATE_CHANNEL | Channel | |
| UPDATE_CHANNEL | | |
| DELETE_CHANNEL | | |
| CREATE_CHANNEL_MEMBERSHIP | | |
| DELETE_CHANNEL_MEMBERSHIP | | |
| CREATE_CHANNEL_BAN | | |
| DELETE_CHANNEL_BAN | | |
| CREATE_CHANNEL_MODERATOR | | |
| DELETE_CHANNEL_MODERATOR | | |

3. Start reading the data from your configured Kinesis stream. Remember, events that occur before the configuration in Step 2 will not be retroactively streamed.

### Data format

Kinesis outputs records in JSON format with the following fields: `EventType` and `Payload`. The payload format depends on the `EventType`. The following table lists the event types and their corresponding payload formats.

| EventType | Payload format | |
|---|---|---|
| CREATE_CHANNEL_MESSAGE | Channel message | |
| REDACT_CHANNEL_MESSAGE | | |
| UPDATE_CHANNEL_MESSAGE | | |
| DELETE_CHANNEL_MESSAGE | | |
| | | |
| CREATE_CHANNEL | Channel | |
| UPDATE_CHANNEL | | |
| DELETE_CHANNEL | | |
| | | |
| CREATE_CHANNEL_MEMBERSHIP | ChannelMembership | |
| DELETE_CHANNEL_MEMBERSHIP | | |
| | | |
| CREATE_CHANNEL_BAN | ChannelBan | |
| DELETE_CHANNEL_BAN | | |
| | | |
| CREATE_CHANNEL_MODERATOR | ChannelModerator | |
| DELETE_CHANNEL_MODERATOR | | |

# Using service-linked roles

Amazon Chime uses AWS Identity and Access Management (IAM) service-linked roles . A service-linked role is a unique type of IAM role that links directly to Amazon Chime. Amazon Chime predefines the service-linked roles, and they include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Chime more efficient, because you aren't required to manually add the necessary permissions. Amazon Chime defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon Chime can assume its roles. The defined permissions include the trust and permissions policies. The permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your Amazon Chime resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM . Look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the documentation for that service.

**Topics in this section**



# Using service-linked roles for data streaming

The following sections explain how to manage the service-linked role for data streaming.

**Topics in this section**



## Service-linked role permissions

Amazon Chime uses the service-linked role named **AmazonChimeServiceChatStreamingAccess**. The role grants access to the AWS services and resources used or managed by Amazon Chime, such as the Kinesis streams used for data streaming.

The **AmazonChimeServiceChatStreamingAccess** service-linked role trusts the following services so that those services can assume the role:

- chime.amazonaws.com

The role permissions policy allows Amazon Chime to complete the following actions on the specified resource:

- `kms:GenerateDataKey` only when the request is made using `kinesis.*.amazonaws.com`.
- `kinesis:PutRecord`, `kinesis:PutRecords`, or `kinesis:DescribeStream` only on streams of the following format: `arn:aws:kinesis:{{region}}:{{account-id}}:stream/chime-messaging-*`.

You must configure permissions to allow an IAM entity such as a user, group, or role to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

## Creating a service-linked role

You don't need to manually create a service-linked role. When you use the PutAppInstanceStreamingConfigurations API to create a data streaming configuration, Amazon Chime creates the service-linked role for you.

You can also use the IAM console to create a service-linked role with the Amazon Chime use case. In the AWS CLI or the AWS API, create a service-linked role with the chime.amazonaws.com service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this role, you can repeat this process to create it again.

## Editing a service-linked role

After you create a service-linked role, you can only edit its description, and you do that using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

## Deleting the resources used by a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
> Deletions can fail if you try to delete resources while Amazon Chime is using them. If a deletion fails, wait a few minutes and try the operation again.

**To delete resources used by the AmazonChimeServiceChatStreamingAccess role**

Turn off the data streaming feature for your app instance by invoking the following API.

- `aws chime delete-app-instance-streaming-configuration --app-instance-arn {APP_INSTANCE_ARN}`

This action deletes all streaming configurations for your app instance.

## Deleting a service-linked role

When you no longer need a feature or service that requires a service-linked role, it's a best practice to delete that role. Otherwise, you have an unused entity that is not actively monitored or maintained. However, you must delete the resources used by your service-linked role before you can manually delete the role.

You can use the IAM console, AWS CLI, or the AWS API to delete the **AmazonChimeServiceChatStreamingAccess** service-linked role. For more information, see Deleting a service-linked role in the IAM User Guide.

# Managing message retention

Account owners can use the Amazon Chime AWS SDK APIs to turn on retention for messaging. Messages are automatically deleted based on the time period set by the administrator. Retention periods can last from one day to 15 years. You can also use the APIs to update message retention periods or turn off message retention at any time.

**Topics in this section**



# Example CLI retention commands

The following examples show typical CLI commands for retention:

**Enabling**

```
aws chime put-app-instance-retention-settings —app-instance-
arn {appInstanceArn} --app-instance-retention-settings
ChannelRetentionSettings={RetentionDays=60}
```

**Updating**

```
aws chime put-app-instance-retention-settings —app-instance-
arn {appInstanceArn} --app-instance-retention-settings
ChannelRetentionSettings={RetentionDays=30}
```

**Disabling**

```
aws chime put-app-instance-retention-settings —app-instance-arn
{appInstanceArn} --app-instance-retention-settings ChannelRetentionSettings={}
```

## Enabling message retention

You use the Amazon Chime AWS SDK APIs to turn on retention for messaging. You can also use the APIs to update message retention periods or turn off message retention at any time. For more information about configuring messaging retention, see the Amazon Chime API Reference.

## Restoring and deleting messages

You can restore messages to users within 30 days of setting or updating a message-retention period. However, after that 30-day grace period, all messages that fall under the retention period are permanently deleted, and new messages are permanently deleted as soon as they pass the retention period.

> **Note**
> During the 30-day grace period, if you the extend the retention policy, or you turn it off, messages that haven't passed the new retention period become visible again to the users in the account.

Messages are also permanently deleted when an `AppInstanceUser` deletes a channel or a message.

# User interface components for messaging

You can use a component library to reduce the effort needed to build the user interface for chat messaging. See Amazon Chime React Component Library on GitHub for more information.

# Integrating with client libraries

To use the messaging features of the Amazon Chime SDK, you must integrate your client application with the following client libraries:

- **AWS SDK** – Contains APIs for sending messages and managing resources.
- **Amazon Chime SDK client library for JavaScript (NPM)** – A JavaScript library with TypeScript type definitions that helps you integrate your client with the Chime messaging web socket to receive messages.

To integrate your client application with the Amazon Chime SDK, see the instructions in the client library README.md, and use the demos to learn how to learn how to build messaging features.

# Using Amazon Chime SDK messaging with JavaScript

You can use JavaScript to manage Amazon Chime SDK resources and send messages. For more information, see the AWS JavaScript SDK.

You can also create a messaging session in your client application to receive messages from the Amazon Chime SDK messaging. For more information, see Using the Amazon Chime SDK for JavaScript on GitHub.

# Using the Amazon Chime SDK for JavaScript

This guide provides a conceptual overview of the Amazon Chime SDK for JavaScript and example code for critical server and client components.

**Topics**



## Components of an Amazon Chime application

To embed real-time audio, video, and screen-sharing capabilities into your Amazon Chime applications, you use these components:

- **The Amazon Chime SDK for JavaScript**, the client-side SDK that you integrate into your browser or Electron web application. You do that by adding the Amazon Chime SDK for JavaScript NPM package as a dependency. This package leverages the MediaDevices and WebRTC APIs to join meetings, exchange audio and video, and share content with other attendees. It gives you a control surface for managing the different types of media and the ability to bind those resources to your application's user interfaces.
- **The AWS SDK**, the Chime API that your server application uses to authenticate and authorize meeting requests from your web application.

  The AWS SDK gives you API actions such as chime:CreateMeeting and chime:CreateAttendee to create and manage meeting and attendee resources. The Amazon Identity and Access Management (IAM) service configures access to these actions.

  The AWS SDK is available in several programming languages and takes the complexity out of calling the Amazon Chime SDK for JavaScript APIs from your server application. If your application doesn't currently use a server, you can start with the AWS CloudFormation template included in the demos/serverless folder. That demo shows you how to build an AWS Lambda-based serverless application that uses the AWS SDK Chime API.
- **Amazon Chime media services** provides the audio, video, and signaling that the Amazon Chime SDK for JavaScript uses to connect to meetings. Media services are available globally in large number of AWS regions, and they support audio mixing, video forwarding, and NAT traversal using TURN relays. The Amazon Chime service team deploys, monitors, and manages these services. The media services are hosted in a single range of IP addresses – 99.77.128.0/18 – and use ports TCP/443 and UDP/3478 to simplify firewall configurations for IT administrators. Finally, these services leverage the AWS Global Cloud Infrastructure.

## Key concepts

To fully understand how to create and manage meetings and users, you need to understand these concepts:

**Meeting** – A multi-party media session. Every meeting has a unique meeting identifier, and you create meetings in one of the supported AWS regions. When you create a meeting, the APIs return a list of media URLs. That's a key part of the data needed to join a meeting, and you need to disseminate that data to all users trying to join the meeting.

**Attendee** – A user trying to join a multi-party media session. Every attendee has a unique identifier, an external user identifier that you can pass in to map the attendee to a user in your system, and a signed Join Token that grants the attendee access to the meeting.

**MeetingSession** (DefaultMeetingSession) – The root object of the Amazon Chime SDK for JavaScript that represents each user's session in a meeting. The web applications start by instantiating MeetingSession and configuring it with the correct meeting and attendee information.

**MeetingSessionConfiguration** – Stores the meeting and attendee data needed to join a meeting session. The data comes as the response of the CreateMeeting and CreateAttendee API calls made by the server application. This data is passed down from the server application to the web application and used to instantiate the MeetingSession.

**DeviceController** (DefaultDeviceController) – Enables you to enumerate the list of available audio and video devices on a user's system. You can also use the device controller during a meeting to switch the actively used devices.

**AudioVideoFacade** (DefaultAudioVideoFacade) – The key interface that powers a meeting. It provides APIs to start, control, and leave a meeting. It also provides APIs to listen for the key events that drive user experience changes, such as a roster of attendees, by tracking users joining or leaving, being muted or unmuted, actively speaking, or having poor connectivity. You can also use these APIs to bind the audio control HTML element to the meeting audio output and play it through the selected audio output device.

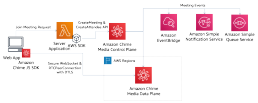
**ActiveSpeakerDetectorFacade** (DefaultActiveSpeakerDetector) – Provides the API that subscribes to active speaker events. Periodically returns a list of attendees ranked by their mic volume over a period of time. You can override the active speaker policy as needed.

**ContentShareController** (DefaultContentShareController) – Provides APIs to start-stop and pause-unpause the content sharing experience. It also provides APIs for listening to lifecycle events that track content sharing status.

**Logger** (ConsoleLogger) – The interface used to leverage the console logging, or pass in a logger object to override the logging implementation. The latter approach gives you different levels of logs from the SDK.

# Service architecture

The following high-level architecture diagram shows how these components interact and work with other AWS services:



# Web application architecture

The web application may be served from a content delivery network and loaded when the user navigates to a URL in a browser. It may also be wrapped in a platform-native Electron application that users install on their machines. The application makes REST requests to the server application to join a new or existing meeting. Typically, this request should carry an authorization token or a cookie that your application uses for other API requests. You can also design your web application to send a region

hint to the server application, which the latter may use when providing the MediaRegion parameter to chime:CreateMeeting. Your web application can determine the closest Amazon Chime media services region by making an HTTP GET request to the https://nearest-media-region.l.chime.aws endpoint.

## Server application architecture

When a server application receives a request from a client, it first authenticates the user and ensures that the user is authorized to start a new meeting or join an existing meeting. The server uses the embedded AWS SDK in the language of choice to make chime:CreateMeeting and chime:CreateAttendee API calls to the global Amazon Chime media control plane to create the meeting, and the attendees, in one of the supported AWS regions. To make these requests, the service needs the appropriate IAM user or role with the AmazonChimeSDK policy.

## The Amazon Chime media control plane

The Amazon Chime media control plane is global – hosted out of us-east-1 – and serves the CreateMeeting and CreateAttendee APIs to create and manage the meeting and attendee resources across the data plane. It validates credentials and ensures the session is bootstrapped in the data plane in the requested region. The control plane also triggers Amazon Chime SDK Events to the notification mechanisms such as Amazon EventBridge, Amazon Simple Queueing Service (SQS) or Amazon Simple Notification Service (SNS). The services are constantly monitored and designed to scale automatically as load increases. The APIs are designed to only accept opaque user identifiers and not user data, so they adhere to data sovereignty requirements.

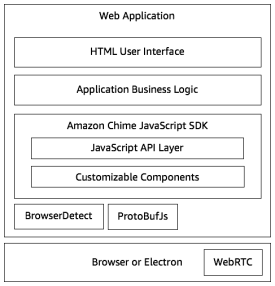
## The Amazon Chime media data plane

The media data plane includes an audio mixing service, a video forwarding service, a TURN service, and Session Initiation Protocol (SIP) interoperability services. AWS continuously monitors the services, and they scale automatically as load increases.

You can create meetings in any media region from any control plane region. To learn more, see Amazon Chime SDK media Regions .

# Web application component architecture

This diagram shows the architecture of an Amazon Chime web client application:




A web application typically consists of an HTML and CSS based user interface layer powered by the application business logic layer. You can build the web application in plain HTML and JavaScript, or you can use UI frameworks such as React and Angular.

The web application's business logic layer interacts with the Amazon Chime SDK for JavaScript through a set of JavaScript APIs exposed by the SDK. The DefaultMeetingSession is the root object of the SDK, and

you use MeetingSessionConfiguration to initialize it with meeting and attendee information from the server application and join the meeting. The DefaultMeetingSession also exposes the AudioVideoFacade, which enables the business logic layer to take actions, and to register for callbacks that update the user interface when the underlying state of the session changes.

The Amazon Chime SDK for JavaScript is open-source and has a set of customizable components that you can override as needed. The default implementations allow you to build a complete unified communications application such as our demo MeetingV2 application. The Amazon Chime SDK for JavaScript depends on two other libraries:

- Browser-Detect for identifying the browser type and capabilities.
- ProtoBufJs to encode and decode signaling commands and responses needed to join a media sessions.

The The SDK also depends on the Browser or Electron application to provide the Device Management APIs and the WebRTC implementation for an audio video session.

The source Amazon Chime SDK for JavaScript is in TypeScript but you can use the TypeScript compiler to compile it to Javascript, and then bundle it using a module bundler such as Webpack. As a best practice, install the Amazon Chime SDK for JavaScript from the NPM registry and use it in a CommonJS environment. We have also included a Rollup script to bundle the Chime SDK into a minified JS file for you in case you want to directly include it as a script tag in your html file.

# Building a server application

The information in the following section explains how to build an Amazon Chime server application. Each section provides example code as needed.

## Creating IAM users or roles with the Chime SDK policy

You create users as IAM users, or in roles as appropriate to your use case. You then assign the following policy to them. This ensures that you have the necessary permissions for the AWS SDK embedded in your server application. That allows you to perform lifecycle operations on the meeting and attendee resources.

```
// Policy ARN:     arn:aws:iam::aws:policy/AmazonChimeSDK
// Description:    Provides access to Amazon Chime SDK operations
{
"Version": "2012-10-17",
"Statement": [
    {
        "Action": [
            "chime:CreateMeeting",
            "chime:DeleteMeeting",
            "chime:GetMeeting",
            "chime:ListMeetings",
            "chime:CreateAttendee",
            "chime:BatchCreateAttendee",
            "chime:DeleteAttendee",
            "chime:GetAttendee",
            "chime:ListAttendees"
        ],
        "Effect": "Allow",
        "Resource": "*"
```

```
    }
]}
```

# Configuring the AWS SDK to invoke the APIs

```
AWS.config.credentials = new AWS.Credentials(accessKeyId, secretAccessKey, null);
const chime = new AWS.Chime({ region: 'us-east-1' });
chime.endpoint = new AWS.Endpoint('https://service.chime.aws.amazon.com/console');
```

# Creating a meeting

You use the CreateMeeting and CreateMeetingWithAttendee API calls to create meetings. The calls have one required parameter, ClientRequestToken. When two or more users try to create the same meeting, the parameter ensures that Chime only creates one meeting. You need to pass in a meeting id, or a unique hash of the id. The Chime service handles the race condition of simultaneous calls — any combination of CreateMeeting and CreateMeetingWithAttendee — and guarantees that only one meeting exists for the given token.

The APIs also accept optional parameters such as MediaRegion, which represents the media services data plane region to chose for the meeting, the MeetingHostId used to pass in an opaque identifier to represent the meeting host, if applicable, and the NotificationsConfiguration for receiving meeting lifecycle events. By default,Amazon EventBridge delivers the events. Events are emitted on a best-effort basis for Amazon Chime SDK events.

Optionally, you can also receive events by passing an SQS queue ARN or an SNS Topic ARN in NotificationsConfiguration. The API Returns a Meeting object that contains a unique MeetingId, the MediaRegion and the MediaPlacement object with a set of media URLs.

```
meeting = await chime.createMeeting({
            ClientRequestToken: clientRequestToken,
            MediaRegion: mediaRegion,
            MeetingHostId: meetingHostId,
            NotificationsConfiguration: {
               SqsQueueArn: sqsQueueArn,
               SnsTopicArn: snsTopicArn
            }
        }).promise();
```

# Creating an attendee

After you create a meeting, you create an attendee resource that represents each user trying to join the media session. The CreateAttendee API takes in the following:

- The MeetingId of the meeting to which you're adding the user.
- An ExternalUserId, which can be any opaque user identifier from your identity system.

For example, if you use Active Directory (AD), this can be the Object Id of the user in the AD. The ExternalUserId is valuable as it is passed back to the client applications when they receive attendee events from the client SDKs. This allows the client application to know who joined or left the meeting

and retrieve additional information from the server application about that user, such as a display name, email, or a picture.

The response to the CreateAttendee API is an Attendee object. The object contains a unique AttendeeId that is generated by the service, the ExternalUserId that was passed in, and a signed JoinToken that allows the attendee to access the meeting for its duration, or until the DeleteAttendee API deletes the attendee.

```
   attendee = await chime.createAttendee({
               MeetingId: meeting.MeetingId,
               ExternalUserId: externalUserId,
             }).promise();
```

## Sending a response to the client application

Once you create the meeting and attendee resources, the server application should encode and send the Meeting and Attendee objects back to the client application. The client needs those pieces of information to bootstrap the Amazon Chime SDK for JavaScript, and enable an attendee to join the meeting successfully from a web or electron based application.

# Building a client application

The information in the following sections explains how to build an Amazon Chime client application. Each section provides example code as needed.

## Initialize the meeting session

Start by initializing the meeting session configuration object with the results of the CreateMeeting and CreateAttendee API calls passed down from the server application. You can also pass in a logger and device controller of your choice, or use the console logger and the default device controller implementation. Finally, create a new instance of the DefaultMeetingSession and access the key AudioVideoFacade object.

```
 async initializeMeetingSession(configuration: MeetingSessionConfiguration): Promise
<void> {
   const logger = new ConsoleLogger('SDK', LogLevel.DEBUG);
   const deviceController = new DefaultDeviceController(logger);
   configuration.enableWebAudio = this.enableWebAudio;
   this.meetingSession = new
DefaultMeetingSession(configuration, logger, deviceController);
   this.audioVideo = this.meetingSession.audioVideo;
   ...
 }
```

## Registering for lifecycle callbacks

The next step is to register for the lifecycle events of the AudioVideoFacade and the ContentShareController. This drives the overall state machine of the meeting window and tells you when to transition from the connecting to the connected state, and from the connected to the disconnected state.

```
      this.audioVideo.addObserver(this);
      this.meetingSession.contentShare.addContentShareObserver(this);
```

# Handling device permissions

Next, trigger the device permissions, especially for web applications. This ensures that the user has granted the permissions needed to access the audio and video capture devices. You can do this by using the navigator.mediaDevices.getUserMedia API provided by the web browsers. This API will block your app until the user accepts or dismisses the permissions dialog box.

```
   this.audioVideo.setDeviceLabelTrigger(
     async (): Promise>MediaStream< => {
       this.switchToFlow('flow-need-permission');
       const stream = await navigator.mediaDevices.getUserMedia({ audio: true, video:
true });
       this.switchToFlow('flow-devices');
       return stream;
     }
   );
```

# Set up devices and join audio-video

Once the user grants permissions to devices, we choose the capture and render devices that the user selected for audio, then invoke the AudioVideoFacade.start API to trigger the meeting join.

```
   await this.audioVideo.chooseAudioInputDevice(audioInputDevice);
   await this.audioVideo.chooseAudioOutputDevice(audioOutputDevice);
   this.audioVideo.start();
```

# Registering device controller callbacks

Now is a good time to register for device-change callbacks. This tracks devices being added or removed from the system.

```
      this.audioVideo.addDeviceChangeObserver(this);
```

# Registering lifecycle events

The *audioVideoDidStartConnecting* callback is triggered first, indicating that we can transition to a connecting screen. Once we establish an audio-video session, we transition to a connected state when we receive the *audioVideoDidStart* callback. If it fails, it triggers the *audioVideoDidStop* callback with appropriate *sessionStatus* information to describe the failure reasons.

```
  audioVideoDidStartConnecting(reconnecting: boolean): void {...}
```

```
   audioVideoDidStart(): void {...}
   audioVideoDidStop(sessionStatus: MeetingSessionStatus): void {...}
```

# Registering real time callbacks

Once the user successfully connects to the meeting for audio and video, we can now register for realtime callbacks to track attendee presence, mute status, signal strength change, and fatal error state. Real time callbacks are received very frequently. Be careful about processing these callbacks because they can slow performance for users.

```
  this.audioVideo.realtimeSubscribeToAttendeeIdPresence(attendeePresenceHandler);
  this.audioVideo.realtimeSubscribeToMuteAndUnmuteLocalAudio(localAudioMuteStateHandler);
  this.audioVideo.realtimeSubscribeToSetCanUnmuteLocalAudio(canUnmuteHandler);

 this.audioVideo.realtimeSubscribeToLocalSignalStrengthChange(localSignalStrengthChangeHandler);
  this.audioVideo.realtimeSubscribeToFatalError(fatalErrorHandler);
```

# Subscribing to remote attendee callbacks when an attendee joins

When an attendee joins the meeting, we receive a callback to the attendeePresenceHandler. Within this handler we register the volume indicator callbacks for that attendee. This can drive the mic activity indicator for each attendee in the roster so users know who is speaking at any time.

```
attendeePresenceHandler(attendeeId: string, present: boolean): void => {
    this.audioVideo.realtimeSubscribeToVolumeIndicator(
        attendeeId,
        async (
          attendeeId: string,
          volume: number | null,
          muted: boolean | null,
          signalStrength: number | null
        ) => {
            // Handle volume strength changed event
        });
    ...
}
```

# *videoTileDidUpdate* callback and binding video tiles

When users turn on video, you receive the *videoTileDidUpdate* callback with a *tileState* object. The *tileState* object has all information about the user's video stream, including: tileId, if the tile is local, if the tile is a content tile, if it's active or paused, size information, if it is bound to a video element, and so on. The application has to manage the tiles as they get added, and bind them to HTML Video Elements using the *bindVideoElement* API of the AudioVideoFacade.

```
videoTileDidUpdate(tileState: VideoTileState): void {
    this.log(`video tile updated: ${JSON.stringify(tileState, null, '  ')}`);
    if (!tileState.boundAttendeeId) {
      return;
```

```
    }
    const selfAttendeeId = this.meetingSession.configuration.credentials.attendeeId;
    const modality = new DefaultModality(tileState.boundAttendeeId);
    if (modality.base() === selfAttendeeId &&
modality.hasModality(DefaultModality.MODALITY_CONTENT)) {
      // don't bind one's own content
      return;
    }
    const tileIndex = tileState.localTile
      ? 16
      : this.tileOrganizer.acquireTileIndex(tileState.tileId);
    const tileElement = document.getElementById(`tile-${tileIndex}`) as HTMLDivElement;
    const videoElement = document.getElementById(`video-${tileIndex}`) as HTMLVideoElement;
    const nameplateElement = document.getElementById(`nameplate-${tileIndex}`) as
HTMLDivElement;

    const pauseButtonElement = document.getElementById(`video-pause-${tileIndex}`) as
HTMLButtonElement;
    const resumeButtonElement = document.getElementById(`video-resume-${tileIndex}`) as
HTMLButtonElement;

    pauseButtonElement.addEventListener('click', () => {
      if (!tileState.paused) {
        this.audioVideo.pauseVideoTile(tileState.tileId);
      }
    });

    resumeButtonElement.addEventListener('click', () => {
      if (tileState.paused) {
        this.audioVideo.unpauseVideoTile(tileState.tileId);
      }
    });

    this.log(`binding video tile ${tileState.tileId} to ${videoElement.id}`);
    this.audioVideo.bindVideoElement(tileState.tileId, videoElement);
    this.tileIndexToTileId[tileIndex] = tileState.tileId;
    this.tileIdToTileIndex[tileState.tileId] = tileIndex;
    // TODO: enforce roster names
    new TimeoutScheduler(2000).start(() => {
      const rosterName = this.roster[tileState.boundAttendeeId]
        ? this.roster[tileState.boundAttendeeId].name
        : '';
      if (nameplateElement.innerHTML !== rosterName) {
        nameplateElement.innerHTML = rosterName;
      }
    });
    tileElement.style.display = 'block';
    this.layoutVideoTiles();
  }
```

# Starting local video

When users turn on video, you first use the *chooseVideoInputDevice* API set the chosen input device. You then invoke the *startLocalVideoTile* API to start sending video. You can use the *chooseVideoInputQuality* API to set the correct parameters for the outbound video stream, such as width, height, framerate, and maximum bandwidth, which controls the resolution of the stream, up to 720p.

```
await this.audioVideo.chooseVideoInputDevice(device);
this.audioVideo.startLocalVideoTile();
```

# Using Amazon Voice Focus

Amazon Voice Focus is a noise suppressor that uses machine learning to reduce unwanted background noise in your users' microphone input. Unlike conventional noise suppressors, Amazon Voice Focus reduces fan noise, road noise, typing, rustling paper, lawnmowers, barking dogs, and other kinds of non-speech input, allowing your users to focus on the human voice.

> **Note**
> Amazon Voice Focus doesn't eliminate those types of noises; it reduces their sound levels. To ensure privacy during a meeting, choose **Mute** to silence yourself or others.

Amazon Voice Focus offers multiple complexity levels, allowing you to trade some quality in order to support a wider range of devices. By default, the Amazon Chime SDK automatically choose the right complexity level at runtime based on device performance.

**When to use Amazon Voice Focus**

Use Amazon Voice Focus when users might experience background noise and they only care about human speech. Because it reduces almost all non-voice sounds, Amazon Voice Focus works best in applications where a person's voice is the most important part of an interaction. In quiet environments, or in situations where other sounds are important, such as music lessons, Amazon Voice Focus can eliminate important audio.

Also, Amazon Voice Focus can suppress quiet speech. For example, if you have several participants in a room with a single laptop, Amazon Voice focus can suppress the quietest participants.

Finally, applications that use a lot of CPU, such as games, might not leave enough resources for Amazon Voice Focus to work smoothly, especially on resource-constrained devices. If you think your application might be used in those scenarios, be sure to test your application with Amazon Voice Focus, and give users the ability to turn noise suppression on and off.

**Amazon Voice Focus stages**

Amazon Voice Focus steps through these stages when users turn it on:

- **Configuration and estimation** – For a given builder intent and runtime environment, decide on runtime parameters. Amazon Voice Focus checks for browser support and CPU speed to determine a complexity level for the user's browser.
- **Preloading and initialization** – Model files have an upper bound of 8MB, and they're downloaded as needed from Amazon's Content Delivery Network. If possible, initialize Amazon Voice Focus prior to the start of a meeting or conference.
- **Interference in `getUserMedia`** – Amazon Voice Focus works best if no other noise suppression applies to a microphone input. By default, Amazon Voice Focus disables a web browser's built-in noise suppressor for you.
- **Application to the chosen audio stream** – The SDK device controller applies noise suppression to the microphone input before use.

For detailed information about implementing Amazon Voice Focus, see https://aws.github.io/amazon-chime-sdk-js/modules/amazonvoice_focus.html .

# Starting content sharing

You use content sharing features to share an entire screen, application, or browser tab. You can trigger sharing with the *startContentShareFromScreenCapture* API. This triggers the necessary browser-specific screen sharing selector experience. It also shares content into the meeting as a video stream.

```
    this.meetingSession.screenShare.startContentShareFromScreenCapture();
```

# Viewing content sharing

Users see all the content shared in a meeting. It appears as a new *VideoTile*, which triggers the *videoTileDidUpdate* callback. The *TileState* for this tile says it's a content tile. You can render these tiles in a larger HTML Video element.

# Tracking metrics and connection status

The Amazon Chime SDK for JavaScript triggers the metricsDidReceive event to provide periodic callbacks for the application to track key WebRTC metrics. The SDK also provides callbacks for tracking a session's health and connection status:

- *connectionHealthDidChange*
- *connectionDidBecomePoor*
- *connectionDidSuggestStopVideo*
- *videoSendDidBecomeUnavailable*
- *videoSendHealthDidChange*
- *videoSendBandwidthDidChange*
- *videoReceiveBandwidthDidChange*

# Tearing down a session

You start tearing down a session by calling stop on the AudioVideoFacade. If the user wants to kick everyone out of the meeting and terminate the session completely, you need an HTTP call to the Server Application – with the same authentication token – to trigger the DeleteMeeting API on the media control plane. That ensures the meeting is terminated and all users are disconnected.

```
    this.audioVideo.stop();
```

# Using the Amazon Chime SDK for Android

Currently, you'll find the Amazon Chime SDK for Android on GitHub. Go to https://github.com/aws/amazon-chime-sdk-android .

# Using the Amazon Chime SDK for iOS

Currently, you'll find the Amazon Chime SDK for iOS on GitHub. Go to https://github.com/aws/amazon-chime-sdk-ios .

# Building Lambda functions for SIP media applications

With this SDK, developers can use AWS Lambda to build functions for Amazon Chime SIP media applications. In turn, Amazon Chime administrators use the Lambda functions when they create SIP media applications. Your Lambda functions control the behavior of phone requests. You can specify different signaling and media actions for each request. The following topics provide a detailed overview and architectural information about SIP applications, plus example JSON responses with actions that you can return from functions created with AWS Lambda.

> **Note**
> The topics in this section assume that you use AWS Lambda. For more information about AWS Lambda, see Getting started with AWS Lambda.

**Topics**



## Using Lambda functions for SIP applications

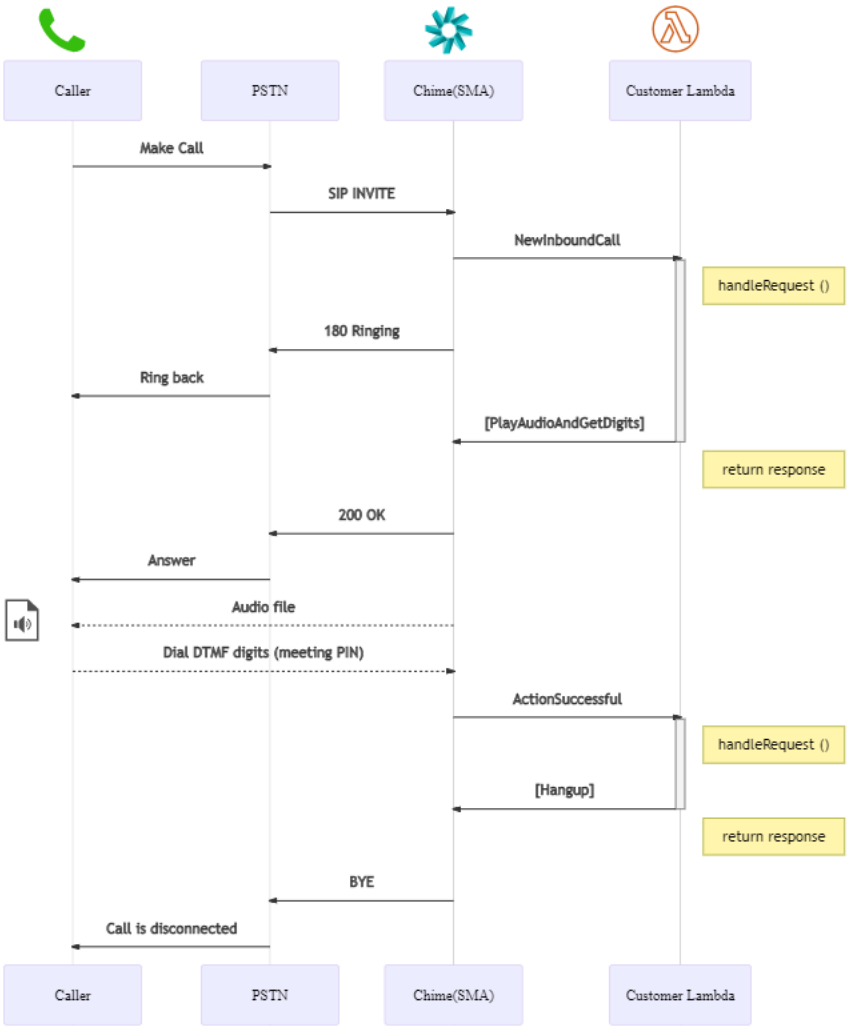
Amazon Chime uses functions created with AWS Lambda for the following reasons:

- You can add logic to your SIP applications. For example, you can make decisions based on the output of certain actions, such as an SDK meeting lookup based on a number from a touch-tone phone, or based on call properties such as a **From** header.
- They save time. You can write lightweight, serverless applications that don't need a server infrastructure.

When you create a SIP application, you choose an AWS Region. The application can only use Lambda functions set for that same Region.

## Basic call flow

This diagram shows the basic flow of a call through a SIP application and its associated Lambda functions. In this case, the application plays a prompt, gathers dual-tone multi frequency numbers (DTMF), and then ends the call.

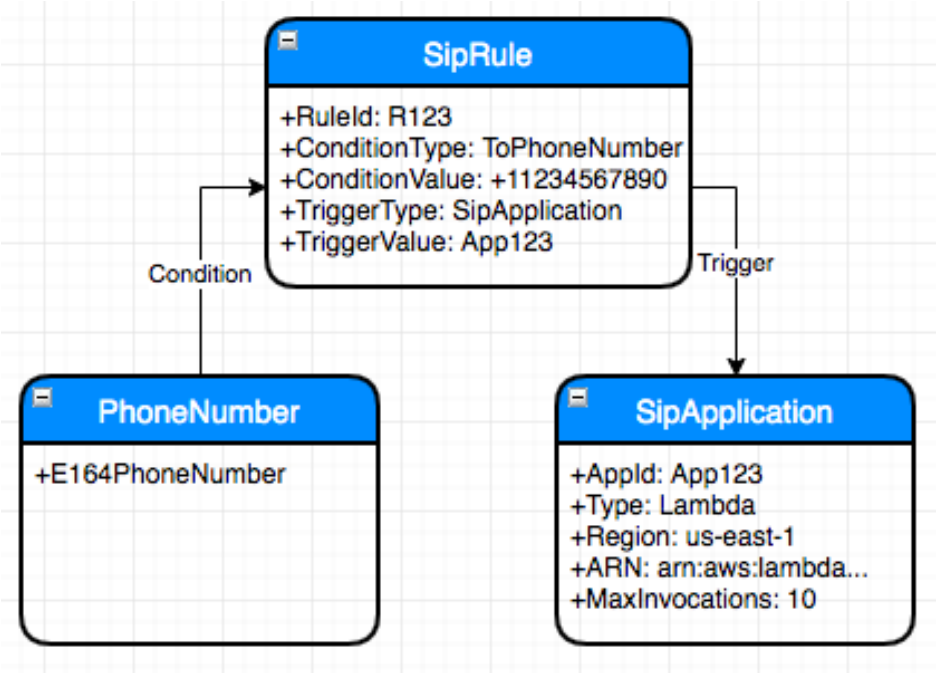



In the diagram:

- When users call, the SIP rule associated with the phone number triggers a SIP media application. In turn, the application invokes Lambda functions that handle the incoming call and return a response. In this case, an audio file asks the users to enter a meeting PIN.
- After the users enter the PIN, they join the meeting.
- When the users hang up. The SIP media application invokes the Lambda functions that end the call.

# Adding invocation rules for incoming invitations

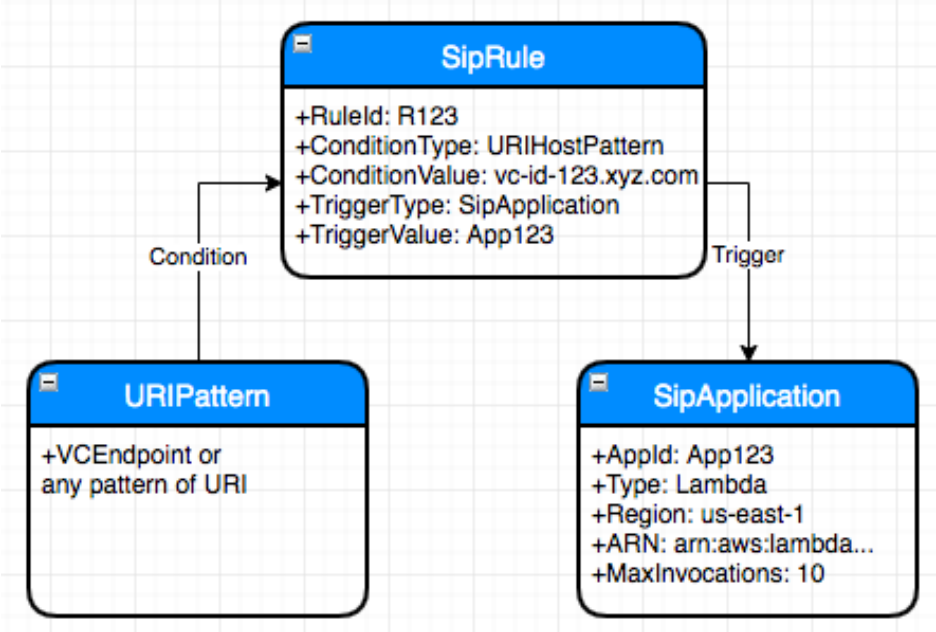
Amazon Chime SIP applications use the concept of a *rule*. Rules control whether the SIP application connects to a phone number or a Request URI hostname. Rules are similar to a Public Branch Exchange or Session Border Controller (PBX/SBC) dial plan. The following diagram shows a typical invocation rule that uses a phone number.




In the diagram:

- The user dials a phone number.
- The rule associated with the phone number triggers a specific SIP application. In turn, the ARN invokes Lambda functions that trigger actions, such as playing a welcome message or accepting a meeting PIN.

The following diagram shows a typical rule that uses a Request URI hostname.




In the diagram:

- The user dials a phone number associated with an Amazon Chime Voice Connector.

- The rule associated with the Request URI hostname triggers a specific SIP media application. In turn, the application invokes Lambda functions that trigger actions, such as playing a welcome message or accepting a meeting PIN.

# SIP application call architecture

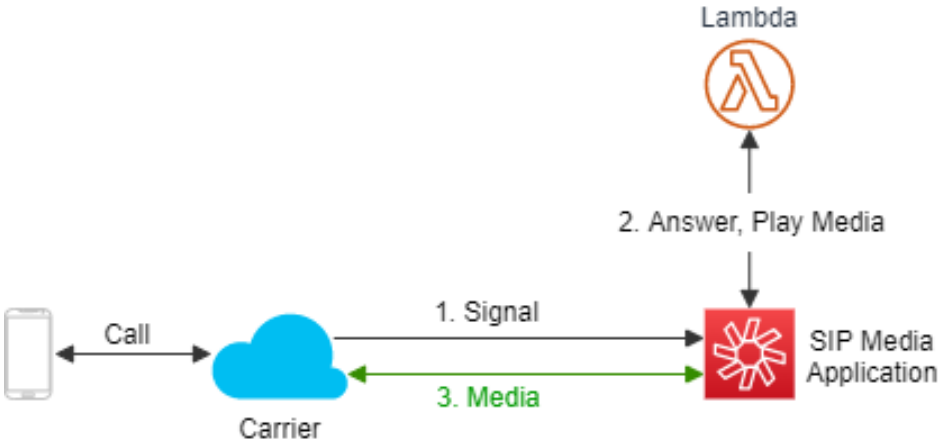
Amazon Chime SIP applications handle both one-legged and two-legged calls. A *one-legged call* can be outbound or inbound, but never both. A *two-legged* call occurs when you connect an inbound and an outbound leg, or two outbound legs. For more details, see the following topics.

**Contents**



## One-legged call architecture

Each of your SIP applications invokes a Lambda function. In turn, your Lambda functions can specify different types of flow actions. Actions that don't create an outbound leg, such as playing audio or receiving digits, constitute a one-legged call. The following diagram shows the architecture of a one-legged call.

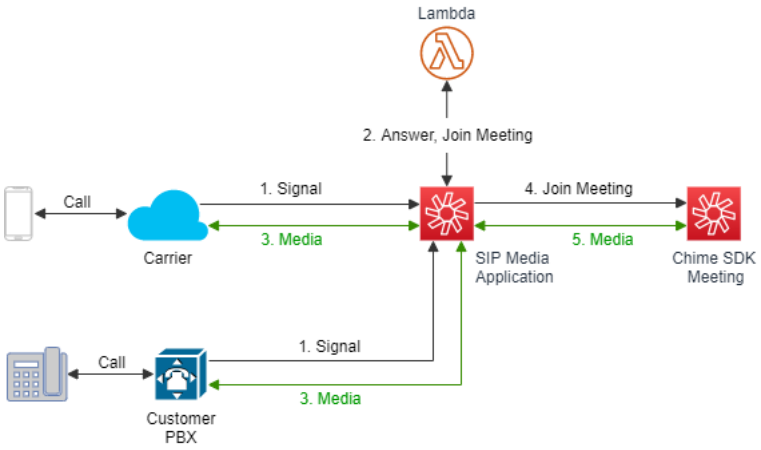



Numbers in the following text correspond to numbers in the diagram and describe the call flow.

1. A user call signals the SIP media application.
2. The application invokes a set of Lambda functions that answer the call and play a media file, such as a welcome message or a request for a PIN.
3. The SIP application routes media to the user.

## Two-legged call architecture

When a Lambda function specifies the `JoinChimeMeeting` action for an incoming SIP request from a customer's PBX/SBC or a phone call, Amazon Chime adds a second leg to the Amazon Chime meeting. The result is a two-legged call. The following diagram shows the architecture of a two-legged, or *bridged*, call.




Numbers in the following text correspond to numbers in the diagram and describe the call flow.

1. When the user calls from a private branch exchange (PBX) or public carrier, the call signals the SIP media application.
2. The application invokes a set of Lambda functions that answer the call and play a media file, such as a welcome message or a request for a PIN.
3. The application routes the media to the user, who then responds.
4. The SIP application invokes the `JoinChimeMeeting` action, which connects the user to the meeting and bridges the call.
5. The user sends and receives audio and video during the meeting.

# End-to-end use case

This use case provides example code for receiving a phone call from an attendee, greeting the attendee with an audio message, getting the meeting PIN from the attendee, validating that PIN, playing audio, and joining the meeting.

> **Note**
> To implement this use case, you need at least one private phone number, a SIP application that uses a Lambda function with an Amazon Resource Name (ARN), and a rule that uses the phone number as its trigger.

When Amazon Chime receives a call on the phone number specified in a rule, the SIP application invokes a Lambda function with the `NEW_INBOUND_CALL` invocation event type.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 1,
    "InvocationEventType": "NEW_INBOUND_CALL",
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+11234567890",
```

```
                "From": "+19876543210",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
```

The Lambda function validates the call details and stores them as needed for future use. For a `NEW_INBOUND_CALL` event, the function responds with a set of actions that play a welcome prompt and ask for the meeting PIN.

```
{
    "SchemaVersion": "1.0",
    "Actions": [
        {
            "Type" : "PlayAudio",
            "Parameters" : {
                "CallId": "call-id-1",
                "
                "AudioSource": {
                    "Type": "S3",
                    "BucketName": "chime-meetings-audio-files-bucket-name",
                    "Key": "welcome-to-meetings.wav"
                }
            }
        },
        {
            "Type": "PlayAudioAndGetDigits",
            "Parameters" : {
                "ParticipantTag": "LEG-A",

                "AudioSource": {
                    "Type": "S3",
                    "BucketName": "chime-meetings-audio-files-bucket-name",
                    "Key": "enter-meeting-pin.wav"
                },
                "FailureAudioSource": {
                    "Type": "S3",
                    "BucketName": "chime-meetings-audio-files-bucket-name",
                    "Key": "invalid-meeting-pin.wav"
                },
                "MinNumberOfDigits": 3,
                "MaxNumberOfDigits": 5,
                "TerminatorDigits": ["#"],
                "InBetweenDigitsDurationInMilliseconds": 5000,
                "Repeat": 3,
                "RepeatDurationInMilliseconds": 10000
            }
        }
    ]
}
```

The SIP application runs these actions on call leg A. Assuming the `PlayAudioAndGetDigits` action receives the digits, the SIP application invokes the Lambda function with the `ACTION_SUCCESSFUL` event.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 2,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
```

```
        "Type": "PlayAudioAndGetDigits",
        "Parameters" : {
            "ParticipantTag": "LEG-A",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "chime-meetings-audio-files-bucket-name",
                "Key": "enter-meeting-pin.wav"
            },
            "FailureAudioSource": {
                "Type": "S3",
                "BucketName": "chime-meetings-audio-files-bucket-name",
                "Key": "invalid-meeting-pin.wav"
            },
            "MinNumberOfDigits": 3,
            "MaxNumberOfDigits": 5,
            "TerminatorDigits": ["#"],
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "Repeat": 3,
            "RepeatDurationInMilliseconds": 10000
        },
        "ReceivedDigits": "12345" // meeting PIN
    },
    "CallDetails": {
        ... // same as in previous event
    }
}
}
```

The Lambda function uses the `CallDetails` data to identify the incoming caller. You can validate the meeting PIN received earlier. Assuming a correct PIN, you then use the Amazon Chime AWS SDK CreateMeeting and CreateAttendee APIs to create the meeting and generate the attendee token. The Lambda function responds with the action to join the Amazon Chime meeting.

```
{
    "SchemaVersion": "1.0",
    "Actions": [
        {
            "Type": "JoinChimeMeeting",
            "Parameters": {
                "JoinToken": "meeting-attendee-join-token"
            }
        }
    ]
}
```

Assuming the `JoinToken` is valid, the SIP application joins the Amazon Chime meeting and invokes a Lambda function with the `ACTION_SUCCESSFUL` event, where `CallDetails` contains the `LEG-B` information.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 3,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type" : "JoinChimeMeeting",
        "Parameters" : {
            "JoinToken": "meeting-attendee-join-token"
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
```

```
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+11234567890",
                "From": "+19876543210",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            },
            {
                "CallId": "call-id-2",
                "ParticipantTag": "LEG-B",
                "To": "SMA",
                "From": "+17035550122",
                "Direction": "Outbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
```

If you want to stop running actions on the call or call leg at this point, you can respond with an empty set of actions.

```
{
    "SchemaVersion": "1.0"
    "Actions": []
}
```

After the caller hangs up, the SIP application invokes the Lambda function with the `HANGUP` event.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 4,
    "InvocationEventType": "HANGUP",
    "ActionData": {
        "Type": "Hangup",
        "Parameters": {
            "CallId": "call-id-1",
            "ParticipantTag": "LEG-A"
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+11234567890",
                "From": "+19876543210",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Disconnected"
            },
```

```
            {
                "CallId": "call-id-2",
                "ParticipantTag": "LEG-B",
                "To": "SMA",
                "From": "+17035550122",
                "Direction": "Outbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Disconnected"
            }
        ]
    }
}
```

If both participants are disconnected, the SIP application ignores the Lambda response.

# Building the Lambda functions for SIP applications

The topics in this section explain how to build the Lambda functions used by your SIP applications.

**Contents**



## Invoking Lambda functions with SIP applications

When a call comes on your phone number, or from a PBX on your Amazon Chime Voice Connector, Amazon Chime invokes the SIP media application's Lambda function with the call details. SIP applications invoke their Lambda functions in different cases. Each invocation of a Lambda function specifies an invocation event type and provides the call details, including its participants, if applicable. The following topics describe the cases for which SIP applications invoke Lambda functions.

### Receiving an inbound call

When a new inbound call is received, the SIP application invokes a Lambda function with this payload.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 2,
    "InvocationEventType": "NEW_INBOUND_CALL"
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
```

```
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
```

## Getting next actions

When a SIP application successfully runs the last action in a list that you provide, the application calls a Lambda function to get the next set of actions. The SIP application also invokes a Lambda function when it successfully runs actions such as `PlayAudioAndGetDigits`. You use `ActionData` to identify which call invoked the function. The following code example shows a sample payload for the `ACTION_SUCCESSFUL` invocation event type after a `PlayAudioAndGetDigits` action.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 3,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type": "PlayAudioAndGetDigits",
        "Parameters" : {
            "CallId": "call-id-1",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "bucket-name",
                "Key": "failure-audio-file.wav"
            },
            "FailureAudioSource": {
                "Type": "S3",
                "BucketName": "bucket-name",
                "Key": "failure-audio-file.wav"
            },
            "MinNumberOfDigits": 3,
            "MaxNumberOfDigits": 5,
            "TerminatorDigits": ["#"],
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "Repeat": 3,
            "RepeatDurationInMilliseconds": 10000
        },
        "ReceivedDigits": "123"
    }
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
}
```

## Failing actions

When any action in the list fails to run, the SIP application invokes a Lambda function to notify you of the failure and get a new set of actions to run on that call. The following code example shows the sample payload for the `ACTION_FAILED` invocation event type after a `PlayAudio` action.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 4,
    "InvocationEventType": "ACTION_FAILED",
    "ActionData": {
        "Type": "PlayAudio",
        "Parameters" : {
            "CallId": "call-id-1",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "audio-file.wav"
            }
        },
        "ErrorType": "InvalidAudioSource",
        "ErrorMessage": "Audio Source parameter value is invalid."
    }
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
}
```

## Capturing digits

You use the `ReceiveDigits` action to receive inbound dual-tone multi frequency (DTMF) digits and symbols. When the SIP application receives one or more DTMF digits, it invokes a Lambda function with the `ReceivedDigits` value contained in the `ActionData` object.

When a Lambda function first returns a `ReceiveDigits` action, and that action runs successfully, it gets back a function similar to this example:

```
{
    "SchemaVersion": "1.0",
    "Sequence": 4,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "ReceivedDigits": "",
        "Type": "ReceiveDigits",
        "Parameters": {
```

```
            "CallId": "call-id-1",
            "InputDigitsRegex": "^\d{2}#$",
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "FlushDigitsDurationInMilliseconds": 10000
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
```

Once the caller enters digits that match the regex pattern, the SIP media application invokes a Lambda function with this type of payload:

```
{
    "SchemaVersion": "1.0",
    "Sequence": 5,
    "InvocationEventType": "DIGITS_RECEIVED",
    "ActionData": {
        "ReceivedDigits": "11#",
        "Type": "ReceiveDigits",
        "Parameters": {
            "CallId": "call-id-1",
            "InputDigitsRegex": "^\d{2}#$",
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "FlushDigitsDurationInMilliseconds": 10000
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "159700958834234",
                "Status": "Connected"
            }
        ]
    }
}
```

## Ending a call

When your SIP application receives a hangup on any leg of a call, the application invokes a Lambda function. See the following code example.

```
// if LEG-A receives a disconnect, i.e., invitee hangs up
{
    "SchemaVersion": "1.0",
    "Sequence": 6,
    "InvocationEventType": "HANGUP",
    "ActionData": {
        "Type": "Hangup",
        "Parameters": {
            "CallId": "call-id-1",
            "ParticipantTag": "LEG-A"
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "LEG-A",
                "Direction": "Inbound",
                "To": "+19876543210",
                "From": "+11234567890",
                "StartTimeInMilliseconds": "1597009588",
                "Status": "Disconnected"
            }
        ]
    }
}

// if LEG-B receives a disconnect in a bridged call, i.e. the meeting ends
{
    "SchemaVersion": "1.0",
    "Sequence": 6,
    "InvocationEventType": "HANGUP",
    "ActionData": {
        "Type": "ReceiveDigits",
        "Parameters": {
            "CallId": "call-id-2",
            "ParticipantTag": "LEG-B"
        }
    },
    "CallDetails": {
        "TransactionId": "transaction-id",
        "AwsAccountId": "aws-account-id",
        "AwsRegion": "us-east-1",
        "SipRuleId": "sip-rule-id",
        "SipApplicationId": "sip-application-id",
        "Participants": [
            {
                "CallId": "call-id-1",
                "ParticipantTag": "Leg-A",
                "To": "+19876543210",
                "From": "+11234567890",
                "Direction": "Inbound",
                "StartTimeInMilliseconds": "1597009588",
                "Status": "Connected"
```

```
            },
            {
                "CallId": "call-id-2",
                "ParticipantTag": "Leg-B",
                "To": "+17035550122",
                "From": "SMA",
                "Direction": "Outbound",
                "StartTimeInMilliseconds": "1597010595",
                "Status": "Disconnected"
            }
        ]
    }
}
```

# Responding to invocations with action lists

You can respond to a Lambda invocation event with a list of actions to run on the individual participants in a call. The following code example shows the general response structure.

```
{
    "SchemaVersion": "1.0",
    "Actions": [
        // List of supported Actions
        {
            "Type": "PlayAudio",
            "Parameters": {
                "ParticipantTag": "LEG-A",
                "AudioSource": {
                    "Type": "S3",
                    "BucketName": "valid-S3-bucket-name",
                    "Key": "audio-file.wav"
                }
            }
        },
        ...
    ]
}
```

Whenever a SIP application invokes a Lambda function, the following operations occur:

1. The application finishes running the current action on a call.
2. The application then replaces the old action set with a new set of actions received from the latest invocation event.
3. If the SIP application doesn't receive new actions from the Lambda function, it keeps the existing actions.

# Supported actions for SIP applications

You can specify different types of signaling and media actions in a response from a Lambda function. The properties of each action vary, depending on the action. The following topics provide code examples and explain how to use the actions.

**Contents**

- ModifyChimeMeetingAttendee (muting and unmuting audio) (p. 83)
- RecordAudio (p. 86)
- JoinChimeMeeting (p. 89)
- Hangup (p. 90)

## Pause

Pause a call for a specified time.

```
{
    "Type": "Pause",
    "Parameters": {
        "CallId": "call-id-1",
        "DurationInMilliSeconds": "3000"
    }
   }
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**DurationInMilliSeconds**

*Description* – Duration of the pause, in milliseconds

*Allowed values* – An integer >0

*Required* – Yes

*Default value* – None

## PlayAudio

Play an audio file on any leg of a call. Currently, Amazon Chime only supports playing audio files from the Amazon Simple Storage Service (Amazon S3) bucket. The S3 bucket must belong to the same AWS account as the SIP application. In addition, you must give the `s3:GetObject` permission to the Amazon Chime Voice Connector service principal. You can do that by using the S3 console or the command-line interface (CLI).

The following code example shows a typical bucket policy.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "SMARead",
            "Effect": "Allow",
            "Principal": {
                "Service": "voiceconnector.chime.amazonaws.com"
            },
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

The following code example shows a typical action.

```
{
    "Type" : "PlayAudio",
    "Parameters" : {
        "CallId": "call-id-1",
        "AudioSource": {
           "Type": "S3",
           "BucketName": "valid-S3-bucket-name",
           "Key": "wave-file.wav"
         }
    }
}
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**AudioSource.Type**

*Description* – Type of source for audio file

*Allowed values* – S3

*Required* – Yes

*Default value* – None

**AudioSource.BucketName**

*Description* – For S3 source types, the S3 bucket must belong to the same AWS account as the SIP application. The bucket must have access to the Amazon Chime Voice Connector service principal, which is voiceconnector.chime.amazonaws.com.

*Allowed values* – A valid S3 bucket for which Amazon Chime has access to the `s3:GetObject` action.

*Required* – Yes

*Default value* – None

**AudioSource.key**

*Description* – For S3 source types, the file name from the S3 bucket specified in the `AudioSource.BucketName` attribute.

*Allowed values* – A valid audio file

*Required* – Yes

*Default value* – None

The SIP application tries to play the audio from the source URL. You can use You can use raw, uncompressed PCM .wav files of no more than 50 MB in size. For best results, use the s16le codec to output 8k mono-channel audio.

When the last instruction in a dialplan is `PlayAudio` and the file finishes playback, or if a user stops playback with a key press, the application invokes the Lambda function with the event shown in the following code example.

```
{
    "SchemaVersion": "1.0",
    "Sequence": INTEGER,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type": "PlayAudio",
        "Parameters" : {
            "CallId": "call-id-1",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "wave-file.wav",
         }
     }
}
```

After a terminating digit stops the audio, it won't be repeated.

**Error handling**

When the validation file contains errors, or an error occurs while running an action, the SIP application calls a Lambda function with the appropriate error code.

| Error | Message | Reason |
|---|---|---|
| `InvalidAudioSource` | The audio source parameter is invalid. | This error can happen for multiple reasons. For example, the SIP media application cannot access the file due to permission issues, or issues with the URL. Or, the audio file may fail validation due to format, duration, size, and so on. |

| Error | Message | Reason |
|---|---|---|
| `SystemException` | System error while running action. | Another system error occurred while running the action. |
| `InvalidActionParameter` | CallId or ParticipantTag parameter for action is invalid. | The action contains an invalid parameter. |

The following code example shows a typical invocation failure.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 2,
    "InvocationEventType": "ACTION_FAILED",
    "ActionData": {
        "Type": "PlayAudio",
        "Parameters" : {
            "CallId": "call-id-1",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "bucket-name",
                "Key": "audio-file.wav"
            },
        },
        "ErrorType": "InvalidAudioSource",
        "ErrorMessage": "Audio Source parameter value is invalid."
    }
    "CallDetails": {
        ...
    }
}
```

## ReceiveDigits

When a user enters digits or symbols that match the regex pattern specified in an action, the SIP application invokes a Lambda function.

```
{
    "Type": "ReceiveDigits",
    "Parameters": {
        "CallId": "call-id-1",
        "InputDigitsRegex": "^\d{2}#$",
        "InBetweenDigitsDurationInMilliseconds": 1000,
        "FlushDigitsDurationInMilliseconds": 10000
    }
}
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**InputDigitsRegex**

*Description* – A regex pattern

*Allowed values* – A valid regular expression pattern

*Required* – Yes

*Default value* – None

**InBetweenDigitsDurationInMilliseconds**

*Description* – Interval between digits before checking to see if the input matches the regex pattern

*Allowed values* – Duration in milliseconds

*Required* – Yes

*Default value* – None

**FlushDigitsDurationInMilliseconds**

*Description* – Interval after which received digits are flushed or ignored. If the SIP application receives a new digit after the interval ends, the timer starts again.

*Allowed values* – `InBetweenDigitsDurationInMilliseconds`

*Required* – Yes

*Default value* – None

A SIP application listens for digits for the length of a call unless it receives a new `ReceiveDigits` action. For every digit received, the application compares it to the regex pattern. If the initial digits match that pattern, the `InBetweenDigitsDurationInMilliseconds` interval starts. If the user doesn't enter the remaining necessary digits, the SIP application invokes the Lambda function for the next set of actions. If the pattern doesn't match, the application continues listening to digits by storing the previously received digit or digits.

The `FlushDigitsDurationInMilliseconds` interval starts when the application receives the first digit. If the application receives a digit after the flush interval ends, it ignores the previous digits and the cycle starts again. If it successfully receives a set of digits, the SIP application invokes the Lambda function described in Capturing digits (p. 70).

## PlayAudioAndGetDigits

Plays audio and gathers dual-tone multi frequency (DTMF) digits. If a failure occurs, such as a user not entering the correct number of digits, the action plays the "failure" audio and then replays the main audio.

You can play audio files from the Amazon Simple Storage Service (Amazon S3) bucket. The bucket must belong to the same AWS account as the SIP application. In addition, you must give the `s3:GetObject` permission to the Amazon Chime Voice Connector service principal. You can do that by using the S3 console or the command-line interface (CLI).

The following code example shows an S3 bucket policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SMARead",
            "Effect": "Allow",
            "Principal": {
                "Service": "voiceconnector.chime.amazonaws.com"
            },
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

The following code example shows a typical `PlayAudioAndGetDigits` action.

```
{
    "Type" : "PlayAudioAndGetDigits",
    "Parameters" : {
        "CallId": "call-id-1",
        "ParticipantTag": "LEG-A", //or LEG-B
        "InputDigitsRegex": "^\d{2}#$",
        "AudioSource": {
            "Type": "S3",
            "BucketName": "valid-s3-bucket-name",
            "Key": "audio-file-1.wav"
        },
        "FailureAudioSource": {
            "Type": "S3",
            "BucketName": "valid-s3-bucket-name",
            "Key": "audio-file-failure.wav"
        },
        "MinNumberOfDigits": 3,
        "MaxNumberOfDigits": 5,
        "TerminatorDigits": ["#"],
        "InBetweenDigitsDurationInMilliseconds": 5000,
        "Repeat": 3,
        "RepeatDurationInMilliseconds": 10000
    }
}
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**InputDigitsRegex**

*Description* – A regex pattern

*Allowed values* – A valid regular expression pattern

*Required* – No

*Default value* – None

**AudioSource.Type**

*Description* – Type of source for the audio file type

*Allowed values* – A remote URL

*Required* – Yes

*Default value* – `RemoteURL`

**AudioSource.BucketName**

*Description* – For S3 `AudioSource.Type` values, the S3 bucket must belong to the same AWS account as the SIP application. The bucket must have access to the Amazon Chime Voice Connector service principal, which is voiceconnector.chime.amazonaws.com.

*Allowed values* – A valid S3 bucket for which Amazon Chime has `s3:GetObject` actions access.

*Required* – Yes

*Default value* – None

**AudioSource.Key**

*Description* – For S3 source types, the file name from the S3 bucket specified in `AudioSource.BucketName`.

*Allowed values* – Valid audio files

*Required* – Yes

*Default value* – None

**FailureAudioSource.Type**

*Description* – Type of source for failure audio file

*Allowed values* – S3

*Required* – Yes

*Default value* – None

**FailureAudioSource.BucketName**

*Description* – For S3 source types, the S3 bucket must belong to the same AWS account as the SIP media application. The bucket should have access to the Amazon Chime Voice Connector service principal, which is voiceconnector.chime.amazonaws.com.

*Allowed values* – A valid S3 bucket for which Amazon Chime has `s3:GetObject` actions access.

*Required* – Yes

*Default value* – None

**FailureAudioSource.Key**

*Description* – For S3 types, the file name from the S3 bucket specified in the `AudioSource.BucketName` attribute.

*Allowed values* – Valid audio files

*Required* – Yes

*Default value* – None

**MinNumberOfDigits**

*Description* – The minimum number of digits to get before timing out or playing a "call failed" audio clip.

*Allowed values* – >=0

*Required* – No

*Default value* – 0

**MaxNumberOfDigits**

*Description* – The maximum number of digits to get before stopping without a terminating digit.

*Allowed values* – `>MinNumberOfDigits`

*Required* – No

*Default value* – 128

**TerminatorDigits**

*Description* – Digits used to end input if the user enters less than the `MaxNumberOfDigits`

*Allowed values* – Any one of these digits: 0123456789#*

*Required* – No

*Default value* – #

**InBetweenDigitsDurationInMilliseconds**

*Description* – The wait time in milliseconds between digit inputs before playing `FailureAudio`.

*Allowed values* – >0

*Required* – No

*Default value* – If not specified, the `RepeatDurationInMilliseconds` value.

**Repeat**

*Description* – Total number of attempts to get digits

*Allowed values* – >0

*Required* – No

*Default value* – 1

**RepeatDurationInMilliseconds**

*Description* – Time in milliseconds to wait before next attempt

*Allowed values* – >0

*Required* – Yes

*Default value* – None

Your SIP application always invokes its Lambda function after running this action, with an `ACTION_SUCCESSFUL` or `ACTION_FAILED` event type. When the application successfully gathers digits, it sets the `ReceivedDigits` variable in the `ActionData` object. The following code example shows the event structure of that Lambda function.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 3,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type": "PlayAudioAndGetDigits",
        "Parameters" : {
            "ParticipantTag": "LEG-A",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "audio-file.mp3"
            },
            "FailureAudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "failure-audio-file.wav"
            },
            "MinNumberOfDigits": 3,
            "MaxNumberOfDigits": 5,
            "TerminatorDigits": ["#"],
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "Repeat": 3,
            "RepeatDurationInMilliseconds": 10000
        },
        "ReceivedDigits": "1234"
    },
    "CallDetails": {
        ...
    }
}
```

**Error handling**

When a validation error occurs, the SIP application calls the Lambda function with the appropriate error message. The following table lists the error messages.

| Error | Message | Reason |
|---|---|---|
| `InvalidAudioSource` | Audio source parameter value is invalid. | This error can happen for multiple reasons. For example, SMA cannot access the file due to permission issues, or issues with the URL. Or, the audio file may fail validation due to format, duration, size, and so on. |
| `InvalidActionParameter` | CallId or ParticipantTag parameter for action is invalid. | A `CallId`, `ParticipantTag`, or other parameter is not valid. |

| Error | Message | Reason |
| --- | --- | --- |
| `SystemException` | System error while running action. | A system error occurred while running the action. |

When the action fails to collect the number of specified digits due to a timeout or too many retries, the SIP application invokes a Lambda function with the `ActionFailed` event type.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 4,
    "InvocationEventType": "ACTION_FAILED",
    "ActionData": {
        "Type": "PlayAudioAndGetDigits",
        "Parameters" : {
            "ParticipantTag": "LEG-A",
            "AudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "audio-file.mp3"
            },
            "FailureAudioSource": {
                "Type": "S3",
                "BucketName": "valid-S3-bucket-name",
                "Key": "failure-audio.mp3"
            },
            "MinNumberOfDigits": 3,
            "MaxNumberOfDigits": 5,
            "TerminatorDigits": ["#"],
            "InBetweenDigitsDurationInMilliseconds": 5000,
            "Repeat": 3,
            "RepeatDurationInMilliseconds": 10000
        },
        "ErrorType": "InvalidAudioSource",
        "ErrorMessage": "Audio Source parameter value is invalid."
    }
    "CallDetails": {
        ...
    }
}
```

## ModifyChimeMeetingAttendee (muting and unmuting audio)

Allows the SIP application to modify the status of a telephony attendee joined in the meeting by providing the Amazon Chime meeting ID and attendee list.

> **Note**
> This action currently supports mute and unmute operations on telephony attendees. Also, the current user must be joined into a meeting using the `JoinChimeMeeting` action. This action can be performed on a `participantTag="LEG-B"`, or a corresponding `CallId`.

This action only applies to the callLeg that joins from the SIP application to `"+17035550122"`, or LEG-B, or the leg that is joined from the SIP application to the meeting.

```
{
"SchemaVersion": "1.0",
  "Actions": [
    {
      "Type" : "ModifyChimeMeetingAttendees",
      "Parameters" : {
```

```
        "Operation": "Mute", //or Unmute
        "MeetingId": "meeting-id",
        "CallId": "call-id",
        "ParticipantTag": participant-tag",
        "AttendeeList": ["attendee-id-1", "attendee-id-2"]
      }
    }
  ]
}
```

**Operation**

**Description** – The operation to perform on the list of attendees

**Allowed values** – Mute, Unmute

**Required** – Yes

**Default value** – None

**MeetingId**

**Description** – The ID of the meeting to which the attendees belong.

**Allowed values** – A valid meeting ID. The person muting or unmuting must also belong to the meeting.

**Required** – Yes

**Default value** – None

**CallId**

**Description** – The ID of the meeting to which the attendees belong.

**Allowed values** – A valid call ID.

**Required** – No

**Default value** – None

**ParticipantTag**

**Description** – The tag assigned to the attendee.

**Allowed values** – A valid tag.

**Required** – No

**Default value** – None

**AttendeeList**

**Description** – List of attendee IDs to mute or unmute

**Allowed values** – A list of valid attendee IDs

**Required** – Yes

**Default value** – None

After running this action, SIP media applications always invoke a Lambda function with the `ACTION_SUCCESSFUL` or `ACTION_FAILED` invocation event type. The following example shows a typical invocation event structure for a Lambda function.

```
{
    "SchemaVersion": "1.0",
    "Sequence": INTEGER,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type" : "ModifyChimeMeetingAttendees",
        "Parameters" : {
            "Operation": "Mute", //or Unmute
            "MeetingId": "meeting-id",
            "CallId": "call-id",
            "ParticipantTag": "participant-tag",
            "AttendeeList": ["attendee-id-1", "attendee-id-2"]
        }
    }
    "CallDetails": {
        ...
    }
}
```

**Error handling**

In cases of invalid instruction parameters or API failures, SIP media applications call a Lambda function with the error message specific to the failed instruction or API.

| Error | Message | Reason |
| --- | --- | --- |
| `InvalidActionParameter` | The `ModifyChimeMeetingAttendees` `Operation` parameter value is invalid | The `Operation` value must be Mute or Unmute. |
| | Meeting ID parameter value is invalid. | Meeting ID is empty. |
| | Attendee List parameter value is invalid. | The Attendee ID list is empty. |
| | Invalid action on the call. | The call isn't bridged. |
| | Call is not connected to Chime Meeting. | The attendee is not connected to a Chime Meeting. |
| | One or more attendees are not part of this meeting. All attendees must be part of this meeting. | The attendee is not authorized to modify attendees in the meeting. |
| `SystemException` | System error while running action. | A system error occurred while running an action. |

The following example code shows a typical failure event:

```
{
    "SchemaVersion": "1.0",
    "Sequence": INTEGER,
    "InvocationEventType": "ACTION_FAILED",
    "ActionData": {
        "Type" : "ModifyChimeMeetingAttendees",
```

```
        "Parameters" : {
            "Operation": "Mute", //or Unmute
            "MeetingId": "meeting-id",
            "CallId": "call-id",
            "ParticipantTag": "participant-tag",
            "AttendeeList": ["attendee-id-1", "attendee-id-2"]
        },
        "ErrorType": "",
        "ErrorMessage": "",
        "ErrorList": []
    }
    "CallDetails": {
        ...
    }
}
```

## RecordAudio

Allows your SIP application to record media from a given call ID. The application records until it reaches the duration that you set, or when a user presses one of the `RecordingTerminators`. In those cases, the action tells your application to send the media file to the specified Amazon Simple Storage Service (Amazon S3) bucket. The S3 bucket must belong to the same AWS account as the SIP application. In addition, the action must give `s3:PutObject` and `s3:PutObjectAcl` permission to the Amazon Chime Voice Connector service principal.

The following example gives the `s3:PutObject` and `s3:PutObjectAcl` permission to the Amazon Chime Voice Connector service principal.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SMARead",
            "Effect": "Allow",
            "Principal": {
                "Service": "voiceconnector.chime.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::bucket-name/"
        }
    ]
}
```

The following example stops recording after 10 seconds and sends the media file to the specifed S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Type" : "RecordAudio",
            "Parameters" : {
                "CallId": "call-id-1",
                "DurationInSeconds": "10",
                "RecordingTerminators": ["#"],
                "RecordinDestination": {
```

```
                "Type": "S3",
                "BucketName": "valid-bucket-name"
        }
    }
}
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**RecordingDestination.Type**

*Description* – Type of destination. Only S3.

*Allowed values* – S3

*Required* – Yes

*Default value* – None

**RecordingDestination.BucketName**

*Description* – For S3 recording destinations, you must provide a valid S3 bucket name. The bucket must have access to the Amazon Chime Voice Connector service principal, which is voiceconnector.chime.amazonaws.com.

*Allowed values* – A valid S3 bucket for which Amazon Chime has access to the `s3:PutObject` and `s3:PutObjectAcl` actions.

*Required* – Yes

*Default value* – None

**DurationInSeconds**

*Description* – The duration of the recording, in seconds

*Allowed values* – >0

*Required* – Yes

*Default value* – None

**RecordingTerminators**

*Description* – Stops the recording when the user presses any of the digits associated with the attribute.

*Allowed values* – An array of single digits and symbols from [123456789*0#]

*Required* – Yes

*Default value* – None

When the recording ends, the application calls its Lambda function, complete with a file name and S3 bucket path.

```
{
    "SchemaVersion": "1.0",
    "Sequence": INTEGER,
    "InvocationEventType": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type" : "RecordAudio",
        "Parameters" : {
            "CallId": "call-id-1",
            "DurationInSeconds": "10",
            "RecordingTerminators": ["#"],
            "RecordinDestination": {
                "Type": "S3",
                "BucketName": "valid-bucket-name"
            }
        },
        "RecordinDestination": {
            "Type": "S3",
            "BucketName": "valid-bucket-name",
            "Key": "call-id-1-recordings-counter.mp3"
        }
    }
    "CallDetails": {
        ...
    }
}
```

**Error handling**

For validation errors, the application calls the Lambda function with the appropriate error message. The following table lists the error messages.

| Error | Message | Reason |
|---|---|---|
| `InvalidActionParameter` | CallId or ParticipantTag parameter for action is invalid | `callId` or other parameter is invalid. |
| `SystemException` | System error while running an action. | Another type of system error occurred while running an action. |

When the action fails to record the media on a call leg, the SIP application invokes a Lambda function with the `ActionFailed` event type. See the following code example.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 5,
    "InvocationEventType": "ACTION_FAILED",
    "ActionData": {
        "Type" : "RecordAudio",
        "Parameters" : {
            "ParticipantTag": "LEG-A",
            "DestinationType": "S3",
```

```
            "DestinationValue": "/path/to/s3/bucket",
            "MaxDurationInSeconds": "10",
            "StopRecordingOnDigits": ["#"]
        },
        "Error": "NoAccessTODestination: Error while accessing destination"
    }
    "CallDetails": {
        ...
    }
}
```

## JoinChimeMeeting

Join an Amazon Chime SDK meeting by providing the attendee join token. To do this, you make AWS SDK calls to the CreateMeeting and CreateAttendee APIs to get the token and pass it on in the action. See the following code example.

**Note**
You can't run this action on a bridged call.

```
{
    "Type": "JoinChimeMeeting",
    "Parameters": {
        "JoinToken": "meeting-attendee-join-token",
        "CallId": "call-id-1"
    }
}
```

**CallID**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**JoinToken**

*Description* – A valid join token of the Amazon Chime meeting attendee.

*Allowed values* – Valid join token

*Required* – Yes

*Default value* – None

The SIP application always invokes a Lambda function after running this action. It returns either the `ACTION_SUCCESSFUL` or `ActionFailed` invocation event types. The following code example shows the invocation event structure of the Lambda function.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 4,
    "InvocationEvent": "ACTION_SUCCESSFUL",
    "ActionData": {
        "Type": "JoinChimeMeeting",
        "Parameters": {
            "JoinToken": "meeting-attendee-join-token",
            "CallId": "call-id-1"
        }
    }
    "CallDetails": {
        ...
    }
  }
```

**Error handling**

When a validation error occurs while bridging a meeting, the SIP application calls its Lambda function with one of the error messages shown in the following table.

| Error | Message | Reason |
|---|---|---|
| `InvalidActionParameter` | `JoinToken` parameter value is invalid. | Any of the action's other parameters is invalid or missing. |
| `SystemException` | System error while running action. | Another type of system error occurred while running the action. |

The following code example shows a typical failure event.

```
{
    "SchemaVersion": "1.0",
    "Sequence": 3,
    "InvocationEvent": "ActionFailed",
    "ActionData": {
        "Type": "JoinChimeMeeting",
        "Parameters": {
            "JoinToken": "meeting-attendee-join-token",
            "CallId": "call-id-1"
        },
        "Error": "ErrorJoiningMeeting: Error while joining meeting."
    }
    "CallDetails": {
        ...
    }
}
```

## Hangup

Sends a `Hangup` value with a `SipStatusCode` to any leg of a call. See the following code example.

```
{
    "Type": "Hangup",
    "Parameters": {
        "ParticipantTag": "MeetingParticipant1",
        // 480 - Unavailable, 486 - Busy, 0 - Terminated
```

```
        "SipResponseCode": "486"
    }
}
```

**CallId**

*Description* – `CallId` of participant in the `CallDetails`

*Allowed values* – A valid call ID

*Required* – No

*Default value* – None

**ParticipantTag**

*Description* – `ParticipantTag` of one of the connected participants in the `CallDetails`

*Allowed values* – `LEG-A` or `LEG-B`

*Required* – No

*Default value* – `ParticipantTag` of the invoked `callLeg`. Ignored if you specify `CallId`.

**SipResponseCode**

*Description* – Any of the supported SIP response codes.

*Allowed values* – 480–Unavailable; 486–Busy; 0–Normal Termination

*Required* – No

*Default value* – 0

After a user ends a call, the SIP application invokes a Lambda function with the code listed in Ending a call (p. 72).

# Timeouts and retries

SIP applications interact with Lambda functions synchronously. Applications wait 5 seconds for Lambda functions to respond before retrying an invocation. When a function throws an error with one of the 4*XX* status codes, then by default the SIP application only retries the invocation once. If you run out of retries, calls terminate with the `480 Unavailable` error code. For more information about Lambda errors, see Troubleshoot invocation issues in AWS Lambda .

# Document history

The following table describes important changes to the *Amazon Chime Developer Guide*, beginning in September 2019. For notifications about updates to this documentation, you can subscribe to an RSS feed.

| update-history-change | update-history-description | update-history-date |
| --- | --- | --- |
| Lambda functions SDK (p. 92) | Developers can build custom Lambda functions for use in the Amazon Chime SIP media applications created by Amazon Chime administrators. For more information, see Building Lambada functions for SIP media applications in the *Amazon Chime Developer Guide*. | November 17, 2020 |
| JavaScript SDK (p. 92) | Developers can use JavaScript to build Amazon Chime applications. For more information, see Using the Amazon Chime SDK for JavaScript in the *Amazon Chime Developer Guide*. | November 17, 2020 |
| Android and iOS SDKs (p. 92) | Developers can find the Amazon Chime SDKs for Android and iOS in less time and with fewer clicks. For more information, see Using the Amazon Chime SDK for Android and Using the Amazon Chime SDK for iOSin the *Amazon Chime Developer Guide*. | November 17, 2020 |
| Proxy phone sessions (p. 92) | Developers can create proxy phone sessions for use with Amazon Chime Voice Connectors. For more information, see Using the Amazon Chime SDK for JavaScript in the *Amazon Chime Developer Guide*. | April 7, 2020 |
| Amazon Chime SDK content sharing (p. 92) | The Amazon Chime SDK supports content sharing. For more information, see Amazon Chime SDK architecture in the *Amazon Chime Developer Guide*. | March 31, 2020 |
| Amazon Chime SDK for Android and iOS (p. 92) | The Amazon Chime SDK for Android and iOS is released. For more information, see | March 24, 2020 |

| | | |
|---|---|---|
| | Integrating with a client library in the *Amazon Chime Developer Guide*. | |
| Amazon Chime SDK (p. 92) | The Amazon Chime SDK is released. For more information, see Using the Amazon Chime SDK in the *Amazon Chime Developer Guide*. | November 20, 2019 |
| Amazon Chime Developer Guide (p. 92) | The *Amazon Chime Developer Guide* is released. | September 11, 2019 |

# AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.