aws

Real-Time Streaming User Guide

Amazon IVS

Amazon IVS: Real-Time Streaming User Guide

Table of Contents

What is Amazon IVS Real-Time Streaming?

Amazon Interactive Video Service (IVS) Real-Time Streaming gives you everything you need to add real-time audio and video to your applications.

Strengths:

- Real-time latency — Build applications for latency-sensitive use cases, helping your viewers stay connected and engaged with IVS real-time streaming. Deliver live streams with a latency that can be under 300 milliseconds from host to viewer.
- High concurrency — Unlock the potential of large-scale interactions with IVS real-time streaming. Accommodate audiences beyond 25,000 viewers and enable up to 12 hosts to take the virtual stage. (For default limits and instructions on requesting an increase, see *Service Quotas* for real-time streaming and low-latency streaming.)
- Mobile optimized — IVS real-time streaming is optimized for mobile use cases, catering to a diverse range of devices and network capabilities. By integrating the Amazon IVS broadcast SDKs for Android and iOS, your users can engage as hosts or viewers, enjoying high-quality live streams on their mobile devices.

Use cases:

- Guest spots — Create applications that allow hosts to promote guests "on stage," turning viewers into hosts for real-time interactions.
- Versus (VS) mode — Produce experiences with side-by-side competitions and let viewers watch hosts compete in real-time.
- Audio rooms — Invite listeners to join the conversation as guests and foster deeper engagement in your audio rooms.
- Live video auctions — Turn auctions into interactive video events and maintain their excitement and integrity with real-time latency.

In addition to the product documentation here, see https://ivs.rocks/, a dedicated site to browse published content (demos, code samples, blog posts), estimate cost, and experience Amazon IVS through live demos.

Global Solution, Regional Control

Streaming and Viewing are Global

You can use Amazon IVS to stream to viewers worldwide:

- When you stream, Amazon IVS automatically ingests video at a location near you.
- Viewers can watch your live streams globally.

Another way of saying this is that the "data plane" is global. The data plane refers to streaming/ingesting and viewing.

Control is Regional

While the Amazon IVS data plane is global, the "control plane" is regional. The control plane refers to the Amazon IVS console, API, and resources (stages).

Another way of saying this is that Amazon IVS is a "regional AWS service." That is, Amazon IVS resources in each region are independent of similar resources in other regions. For example, a stage that you create in one region is independent of stages you create in other regions.

When you use resources (e.g., create a stage), you must specify the region in which it will be created. Subsequently, when you manage resources, you must do so from the same region where they were created.

If you use the ...	You specify the region by ...
Amazon IVS console	Using the **Select a Region** drop-down in the top right of the navigation bar.
Amazon IVS API	Using the appropriate service endpoint. See the Amazon IVS Real-Time Streaming API Reference. (If you access the API through an SDK, set up the SDK's `region` parameter. See Tools to Build on AWS.)
AWS CLI	Either: • Appending `--region <aws-region>` to your CLI command.

If you use the ...	You specify the region by ...
	• Putting the region in your local AWS configuration file.

Remember, regardless of the region in which a stage was created, you can stream to Amazon IVS from anywhere, and viewers can watch from anywhere.

Getting Started with IVS Real-Time Streaming

This document takes you through the steps involved in integrating Amazon IVS Real-Time Streaming into your app.

Topics

Introduction to IVS Real-Time Streaming

This section lists prerequisites for using real-time streaming and introduces key terminology.

Prerequisites

Before you use Real-Time Streaming for the first time, complete the following tasks. For instructions, see Getting Started with IVS Low-Latency Streaming.

- Create an AWS Account
- Set Up Root and Administrative Users

Other References

- IVS Web Broadcast SDK Reference
- IVS Android Broadcast SDK Reference
- IVS iOS Broadcast SDK Reference
- IVS Real-Time Streaming API Reference

Real-Time Streaming Terminology

Term	Description
Stage	A virtual space where participants can exchange video in real time.
Host	A participant that sends local video to the stage.
Viewer	A participant that receives video of the hosts.
Participant	A user connected to the stage as a host or viewer.
Participant token	A token that authenticates a participant when they join a stage.
Broadcast SDK	A client library that enables participants to send and receive video.

Overview of Steps

1. Set up IAM Permissions — Create an AWS Identity and Access Management (IAM) policy that gives users a basic set of permissions and assign that policy to users.
2. Create a stage — Create a virtual space where participants can exchange video in real time.
3. Distribute participant tokens — Send tokens to participants so they can join your stage.
4. Integrate the IVS Broadcast SDK — Add the broadcast SDK to your app to enable participants to send and receive video: the section called “Web”, the section called “Android”, and the section called “iOS”.

5. Publish and subscribe to video — Send your video to the stage and receive video from other hosts: IVS console, the section called “Web”, the section called “Android”, and the section called “iOS”.

Step 1: Set Up IAM Permissions

Next, you must create an AWS Identity and Access Management (IAM) policy that gives users a basic set of permissions (e.g., to create an Amazon IVS stage and create participant tokens) and assign that policy to users. You can either assign the permissions when creating a new user or add permissions to an existing user. Both procedures are given below.

For more information (for example, to learn about IAM users and policies, how to attach a policy to a user, and how to constrain what users can do with Amazon IVS), see:

- Creating an IAM User in the *IAM User Guide*
- The information in Amazon IVS Security on IAM and "Managed Policies for IVS."
- The IAM information in Amazon IVS Security

You can either use an existing AWS managed policy for Amazon IVS or create a new policy that customizes the permissions you want to grant to a set of users, groups, or roles. Both approaches are described below.

Use an Existing Policy for IVS Permissions

In most cases, you will want to use an AWS managed policy for Amazon IVS. They are described fully in the Managed Policies for IVS section of *IVS Security*.

- Use the `IVSReadOnlyAccess` AWS managed policy to give your application developers access to all IVS Get and List API operations (for both low-latency and real-time streaming).
- Use the `IVSFullAccess` AWS managed policy to give your application developers access to all IVS API operations (for both low-latency and real-time streaming).

Optional: Create a Custom Policy for Amazon IVS Permissions

Follow these steps:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, then choose **Create policy**. A **Specify permissions** window opens..
3. In the **Specify permissions** window, choose the **JSON** tab, and copy and paste the following IVS policy to the **Policy editor** text area. (The policy does not include all Amazon IVS actions. You can add/delete (Allow/Deny) operation access permissions as needed. See IVS Real-Time Streaming API Reference for details on IVS operations.)

   JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ivs:CreateStage",
            "ivs:CreateParticipantToken",
            "ivs:GetStage",
            "ivs:GetStageSession",
            "ivs:ListStages",
            "ivs:ListStageSessions",
            "ivs:CreateEncoderConfiguration",
            "ivs:GetEncoderConfiguration",
            "ivs:ListEncoderConfigurations",
            "ivs:GetComposition",
            "ivs:ListCompositions",
            "ivs:StartComposition",
            "ivs:StopComposition"
          ],
          "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:DescribeAlarms",
            "cloudwatch:GetMetricData",
            "s3:DeleteBucketPolicy",
            "s3:GetBucketLocation",
            "s3:GetBucketPolicy",
            "s3:PutBucketPolicy",
```

```
            "servicequotas:ListAWSDefaultServiceQuotas",
            "servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota",
            "servicequotas:ListServiceQuotas",
            "servicequotas:ListServices",
            "servicequotas:ListTagsForResource"
        ],
        "Resource": "*"
    }
  ]
}
```

4. Still in the **Specify permissions** window, choose **Next** (scroll to the bottom of the window to see this). A **Review and create** window opens.
5. On the **Review and create** window, enter a **Policy name** and optionally add a **Description**. Make a note of the policy name, as you will need it when creating users (below). Choose **Create policy** (at the bottom of the window).
6. You are returned to the IAM console window, where you should see a banner confirming that your new policy was created.

Create a New User and Add Permissions

IAM User Access Keys

IAM access keys consist of an access key ID and a secret access key. They are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not create root-user access keys.

The only time that you can view or download a secret access key is when you create access keys. You cannot recover them later. However, you can create new access keys at any time; you must have permissions to perform the required IAM actions.

Always store access keys securely. Never share them with third parties (even if an inquiry seems to come from Amazon). For more information, see Managing access keys for IAM users in the *IAM User Guide*.

Procedure

Follow these steps:

1. In the navigation pane, choose **Users**, then choose **Create user**. A **Specify user details** window opens.
2. In the **Specify user details** window:
   a. Under **User details**, type the new **User name** to be created.
   b. Check **Provide user access to the AWS Management Console**.
   c. Under **Console password**, select **Autogenerated password**.
   d. Check **Users must create a new password at next sign-in**.
   e. Choose **Next**. A **Set permissions** window opens.
3. Under **Set permissions**, select **Attach policies directly**. A **Permissions policies** window opens.
4. In the search box, enter an IVS policy name (either an AWS managed policy or your previously created custom policy). When it is found, check the box to select the policy.
5. Choose **Next** (at the bottom of the window). A **Review and create** window opens.
6. On the **Review and create** window, confirm that all user details are correct, then choose **Create user** (at the bottom of the window).
7. The **Retrieve password** window opens, containing your **Console sign-in details**. *Save this information securely for future reference*. When you are done, choose **Return to users list**.

Add Permissions to an Existing User

Follow these steps:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**, then choose an existing user name to be updated. (Choose the name by clicking on it; do not check the selection box.)
3. On the **Summary** page, on the **Permissions** tab, choose **Add permissions**. An **Add permissions** window opens.
4. Select **Attach existing policies directly**. A **Permissions policies** window opens.
5. In the search box, enter an IVS policy name (either an AWS managed policy or your previously created custom policy). When the policy is found, check the box to select the policy.
6. Choose **Next** (at the bottom of the window). A **Review** window opens.
7. On the **Review** window, select **Add permissions** (at the bottom of the window).

8. On the **Summary** page, confirm that the IVS policy was added.

Step 2: Create a Stage with Optional Participant Recording

A stage is a virtual space where participants can exchange video in real time. It is the foundational resource of the Real-Time Streaming API. You can create a stage using either the console or the CreateStage operation.

We recommend that where possible, you create a new stage for each logical session and delete it when done, rather than keeping around old stages for possible reuse. If stale resources (old stages, not to be reused) are not cleaned up, you're likely to hit the limit of the maximum number of stages faster.

You can create a stage — with or without individual participant recording — through the Amazon IVS console or the AWS CLI. Stage creation and recording are discussed below.

Individual Participant Recording

You have the option of enabling individual participant recording for a stage. If the individual participant recording to S3 feature is enabled, all individual participant broadcasts to the stage are recorded and saved to an Amazon S3 storage bucket that you own. Subsequently, the recording is available for on-demand playback.

Setting this up is an advanced option. By default, recording is disabled when a stage is created.

Before you can set up a stage for recording, you must create a *storage configuration*. This is a resource which specifies an Amazon S3 location where the recorded streams for the stage are stored. You can create and manage storage configurations using the console or CLI; both procedures are given below. After you create the storage configuration, you associate it with a stage either when you create the stage (as described below) or later, by updating an existing stage. (In the API, see CreateStage and UpdateStage.) You can associate multiple stages with the same storage configuration. You can delete a storage configuration that is no longer associated with any stages.

Keep in mind the following constraints:

- You must own the S3 bucket. That is, the account that sets up a stage to be recorded must own the S3 bucket where recordings will be stored.

- The stage, storage configuration, and S3 location must be in the same AWS region. If you create stages in other regions and want to record them, you must also set up storage configurations and S3 buckets in those regions.

Recording to your S3 bucket requires authorization with your AWS credentials. To give IVS the required access, an AWS IAM Service-Linked Role (SLR) is created automatically when the recording configuration is created: the SLR is limited to give IVS write permission only on the specific bucket.

Note that network issues between the streaming location and AWS or within AWS could result in some data loss while recording your stream. In these cases, Amazon IVS prioritizes the live stream over the recording. For redundancy, record locally via your streaming tool.

For more information (including how to set up post-processing or VOD playback on your recorded files), see Individual Participant Recording.

How to Disable Recording

To disable Amazon S3 recording on an existing stage:

- Console — On the details page for the relevant stage, in the **Record individual participants** streams section, toggle off **Enable automatic recording** under **Auto-record to S3** and then choose **Save changes**. This removes the storage configuration's association with the stage; streams on that stage will no longer be recorded.
- CLI — Run the `update-stage` command and pass in the recording-configuration ARN as an empty string:

```
aws ivs-realtime update-stage --arn arn:aws:ivs:us-west-2:123456789012:stage/
abcdABCDefgh --auto-participant-recording-configuration storageConfigurationArn=""
```

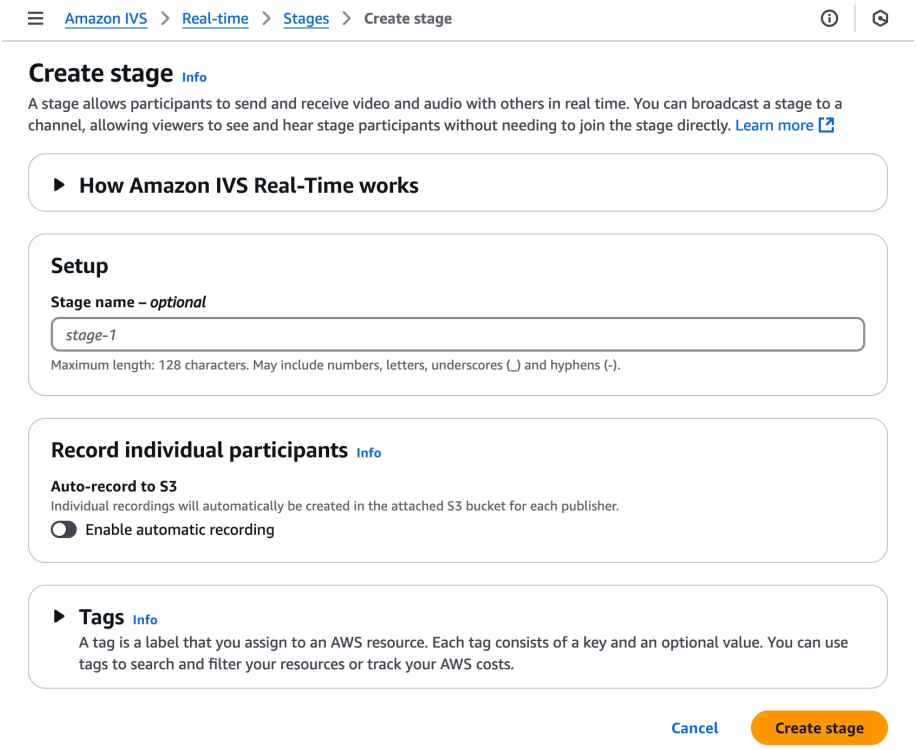
  This returns a stage object with an empty string for `storageConfigurationArn`, indicating that the recording is disabled.

Console Instructions for Creating an IVS Stage

1. Open the Amazon IVS console.

   (You also can access the Amazon IVS console through the AWS Management Console.)

2. On the left navigation pane, select **Stages**, then select **Create stage**. The **Create stage** window appears.

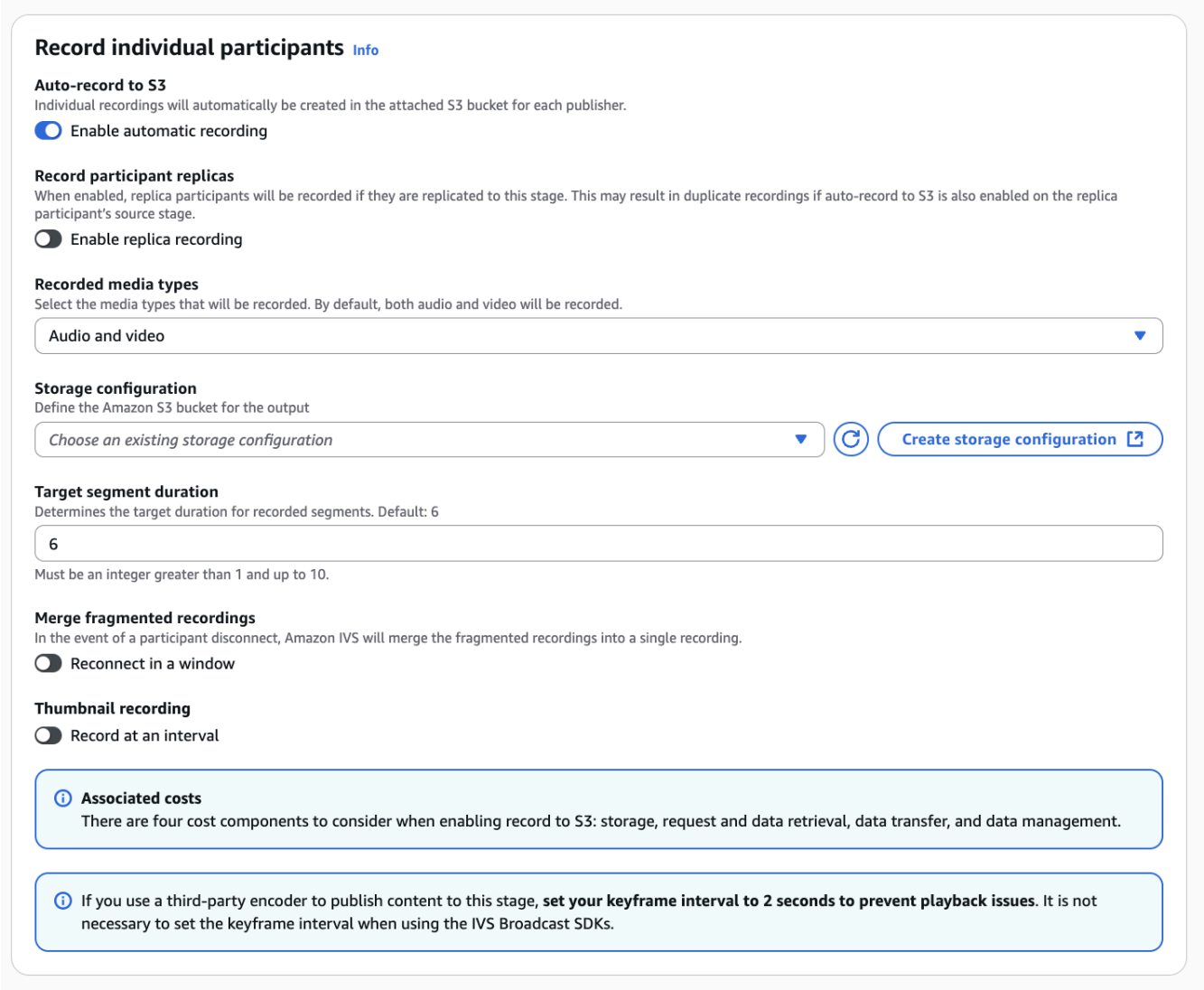


3. Optionally enter a **Stage name**.
4. If you want to enable individual participant recording, complete the steps in Set Up Automatic Individual Participant Recording to Amazon S3 (Optional) below.
5. Select **Create stage** to create the stage. The stage details page appears, for the new stage.

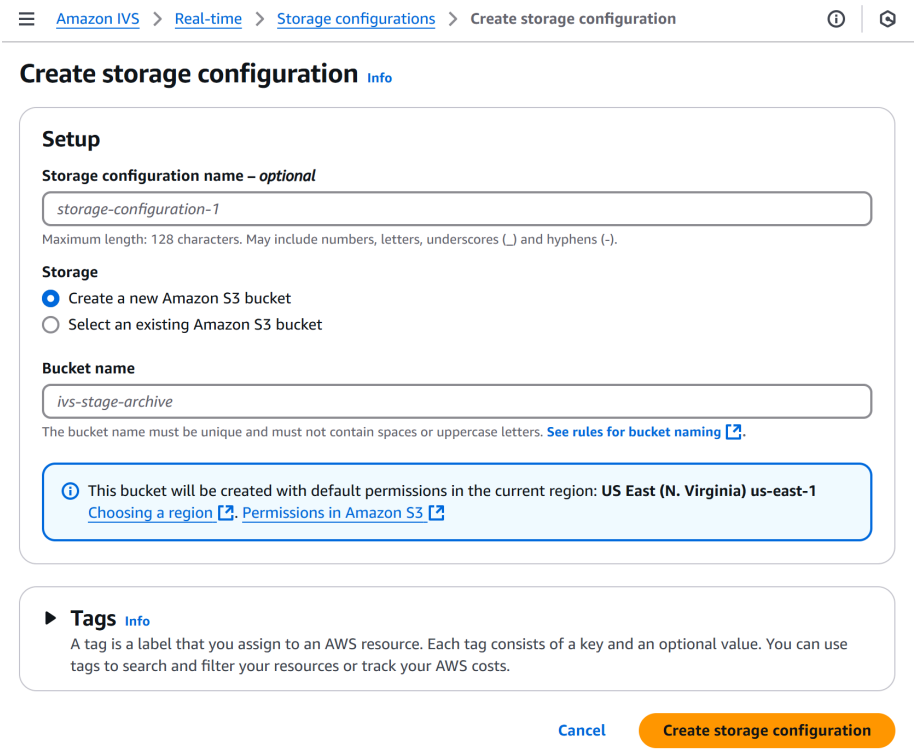
Set Up Automatic Individual Participant Recording to Amazon S3 (Optional)

Follow these steps to enable individual participant recording while creating a stage:

1. On the **Create stage** page, under **Record individual participants**, turn on **Enable automatic recording**. Additional fields display, to choose **Recorded media types**, to choose an existing **Storage configuration** or create a new one, and to choose whether to record thumbnails at an interval.



2. Choose which media types to record.
3. Choose **Create storage configuration**. A new window opens, with options for creating an Amazon S3 bucket and attaching it to the new recording configuration.

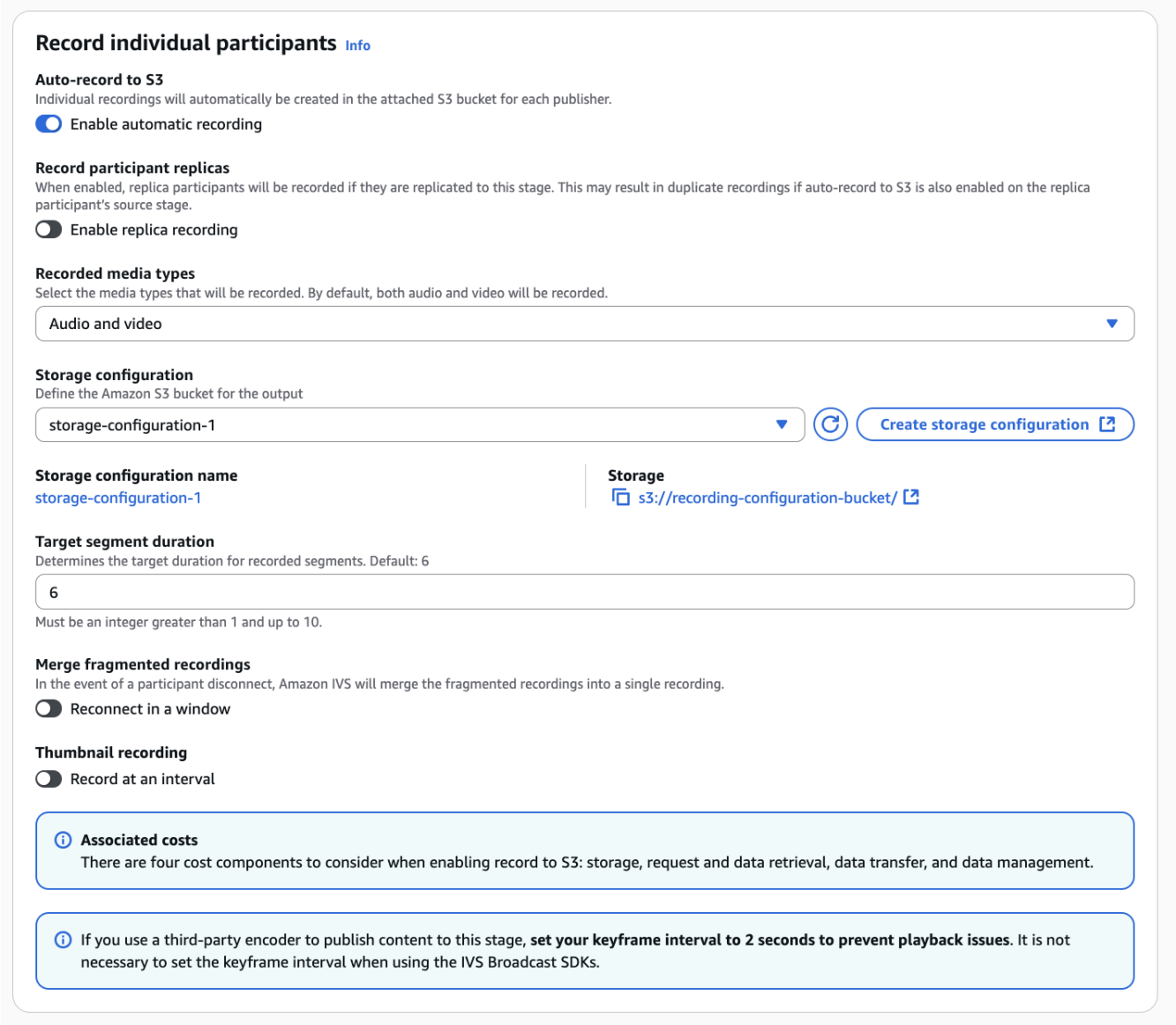


4. Fill out the fields:

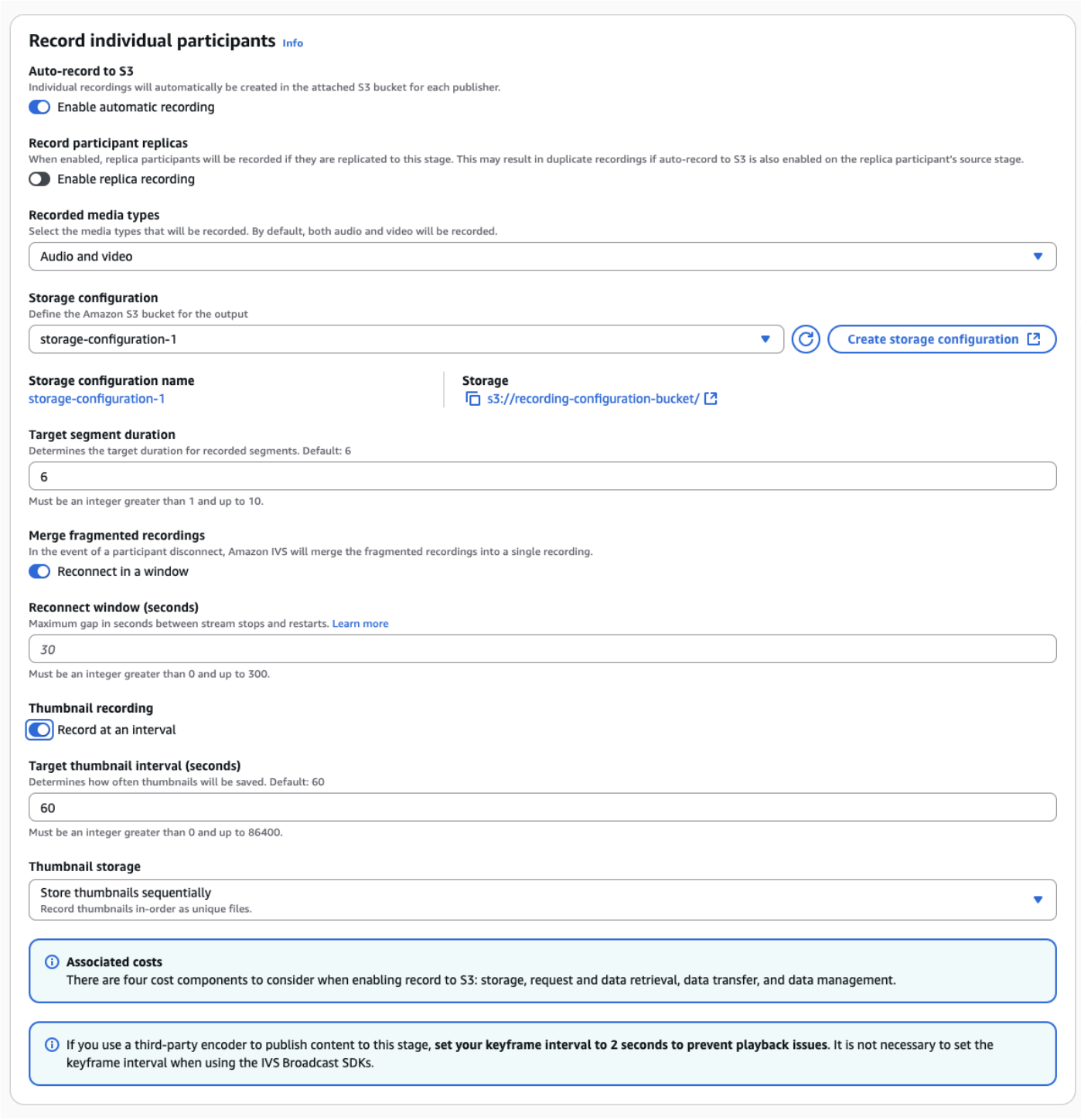
   a. Optionally enter a **Storage configuration name**.

   b. Enter a **Bucket name**.

5. Choose **Create storage configuration**, to create a new storage-configuration resource with a unique ARN. Typically, creation of the recording configuration takes a few seconds, but it can be up to 20 seconds. When the storage configuration is created, you are returned to the **Create stage** window. There, the **Record individual participants** area shows your new **Storage configuration** and the S3 bucket (**Storage**) that you created.



6. You can optionally enable other non-default options such as recording participant replicas, merging individual participant recordings, and thumbnail recording.



CLI Instructions for Creating an IVS Stage

To install the AWS CLI, see Install or update to the latest version of the AWS CLI.

Now you can use the CLI to create and manage resources following one of the two procedures below, depending on whether you want to create a stage with or without individual participant recording enabled.

Create a Stage without Individual Participant Recording

The stage API is under the ivs-realtime namespace. For example, to create a stage:

```
aws ivs-realtime create-stage --name "test-stage"
```

The response is:

```
{
   "stage": {
      "arn": "arn:aws:ivs:us-west-2:376666121854:stage/VSWjvX5XOkU3",
      "name": "test-stage"
   }
}
```

Create a Stage with Individual Participant Recording

To create a stage with individual participant recording enabled:

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration storageConfigurationArn=arn:aws:ivs:us-
west-2:123456789012:storage-configuration/LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO
```

Optionally, pass the `thumbnailConfiguration` parameter to manually set the thumbnail storage and recording mode, and thumbnail interval seconds:

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration storageConfigurationArn=arn:aws:ivs:us-
west-2:123456789012:storage-configuration/
LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO,thumbnailConfiguration="{targetIntervalSeconds=10,storage=[
```

Optionally, pass the `recordingReconnectWindowSeconds` parameter to enable merge fragmented individual participant recordings:

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration "storageConfigurationArn=arn:aws:ivs:us-
```

```
west-2:123456789012:storage-configuration/
LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO,thumbnailConfiguration="{targetIntervalSeconds=10,storage=[
```

The response is:

```
{
   "stage": {
      "arn": "arn:aws:ivs:us-west-2:123456789012:stage/VSWjvX5XOkU3",
      "autoParticipantRecordingConfiguration": {
         "hlsConfiguration": {
             "targetSegmentDurationSeconds": 6
         },
         "mediaTypes": [
            "AUDIO_VIDEO"
         ],
         "recordingReconnectWindowSeconds": 60,
         "recordParticipantReplicas": true,
         "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/LKZ6QR7r55c2",
         "thumbnailConfiguration": {
            "recordingMode": "INTERVAL",
            "storage": [
               "SEQUENTIAL",
               "LATEST"
            ],
            "targetIntervalSeconds": 10
         }
      },
      "endpoints": {
         "events": "<events-endpoint>",
         "rtmp": "<rtmp-endpoint>",
         "rtmps": "<rtmps-endpoint>",
         "whip": "<whip-endpoint>"
      },
      "name": "test-stage-participant-recording"
   }
}
```

Step 3: Distribute Tokens

Now that you have a stage, create and distribute the tokens that clients use to join it. Each client needs a participant token to join a stage and send or receive video. Applications that use a `RealTimeConnection` also need a connection token.

A participant token authorizes a participant to join one specific stage and defines that participant's publish and subscribe capabilities. A connection token authorizes a shared network connection that a client can reuse while moving between stages in the same AWS account and Region. A connection token does not replace a participant token. The client needs a participant token for every stage that it joins.

There are two approaches to generating tokens:

- Create tokens with a key pair.
- Create tokens with the IVS real-time-streaming API.

Both of these approaches are described below.

Creating Tokens with a Key Pair

You can create participant tokens and connection tokens on your server application by signing JWTs with an ECDSA public/private key pair. Import the public key into IVS so IVS can verify the JWT signature when a client connects.

> **Important**
>
> IVS does not offer key expiry. If your private key is compromised, you must delete the old public key.

Create a New Key Pair

There are various ways to create a key pair. Below, we give two examples.

To create a new key pair in the console, follow these steps:

1. Open the Amazon IVS console. Choose your stage's region if you are not already on it.
2. In the left navigation menu, choose **Real-time streaming > Public keys**.

3. Choose **Create public key**. A **Create public key** dialog appears.
4. Follow the prompts and choose **Create**.
5. Amazon IVS generates a new key pair. The public key is imported as a public key resource and the private key is immediately made available for download. The public key can also be downloaded later if necessary.

   Amazon IVS generates the key on the client side and does not store the private key. ***Be sure you save the key; you cannot retrieve it later.***

To create a new P384 EC key pair with OpenSSL (you might have to install OpenSSL first), follow these steps. This process enables you to access both the private and public keys. You need the public key only if you want to test verification of your tokens.

```
openssl ecparam -name secp384r1 -genkey -noout -out priv.pem
openssl ec -in priv.pem -pubout -out public.pem
```

Now import your new public key, using the instructions below.

Import the Public Key

Once you have a key pair, you can import the public key into IVS. The private key is not needed by our system but is employed by you to sign tokens.

To import an existing public key with the console:

1. Open the Amazon IVS console. Choose your stage's region if you are not already on it.
2. In the left navigation menu, choose **Real-time streaming > Public keys**.
3. Choose **Import**. An **Import public key** dialog appears.
4. Follow the prompts and choose **Import**.
5. Amazon IVS imports your public key and generates a public key resource.

To import an existing public key with the CLI:

```
aws ivs-realtime import-public-key --public-key-material "`cat public.pem`" --region
 <aws-region>
```

You can omit `--region <aws-region>` if the region is in your local AWS configuration file.

Here is an example response:

```
{
    "publicKey": {
        "arn": "arn:aws:ivs:us-west-2:123456789012:public-key/f99cde61-c2b0-4df3-8941-
ca7d38acca1a",
        "fingerprint": "98:0d:1a:a0:19:96:1e:ea:0a:0a:2c:9a:42:19:2b:e7",
        "publicKeyMaterial": "-----BEGIN PUBLIC KEY-----
\nMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEVjYMV+P4ML6xemanCrtse/FDwsNnpYmS
\nS6vRV9Wx37mjwi02hObKuCJqpj7x0lpz0bHm5v1JBvdZYAd/r2LR5aChK+/GM2Wj
\nl8MG9NJIVFaw1u3bvjEjzTASSfS1BDX1\n-----END PUBLIC KEY-----\n",
        "tags": {}
    }
}
```

API Request

```
POST /ImportPublicKey HTTP/1.1
{
  "publicKeyMaterial": "<pem file contents>"
}
```

Participant Tokens

A participant token authorizes one participant to join one stage. It contains the stage ARN and ID, endpoints, optional participant attributes, and publish and subscribe capabilities.

Create Participant Tokens with a Key Pair

For details on working with JWTs and the supported libraries for signing tokens, visit jwt.io. On the jwt.io interface, you must enter your private key to sign tokens. The public key is needed only if you want to verify tokens.

All JWTs have three fields: header, payload, and signature.

The JSON schemas for the JWT's header and payload are described below. Alternatively you can copy a sample JSON from the IVS console. To get the header and payload JSON from the IVS console:

1. Open the Amazon IVS console. Choose your stage's region if you are not already on it.

2. In the left navigation menu, choose **Real-time streaming > Stages**.
3. Select the stage you want to use. Select **View details**.
4. In the **Participant tokens** section, select the drop-down next to **Create token**.
5. Select **Build token header and payload**.
6. Fill in the form and copy the JWT header and payload shown at the bottom of the popup.

Token Schema: Header

The header specifies:

- `alg` is the signing algorithm. This is ES384, an ECDSA signature algorithm that uses the SHA-384 hash algorithm.
- `typ` is the token type, JWT.
- `kid` is the ARN of the public key used to sign the token. It must be the same ARN returned from the GetPublicKey API request.

```
{
  "alg": "ES384",
  "typ": "JWT"
  "kid": "arn:aws:ivs:123456789012:us-east-1:public-key/abcdefg12345"
}
```

Token Schema: Payload

The payload contains data specific to IVS. All fields except `user_id` are mandatory.

- `RegisteredClaims` in the JWT specification are reserved claims that need to be provided for stage token to be valid:
  - `exp` (expiration time) is a Unix UTC timestamp for when the token expires. (A Unix timestamp is a numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.) The token is validated when the participant joins a stage. IVS provides tokens with a default 12-hour TTL, which we recommend; this can be extended to a maximum of 14 days from the issued at time (iat). This must be an integer type value.
  - `iat` (issued at time) is a Unix UTC timestamp for when the JWT was issued. (See the note for `exp` about Unix timestamps.) It must be an integer type value.

  - `jti` (JWT ID) is the participant ID used for tracking and referring to the participant to whom the token is granted. Every token must have a unique participant ID. It must be a case-sensitive string, up to 64 characters long, containing only alphanumeric, hyphen (-), and underscore (_) characters. No other special characters are allowed.
- `user_id` is an optional, customer-assigned name to help identify the token; this can be used to link a participant to a user in the customer's own systems. This should match the `userId` field in the CreateParticipantToken API request. It can be any UTF-8 encoded text and is a string of up to 128 characters. *This field is exposed to all stage participants and should not be used for personally identifying, confidential, or sensitive information.*
- `resource` is the ARN of the stage; for example, `arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ`.
- `topic` is the ID of the stage, which can be extracted from stage ARN. For example, if the stage ARN is `arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ`, the stage ID is `oRmLNwuCeMlQ`.
- `events_url` must be the events endpoint returned from the CreateStage or GetStage operation. We recommend that you cache this value at stage-creation time; the value can be cached for up to 14 days. An example value is `wss://global.events.live-video.net`.
- `whip_url` must be the WHIP endpoint returned from the CreateStage or GetStage operation. We recommend that you cache this value at stage-creation time; the value can be cached for up to 14 days. An example value is `https://453fdfd2ad24df.global-bm.whip.live-video.net`.
- `capabilities` specifies the capabilities of the token; valid values are `allow_publish` and `allow_subscribe`. For subscribe-only tokens, set only `allow_subscribe` to `true`.
- `attributes` is an optional field where you can specify application-provided attributes to encode into the token and attach to a stage. Map keys and values can contain UTF-8 encoded text. The maximum length of this field is 1 KB total. *This field is exposed to all stage participants and should not be used for personally identifying, confidential, or sensitive information.*
- `version` must be `1.0`.

```
{
  "exp": 1697322063,
  "iat": 1697149263,
  "jti": "Mx6clRRHODPy",
  "user_id": "<optional_customer_assigned_name>",
  "resource": "<stage_arn>",
  "topic": "<stage_id>",
```

```
  "events_url": "wss://global.events.live-video.net",
  "whip_url": "https://114ddfabadaf.global-bm.whip.live-video.net",
  "capabilities": {
    "allow_publish": true,
    "allow_subscribe": true
  },
  "attributes": {
    "optional_field_1": "abcd1234",
    "optional_field_2": "false"
  },
  "version": "1.0"
}
```

Token Schema: Signature

To create the signature, use the private key with the algorithm specified in the header (ES384) to sign the encoded header and encoded payload.

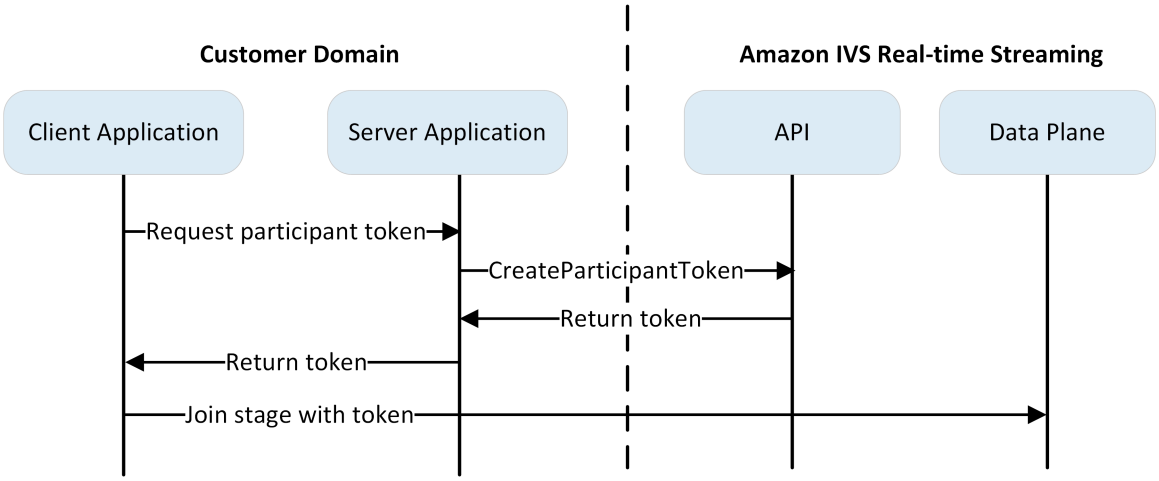
```
ECDSASHA384(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  <private-key>
)
```

Instructions

1. Generate the token's signature with an ES384 signing algorithm and a private key that is associated with the public key provided to IVS.
2. Assemble the token.

```
base64UrlEncode(header) + "." +
base64UrlEncode(payload) + "." +
base64UrlEncode(signature)
```

Creating Tokens with the IVS Real-Time Streaming API



As shown above, a client application asks your server application for a token, and the server application calls `CreateParticipantToken` using an AWS SDK or SigV4 signed request. Since AWS credentials are used to call the API, the token should be generated in a secure server-side application, not the client-side application.

When creating a participant token, you can optionally specify attributes and/or capabilities:

- You can specify application-provided attributes to encode into the token and attach to a stage. Map keys and values can contain UTF-8 encoded text. The maximum length of this field is 1 KB total. *This field is exposed to all stage participants and should not be used for personally identifying, confidential, or sensitive information.*
- You can specify capabilities enabled by the token. The default is `PUBLISH` and `SUBSCRIBE`, which allows the participant to send and receive audio and video, but you could issue tokens with a subset of capabilities. For example, you could issue a token with only the `SUBSCRIBE` capability for moderators. In that case, the moderators could see the participants that are sending video but not send their own video.

For details, see CreateParticipantToken.

You can create participant tokens through the console or CLI for testing and development, but most likely you will want to create them with the AWS SDK in your production environment.

You will need a way to distribute tokens from your server to each client (for example, through an API request). We do not provide this functionality. For this guide, you can simply copy and paste the tokens into client code in the following steps.

Important: Treat tokens as opaque; do not build functionality based on token contents. The format of tokens could change in the future.

Console Instructions

1. Navigate to the stage you created in the prior step.
2. Select **Create token**. The **Create token** window appears.
3. Enter a user ID to be associated with the token. This can be any UTF-8 encoded text.
4. Select **Create**.
5. Copy the token. *Important: Be sure to save the token; IVS does not store it and you cannot retrieve it later.*

CLI Instructions

Creating a token with the AWS CLI requires that you first download and configure the CLI on your machine. For details, see the AWS Command Line Interface User Guide. Note that generating tokens with the AWS CLI is good for testing purposes, but for production use, we recommend that you generate tokens on the server side with the AWS SDK (see instructions below).

1. Run the `create-participant-token` command with the stage ARN. Include any or all of the following capabilities: `"PUBLISH"`, `"SUBSCRIBE"`.

```
aws ivs-realtime create-participant-token --stage-arn arn:aws:ivs:us-
west-2:123456789012:stage/VSWjvX5XOkU3 --capabilities '["PUBLISH", "SUBSCRIBE"]'
```

2. This returns a participant token:

```
{
    "participantToken": {
        "capabilities": [
            "PUBLISH",
            "SUBSCRIBE"
        ],
        "expirationTime": "2023-06-03T07:04:31+00:00",
        "participantId": "tU06DT5jCJeb",
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJ
TaKjllW9Qac6c5xBrdAk"
    }
```

```
}
```

3. Save this token. You will need it to join the stage and send and receive video.

AWS SDK Instructions

You can use the AWS SDK to create tokens. Below are instructions for the AWS SDK using JavaScript.

Important: This code must be executed on the server side and its output passed to the client.

Prerequisite: To use the code sample below, you need to install the aws-sdk/client-ivs-realtime package. For details, see Getting started with the AWS SDK for JavaScript.

```
import { IVSRealTimeClient, CreateParticipantTokenCommand } from "@aws-sdk/client-ivs-
realtime";

const ivsRealtimeClient = new IVSRealTimeClient({ region: 'us-west-2' });
const stageArn = 'arn:aws:ivs:us-west-2:123456789012:stage/VSWjvX5XOkU3';
const createStageTokenRequest = new CreateParticipantTokenCommand({
  stageArn,
});
const response = await ivsRealtimeClient.send(createStageTokenRequest);
console.log('token', response.participantToken.token);
```

Connection Tokens

A connection token is a self-signed JWT that authorizes a shared network connection. A client can reuse this connection while moving between stages in the same AWS account and Region, until the token expires.

Create and sign connection tokens on your server with the key pair created earlier. Send the connection token to the client before it begins a workflow that moves between stages. The client creates one `RealTimeConnection` and supplies it whenever it creates a stage. The client then uses the appropriate participant token for each stage that it joins.

Token Header

Use the token header described in Create Participant Tokens with a Key Pair.

Token Payload

```
{
    "exp": 1697322063,
    "iat": 1697149263,
    "jti": "Mx6clRRHODPy",
    "account_id": "123456789012",
    "region": "us-west-2",
    "events_url": "wss://global.events.live-video.net",
    "version": "1.0"
}
```

The payload contains data specific to IVS. All fields are mandatory:

- `RegisteredClaims` in the JWT specification are reserved claims that must be present for the token to be valid:
  - `exp` (expiration time) is a Unix UTC timestamp for when the token expires. A token can expire up to four weeks after its creation time.
  - `iat` (issued-at time) is a Unix UTC timestamp for when the JWT was issued.
  - `jti` (JWT ID) is a unique token ID. Generate a new, case-sensitive ID for every Connection token. The ID can contain up to 64 alphanumeric characters, hyphens (-), and underscores (_).
- `account_id` is the AWS account ID that owns the stages.
- `region` is the home Region of the IVS public key used to sign the token.
- `events_url` must be `wss://global.events.live-video.net`.
- `version` must be `1.0`.

A connection token does not contain a stage ARN, stage ID, WHIP endpoint, participant capabilities, participant attributes, or user ID. Those values belong in the participant token for the stage that the client joins.

Sign the token as described in Create Participant Tokens with a Key Pair, using the connection token header and payload above.

Step 4: Integrate the IVS Broadcast SDK

IVS provides a broadcast SDK for web, Android, and iOS that you can integrate into your application. The broadcast SDK is used for both sending and receiving video. If you have configured

RTMP Ingest for your stage, you may use any encoder that can broadcast to an RTMP endpoint (e.g., OBS or ffmpeg).

In this section, we write a simple application that enables two or more participants to interact in real time. The steps below guide you through creating an app called BasicRealTime. The full app code is on CodePen and GitHub:

- Web: https://codepen.io/amazon-ivs/pen/ZEqgrpo
- Android: https://github.com/aws-samples/amazon-ivs-real-time-streaming-android-samples
- iOS: https://github.com/aws-samples/amazon-ivs-real-time-streaming-ios-samples

Web

Set Up Files

To start, set up your files by creating a folder and an initial HTML and JS file:

```
mkdir realtime-web-example
cd realtime-web-example
touch index.html
touch app.js
```

You can install the broadcast SDK using a script tag or npm. Our example uses the script tag for simplicity but is easy to modify if you choose to use npm later.

Using a Script Tag

The Web broadcast SDK is distributed as a JavaScript library and can be retrieved at https://web-broadcast.live-video.net/1.39.0/amazon-ivs-web-broadcast.js.

When loaded via `<script>` tag, the library exposes a global variable in the window scope named `IVSBroadcastClient`.

Using npm

To install the npm package:

```
npm install amazon-ivs-web-broadcast
```

You can now access the IVSBroadcastClient object:

```
const { Stage } = IVSBroadcastClient;
```

Android

Create the Android Project

1. In Android Studio, create a **New Project**.
2. Choose **Empty Views Activity**.

   Note: In some older versions of Android Studio, the View-based activity is called **Empty Activity**. If your Android Studio window shows **Empty Activity** and does *not* show **Empty Views** Activity, select **Empty Activity**. Otherwise, don't select **Empty Activity**, since we'll be using View APIs (not Jetpack Compose).
3. Give your project a **Name**, then select **Finish**.

Install the Broadcast SDK

To add the Amazon IVS Android broadcast library to your Android development environment, add the library to your module's `build.gradle` file, as shown here (for the latest version of the Amazon IVS broadcast SDK). In newer projects the `mavenCentral` repository may already be included in your `settings.gradle` file, if that is the case you can omit the `repositories` block. For our sample, we'll also need to enable data binding in the `android` block.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

dependencies {
     implementation 'com.amazonaws:ivs-broadcast:1.46.0:stages@aar'
}
```

Alternately, to install the SDK manually, download the latest version from this location:

https://search.maven.org/artifact/com.amazonaws/ivs-broadcast

iOS

Create the iOS Project

1. Create a new Xcode project.
2. For **Platform**, select **iOS**.
3. For **Application**, select **App**.
4. Enter the **Product Name** of your app, then select **Next**.
5. Choose (navigate to) a directory in which to save the project, then select **Create**.

Next you need to bring in the SDK. For instructions, see Install the Library in the *iOS Broadcast SDK Guide*.

Configure Permissions

You need to update your project's `Info.plist` to add two new entries for `NSCameraUsageDescription` and `NSMicrophoneUsageDescription`. For the values, provide user-facing explanations of why your app is asking for camera and microphone access.

Key	Type	Value
Information Property List	Dictionary	(3 items)
Application Scene Manifest	Dictionary	(2 items)
Privacy - Microphone Usage Description	String	We need access to your microphone to publish your audio feed
Privacy - Camera Usage Description	String	We need access to your camera to publish your video feed

Step 5: Publish and Subscribe to Video

You can publish/subscribe (real-time) to IVS with:

- The native IVS broadcast SDKs, which support WebRTC and RTMPS. We recommend this, especially for production scenarios. See the details below for Web, Android, and iOS.
- The Amazon IVS console — This is suitable for testing streams. See below.
- Other streaming software and hardware encoders — You can use any streaming encoder that supports the RTMP, RTMPS, or WHIP protocols. See Stream Ingest for more information.

IVS Console

1. Open the Amazon IVS console.

(You can also access the Amazon IVS console through the AWS Management Console.)

2. In the navigation pane, select **Stages**. (If the nav pane is collapsed, expand it by selecting the hamburger icon.)
3. Select the stage to which you want to subscribe or publish, to go to its details page.
4. To subscribe: If the stage has one or more publishers, you can subscribe to it by pressing the **Subscribe** button, under the **Subscribe** tab. (Tabs are below the **General Configuration** section.)
5. To publish:
   a. Select the **Publish** tab.
   b. You will be prompted to grant the IVS console access to your camera and microphone; **Allow** those permissions.
   c. Toward the bottom of the **Publish** tab, use the dropdown boxes to select input devices for the microphone and camera.
   d. To begin publishing, select **Start publishing**.
   e. To view your published content, go back to the **Subscribe** tab.
   f. To stop publishing, go to the **Publish** tab and press the **Stop publishing** button towards the bottom.

Note: Subscribing and publishing consume resources, and you will incur an hourly rate for the time you are connected to the stage. To learn more, see Real-Time Streaming on the IVS Pricing page.

Publish & Subscribe with the IVS Web Broadcast SDK

This section takes you through the steps involved in publishing and subscribing to a stage using your web app.

Create HTML Boilerplate

First let's create the HTML boilerplate and import the library as a script tag:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Import the SDK -->
  <script src="https://web-broadcast.live-video.net/1.39.0/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>

<!-- TODO - fill in with next sections -->
<script src="./app.js"></script>

</body>
</html>
```

Accept Token Input and Add Join/Leave Buttons

Here we fill in the body with our input controls. These take as input the token, and they set up **Join** and **Leave** buttons. Typically applications will request the token from your application's API, but for this example you'll copy and paste the token into the token input.

```
<h1>IVS Real-Time Streaming</h1>
<hr />

<label for="token">Token</label>
<input type="text" id="token" name="token" />
<button class="button" id="join-button">Join</button>
<button class="button" id="leave-button" style="display: none;">Leave</button>
<hr />
```

Add Media Container Elements

These elements will hold the media for our local and remote participants. We add a script tag to load our application's logic defined in `app.js`.

```
<!-- Local Participant -->
<div id="local-media"></div>

<!-- Remote Participants -->
<div id="remote-media"></div>
```

```
<!-- Load Script -->
<script src="./app.js"></script>
```

This completes the HTML page and you should see this when loading `index.html` in a browser:

Create app.js

Let's move to defining the contents of our `app.js` file. Begin by importing all the requisite properties from the SDK's global:

```
const {
  Stage,
  LocalStageStream,
  SubscribeType,
  StageEvents,
  ConnectionState,
  StreamType
} = IVSBroadcastClient;
```

Create Application Variables

Establish variables to hold references to our **Join** and **Leave** button HTML elements and store state for the application:

```
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
```

```
let micStageStream;
```

Create joinStage 1: Define the Function and Validate Input

The `joinStage` function takes the input token, creates a connection to the stage, and begins to publish video and audio retrieved from `getUserMedia`.

To start, we define the function and validate the state and token input. We'll flesh out this function in the next few sections.

```
const joinStage = async () => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById("token").value;

  if (!token) {
    window.alert("Please enter a participant token");
    joining = false;
    return;
  }

  // Fill in with the next sections
};
```

Create joinStage 2: Get Media to Publish

Here is the media that will be published to the stage:

```
async function getCamera() {
  // Use Max Width and Height
  return navigator.mediaDevices.getUserMedia({
    video: true,
    audio: false
  });
}

async function getMic() {
  return navigator.mediaDevices.getUserMedia({
```

```
    video: false,
    audio: true
  });
}

// Retrieve the User Media currently set on the page
localCamera = await getCamera();
localMic = await getMic();

// Create StageStreams for Audio and Video
cameraStageStream = new LocalStageStream(localCamera.getVideoTracks()[0]);
micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);
```

Create joinStage 3: Define the Stage Strategy and Create the Stage

This stage strategy is the heart of the decision logic that the SDK uses to decide what to publish and which participants to subscribe to. For more information on the function's purpose, see Strategy.

This strategy is simple. After joining the stage, publish the streams we just retrieved and subscribe to every remote participant's audio and video:

```
const strategy = {
  stageStreamsToPublish() {
    return [cameraStageStream, micStageStream];
  },
  shouldPublishParticipant() {
    return true;
  },
  shouldSubscribeToParticipant() {
    return SubscribeType.AUDIO_VIDEO;
  }
};

stage = new Stage(token, strategy);
```

Create joinStage 4: Handle Stage Events and Render Media

Stages emit many events. We'll need to listen to the `STAGE_PARTICIPANT_STREAMS_ADDED` and `STAGE_PARTICIPANT_LEFT` to render and remove media to and from the page. A more exhaustive set of events are listed in Events.

Note that we create four helper functions here to assist us in managing necessary DOM elements: `setupParticipant`, `teardownParticipant`, `createVideoEl`, and `createContainer`.

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
  connected = state === ConnectionState.CONNECTED;

  if (connected) {
    joining = false;
    joinButton.style = "display: none";
    leaveButton.style = "display: inline-block";
  }
});

stage.on(
  StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED,
  (participant, streams) => {
    console.log("Participant Media Added: ", participant, streams);

    let streamsToDisplay = streams;

    if (participant.isLocal) {
      // Ensure to exclude local audio streams, otherwise echo will occur
      streamsToDisplay = streams.filter(
        (stream) => stream.streamType === StreamType.VIDEO
      );
    }

    const videoEl = setupParticipant(participant);
    streamsToDisplay.forEach((stream) =>
      videoEl.srcObject.addTrack(stream.mediaStreamTrack)
    );
  }
);

stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
  console.log("Participant Left: ", participant);
  teardownParticipant(participant);
});


// Helper functions for managing DOM

function setupParticipant({ isLocal, id }) {
```

```
  const groupId = isLocal ? "local-media" : "remote-media";
  const groupContainer = document.getElementById(groupId);

  const participantContainerId = isLocal ? "local" : id;
  const participantContainer = createContainer(participantContainerId);
  const videoEl = createVideoEl(participantContainerId);

  participantContainer.appendChild(videoEl);
  groupContainer.appendChild(participantContainer);

  return videoEl;
}

function teardownParticipant({ isLocal, id }) {
  const groupId = isLocal ? "local-media" : "remote-media";
  const groupContainer = document.getElementById(groupId);
  const participantContainerId = isLocal ? "local" : id;

  const participantDiv = document.getElementById(
    participantContainerId + "-container"
  );
  if (!participantDiv) {
    return;
  }
  groupContainer.removeChild(participantDiv);
}

function createVideoEl(id) {
  const videoEl = document.createElement("video");
  videoEl.id = id;
  videoEl.autoplay = true;
  videoEl.playsInline = true;
  videoEl.srcObject = new MediaStream();
  return videoEl;
}

function createContainer(id) {
  const participantContainer = document.createElement("div");
  participantContainer.classList = "participant-container";
  participantContainer.id = id + "-container";

  return participantContainer;
}
```

Create joinStage 5: Join the Stage

Let's complete our `joinStage` function by finally joining the stage!

```
try {
  await stage.join();
} catch (err) {
  joining = false;
  connected = false;
  console.error(err.message);
}
```

Create leaveStage

Define the `leaveStage` function which the leave button will invoke.

```
const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;
};
```

Initialize Input-Event Handlers

We'll add one last function to our `app.js` file. This function is invoked immediately when the page loads and establishes event handlers for joining and leaving the stage.

```
const init = async () => {
  try {
    // Prevents issues on Safari/FF so devices are not blank
    await navigator.mediaDevices.getUserMedia({ video: true, audio: true });
  } catch (e) {
    alert(
      "Problem retrieving media! Enable camera and microphone permissions."
    );
  }

  joinButton.addEventListener("click", () => {
    joinStage();
  });
```

```
  leaveButton.addEventListener("click", () => {
    leaveStage();
    joinButton.style = "display: inline-block";
    leaveButton.style = "display: none";
  });
};

init(); // call the function
```

Run the Application and Provide a Token

At this point you can share the web page locally or with others, open the page, and put in a participant token and join the stage.

What's Next?

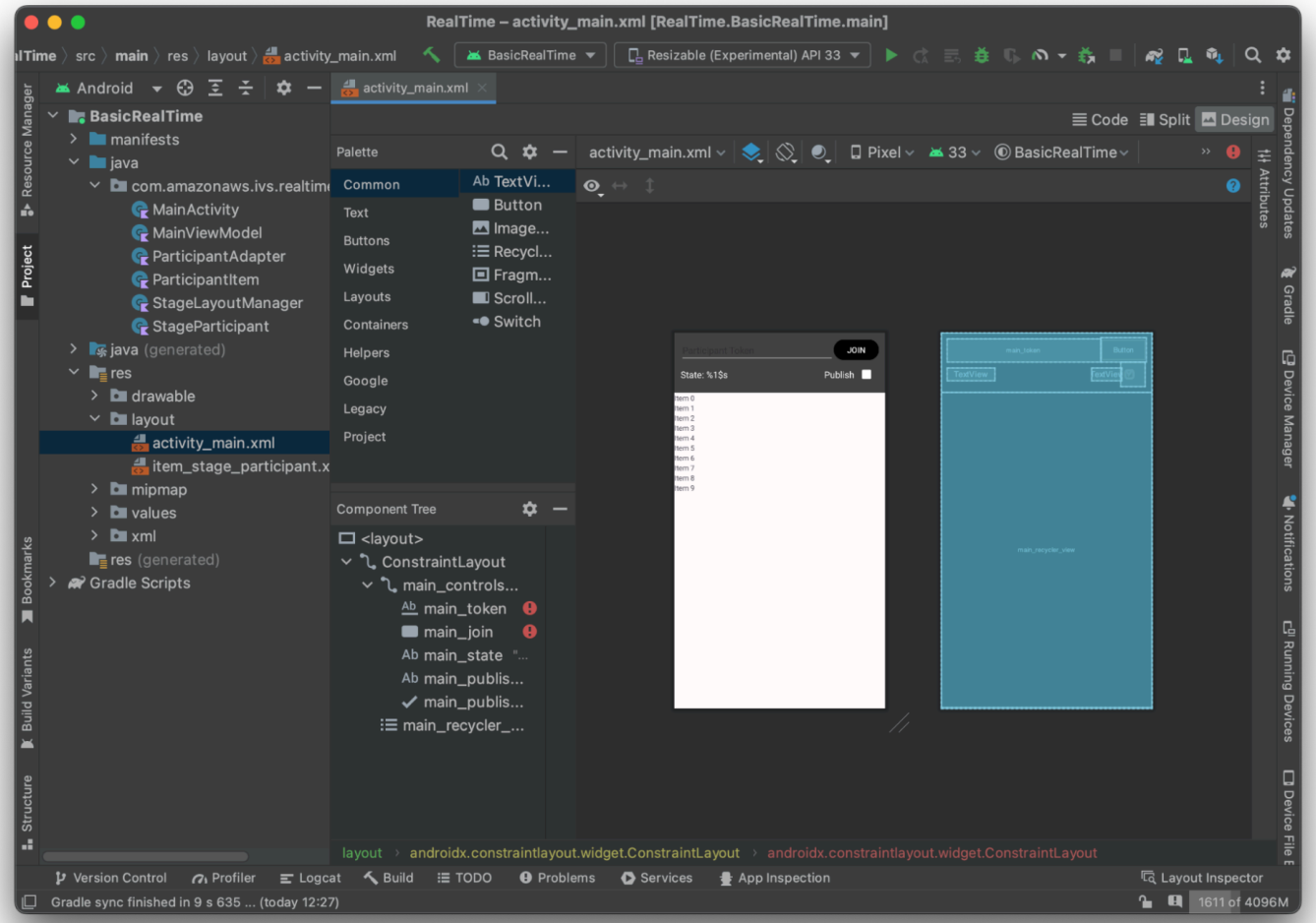
For more detailed examples involving npm, React, and more, see the IVS Broadcast SDK: Web Guide (Real-Time Streaming Guide).

Publish & Subscribe with the IVS Android Broadcast SDK

This section takes you through the steps involved in publishing and subscribing to a stage using your Android app.

Create Views

We start by creating a simple layout for our app using the auto-created `activity_main.xml` file. The layout contains an `EditText` to add a token, a Join `Button`, a `TextView` to show the stage state, and a `CheckBox` to toggle publishing.



Here is the XML behind the view:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:keepScreenOn="true"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".BasicActivity">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:id="@+id/main_controls_container"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:background="@color/cardview_dark_background"
            android:padding="12dp"
            app:layout_constraintTop_toTopOf="parent">

            <EditText
                android:id="@+id/main_token"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:autofillHints="@null"
                android:backgroundTint="@color/white"
                android:hint="@string/token"
                android:imeOptions="actionDone"
                android:inputType="text"
                android:textColor="@color/white"
                app:layout_constraintEnd_toStartOf="@id/main_join"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

            <Button
                android:id="@+id/main_join"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:backgroundTint="@color/black"
                android:text="@string/join"
                android:textAllCaps="true"
                android:textColor="@color/white"
                android:textSize="16sp"
                app:layout_constraintBottom_toBottomOf="@+id/main_token"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toEndOf="@id/main_token" />

            <TextView
                android:id="@+id/main_state"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/state"
                android:textColor="@color/white"
                android:textSize="18sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@id/main_token" />

            <TextView
```

```
                android:id="@+id/main_publish_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/publish"
                android:textColor="@color/white"
                android:textSize="18sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toStartOf="@id/main_publish_checkbox"
                app:layout_constraintTop_toBottomOf="@id/main_token" />

            <CheckBox
                android:id="@+id/main_publish_checkbox"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:buttonTint="@color/white"
                android:checked="true"
                app:layout_constraintBottom_toBottomOf="@id/main_publish_text"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintTop_toTopOf="@id/main_publish_text" />

        </androidx.constraintlayout.widget.ConstraintLayout>

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/main_recycler_view"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            app:layout_constraintTop_toBottomOf="@+id/main_controls_container"
            app:layout_constraintBottom_toBottomOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
<layout>
```

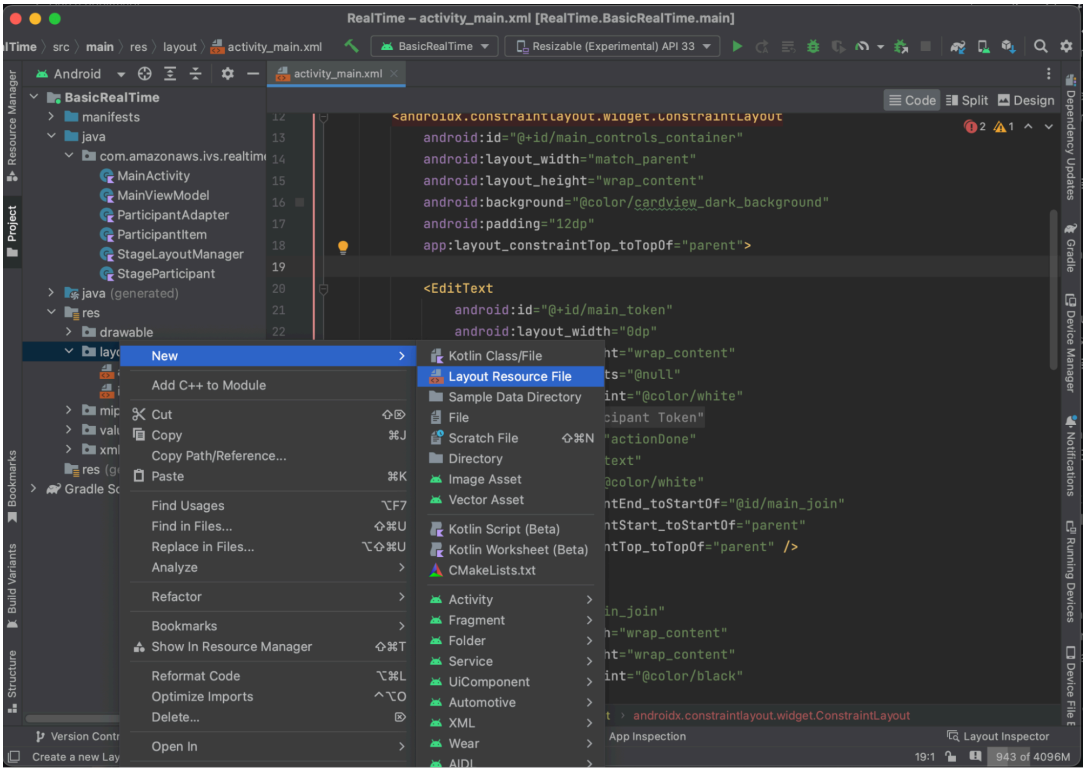
We referenced a couple of string IDs here, so we'll create our entire `strings.xml` file now:

```
<resources>
    <string name="app_name">BasicRealTime</string>
    <string name="join">Join</string>
    <string name="leave">Leave</string>
    <string name="token">Participant Token</string>
    <string name="publish">Publish</string>
    <string name="state">State: %1$s</string>
</resources>
```

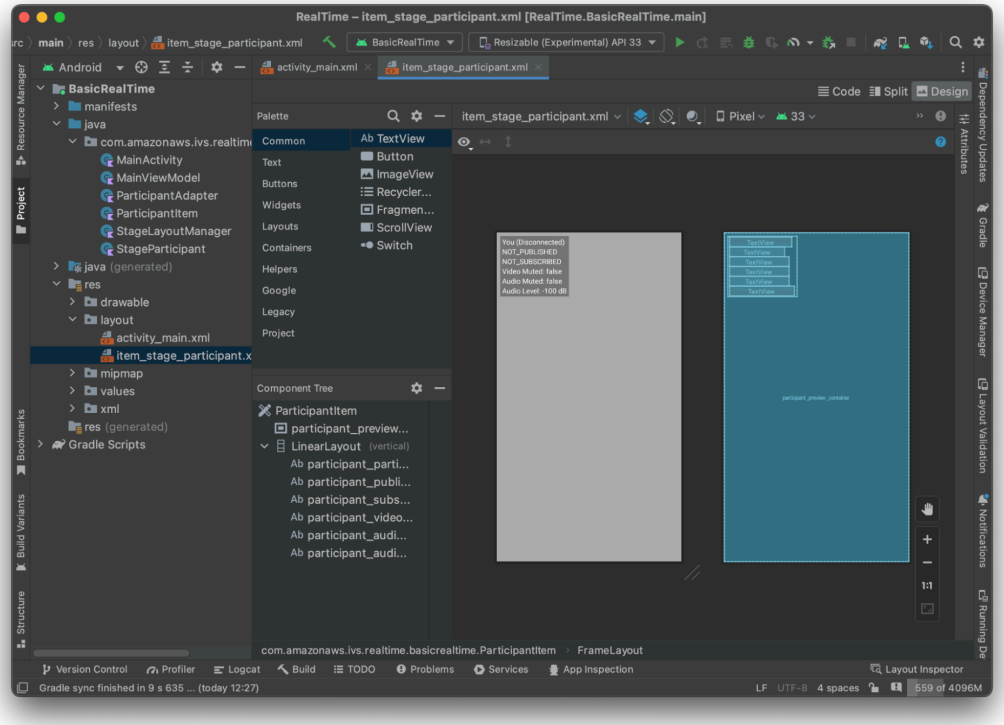
Let's link those views in the XML to our `MainActivity.kt`:

```
import android.widget.Button
import android.widget.CheckBox
import android.widget.EditText
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

private lateinit var checkboxPublish: CheckBox
private lateinit var recyclerView: RecyclerView
private lateinit var buttonJoin: Button
private lateinit var textViewState: TextView
private lateinit var editTextToken: EditText

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    checkboxPublish = findViewById(R.id.main_publish_checkbox)
    recyclerView = findViewById(R.id.main_recycler_view)
    buttonJoin = findViewById(R.id.main_join)
    textViewState = findViewById(R.id.main_state)
    editTextToken = findViewById(R.id.main_token)
}
```

Now we create an item view for our `RecyclerView`. To do this, right-click your `res/layout` directory and select **New > Layout Resource File**. Name this new file `item_stage_participant.xml`.



The layout for this item is simple: it contains a view for rendering a participant's video stream and a list of labels for displaying information about the participant:



Here is the XML:

```
<?xml version="1.0" encoding="utf-8"?>
<com.amazonaws.ivs.realtime.basicrealtime.ParticipantItem xmlns:android="http://
schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/participant_preview_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:background="@android:color/darker_gray" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="#50000000"
        android:orientation="vertical"
        android:paddingLeft="4dp"
        android:paddingTop="2dp"
        android:paddingRight="4dp"
        android:paddingBottom="2dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/participant_participant_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="You (Disconnected)" />

        <TextView
            android:id="@+id/participant_publishing"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
```

```
            tools:text="NOT_PUBLISHED" />

        <TextView
            android:id="@+id/participant_subscribed"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="NOT_SUBSCRIBED" />

        <TextView
            android:id="@+id/participant_video_muted"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="Video Muted: false" />

        <TextView
            android:id="@+id/participant_audio_muted"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="Audio Muted: false" />

        <TextView
            android:id="@+id/participant_audio_level"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="Audio Level: -100 dB" />

    </LinearLayout>

</com.amazonaws.ivs.realtime.basicrealtime.ParticipantItem>
```

This XML file inflates a class we haven't created yet, `ParticipantItem`. Because the XML includes the full namespace, be sure to update this XML file to your namespace. Let's create this class and set up the views, but otherwise leave it blank for now.

Create a new Kotlin class, `ParticipantItem`:

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.content.Context
import android.util.AttributeSet
import android.widget.FrameLayout
import android.widget.TextView
import kotlin.math.roundToInt

class ParticipantItem @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0,
    defStyleRes: Int = 0,
) : FrameLayout(context, attrs, defStyleAttr, defStyleRes) {

    private lateinit var previewContainer: FrameLayout
    private lateinit var textViewParticipantId: TextView
    private lateinit var textViewPublish: TextView
    private lateinit var textViewSubscribe: TextView
    private lateinit var textViewVideoMuted: TextView
    private lateinit var textViewAudioMuted: TextView
    private lateinit var textViewAudioLevel: TextView

    override fun onFinishInflate() {
        super.onFinishInflate()
        previewContainer = findViewById(R.id.participant_preview_container)
        textViewParticipantId = findViewById(R.id.participant_participant_id)
        textViewPublish = findViewById(R.id.participant_publishing)
        textViewSubscribe = findViewById(R.id.participant_subscribed)
        textViewVideoMuted = findViewById(R.id.participant_video_muted)
        textViewAudioMuted = findViewById(R.id.participant_audio_muted)
        textViewAudioLevel = findViewById(R.id.participant_audio_level)
    }
}
```

Permissions

To use the camera and microphone, you need to request permissions from the user. We follow a standard permissions flow for this:

```
override fun onStart() {
    super.onStart()
    requestPermission()
```

```
}

private val requestPermissionLauncher =
    registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions())
 { permissions ->
        if (permissions[Manifest.permission.CAMERA] == true &&
 permissions[Manifest.permission.RECORD_AUDIO] == true) {
            viewModel.permissionGranted() // we will add this later
        }
    }

private val permissions = listOf(
    Manifest.permission.CAMERA,
    Manifest.permission.RECORD_AUDIO,
)

private fun requestPermission() {
    when {
        this.hasPermissions(permissions) -> viewModel.permissionGranted() // we will
 add this later
        else -> requestPermissionLauncher.launch(permissions.toTypedArray())
    }
}

private fun Context.hasPermissions(permissions: List<String>): Boolean {
    return permissions.all {
        ContextCompat.checkSelfPermission(this, it) ==
 PackageManager.PERMISSION_GRANTED
    }
}
```

App State

Our application keeps track of the participants locally in a `MainViewModel.kt` and the state will be communicated back to the `MainActivity` using Kotlin's StateFlow.

Create a new Kotlin class `MainViewModel`:

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.app.Application
import androidx.lifecycle.AndroidViewModel
```

```
class MainViewModel(application: Application) : AndroidViewModel(application),
 Stage.Strategy, StageRenderer {

}
```

In `MainActivity.kt` we manage our view model:

```
import androidx.activity.viewModels

private val viewModel: MainViewModel by viewModels()
```

To use `AndroidViewModel` and these Kotlin `ViewModel` extensions, you'll need to add the following to your module's `build.gradle` file:

```
implementation 'androidx.core:core-ktx:1.10.1'
implementation "androidx.activity:activity-ktx:1.7.2"
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'com.google.android.material:material:1.10.0'
implementation "androidx.lifecycle:lifecycle-extensions:2.2.0"

def lifecycle_version = "2.6.1"
implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
```

RecyclerView Adapter

We'll create a simple `RecyclerView.Adapter` subclass to keep track of our participants and update our `RecyclerView` on stage events. But first, we need a class that represents a participant. Create a new Kotlin class `StageParticipant`:

```
package com.amazonaws.ivs.realtime.basicrealtime

import com.amazonaws.ivs.broadcast.Stage
import com.amazonaws.ivs.broadcast.StageStream

class StageParticipant(val isLocal: Boolean, var participantId: String?) {
    var publishState = Stage.PublishState.NOT_PUBLISHED
    var subscribeState = Stage.SubscribeState.NOT_SUBSCRIBED
    var streams = mutableListOf<StageStream>()
```

```
    val stableID: String
        get() {
            return if (isLocal) {
                "LocalUser"
            } else {
                requireNotNull(participantId)
            }
        }
}
```

We'll use this class in the `ParticipantAdapter` class that we'll create next. We start by defining the class and creating a variable to track the participants:

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView

class ParticipantAdapter : RecyclerView.Adapter<ParticipantAdapter.ViewHolder>() {

    private val participants = mutableListOf<StageParticipant>()
```

We also have to define our `RecyclerView.ViewHolder` before implementing the rest of the overrides:

```
class ViewHolder(val participantItem: ParticipantItem) :
 RecyclerView.ViewHolder(participantItem)
```

Using this, we can implement the standard `RecyclerView.Adapter` overrides:

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val item = LayoutInflater.from(parent.context)
        .inflate(R.layout.item_stage_participant, parent, false) as ParticipantItem
    return ViewHolder(item)
}

override fun getItemCount(): Int {
    return participants.size
}

override fun getItemId(position: Int): Long =
```

```
    participants[position]
        .stableID
        .hashCode()
        .toLong()

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    return holder.participantItem.bind(participants[position])
}

override fun onBindViewHolder(holder: ViewHolder, position: Int, payloads:
 MutableList<Any>) {
    val updates = payloads.filterIsInstance<StageParticipant>()
    if (updates.isNotEmpty()) {
        updates.forEach { holder.participantItem.bind(it) // implemented later }
    } else {
        super.onBindViewHolder(holder, position, payloads)
    }
}
```

Finally, we add new methods that we will call from our `MainViewModel` when changes to participants are made. These methods are standard CRUD operations on the adapter.

```
fun participantJoined(participant: StageParticipant) {
    participants.add(participant)
    notifyItemInserted(participants.size - 1)
}

fun participantLeft(participantId: String) {
    val index = participants.indexOfFirst { it.participantId == participantId }
    if (index != -1) {
        participants.removeAt(index)
        notifyItemRemoved(index)
    }
}

fun participantUpdated(participantId: String?, update: (participant: StageParticipant)
 -> Unit) {
    val index = participants.indexOfFirst { it.participantId == participantId }
    if (index != -1) {
        update(participants[index])
        notifyItemChanged(index, participants[index])
    }
}
```

Back in `MainViewModel` we need to create and hold a reference to this adapter:

```
internal val participantAdapter = ParticipantAdapter()
```

Stage State

We also need to track some stage state within `MainViewModel`. Let's define those properties now:

```
private val _connectionState = MutableStateFlow(Stage.ConnectionState.DISCONNECTED)
val connectionState = _connectionState.asStateFlow()

private var publishEnabled: Boolean = false
    set(value) {
        field = value
        // Because the strategy returns the value of `checkboxPublish.isChecked`, just
 call `refreshStrategy`.
        stage?.refreshStrategy()
    }

private var deviceDiscovery: DeviceDiscovery? = null
private var stage: Stage? = null
private var streams = mutableListOf<LocalStageStream>()
```

To see your own preview before joining a stage, we create a local participant immediately:

```
init {
    deviceDiscovery = DeviceDiscovery(application)

    // Create a local participant immediately to render our camera preview and
 microphone stats
    val localParticipant = StageParticipant(true, null)
    participantAdapter.participantJoined(localParticipant)
}
```

We want to make sure we clean up these resources when our `ViewModel` is cleaned up. We override `onCleared()` right away, so we don't forget to clean these resources.

```
override fun onCleared() {
    stage?.release()
    deviceDiscovery?.release()
    deviceDiscovery = null
    super.onCleared()
```

```
}
```

Now we populate our local `streams` property as soon as permissions are granted, implementing the `permissionsGranted` method that we called earlier:

```
internal fun permissionGranted() {
    val deviceDiscovery = deviceDiscovery ?: return
    streams.clear()
    val devices = deviceDiscovery.listLocalDevices()
    // Camera
    devices
        .filter { it.descriptor.type == Device.Descriptor.DeviceType.CAMERA }
        .maxByOrNull { it.descriptor.position == Device.Descriptor.Position.FRONT }
        ?.let { streams.add(ImageLocalStageStream(it)) }
    // Microphone
    devices
        .filter { it.descriptor.type == Device.Descriptor.DeviceType.MICROPHONE }
        .maxByOrNull { it.descriptor.isDefault }
        ?.let { streams.add(AudioLocalStageStream(it)) }

    stage?.refreshStrategy()

    // Update our local participant with these new streams
    participantAdapter.participantUpdated(null) {
        it.streams.clear()
        it.streams.addAll(streams)
    }
}
```

Implementing the Stage SDK

Three core concepts underlie real-time functionality: stage, strategy, and renderer. The design goal is minimizing the amount of client-side logic necessary to build a working product.

Stage.Strategy

Our `Stage.Strategy` implementation is simple:

```
override fun stageStreamsToPublishForParticipant(
    stage: Stage,
    participantInfo: ParticipantInfo
): MutableList<LocalStageStream> {
    // Return the camera and microphone to be published.
```

```
    // This is only called if `shouldPublishFromParticipant` returns true.
    return streams
}

override fun shouldPublishFromParticipant(stage: Stage, participantInfo:
 ParticipantInfo): Boolean {
    return publishEnabled
}

override fun shouldSubscribeToParticipant(stage: Stage, participantInfo:
 ParticipantInfo): Stage.SubscribeType {
    // Subscribe to both audio and video for all publishing participants.
    return Stage.SubscribeType.AUDIO_VIDEO
}
```

To summarize, we publish based on our internal `publishEnabled` state, and if we publish we will publish the streams we collected earlier. Finally for this sample, we always subscribe to other participants, receiving both their audio and video.

StageRenderer

The `StageRenderer` implementation also is fairly simple, though given the number of functions it contains quite a bit more code. The general approach in this renderer is to update our `ParticipantAdapter` when the SDK notifies us of a change to a participant. There are certain scenarios where we handle local participants differently, because we have decided to manage them ourselves so they can see their camera preview before joining.

```
override fun onError(exception: BroadcastException) {
    Toast.makeText(getApplication(), "onError ${exception.localizedMessage}",
 Toast.LENGTH_LONG).show()
    Log.e("BasicRealTime", "onError $exception")
}

override fun onConnectionStateChanged(
    stage: Stage,
    connectionState: Stage.ConnectionState,
    exception: BroadcastException?
) {
    _connectionState.value = connectionState
}

override fun onParticipantJoined(stage: Stage, participantInfo: ParticipantInfo) {
```

```
    if (participantInfo.isLocal) {
        // If this is the local participant joining the stage, update the participant
 with a null ID because we
        // manually added that participant when setting up our preview
        participantAdapter.participantUpdated(null) {
            it.participantId = participantInfo.participantId
        }
    } else {
        // If they are not local, add them normally
        participantAdapter.participantJoined(
            StageParticipant(
                participantInfo.isLocal,
                participantInfo.participantId
            )
        )
    }
}

override fun onParticipantLeft(stage: Stage, participantInfo: ParticipantInfo) {
    if (participantInfo.isLocal) {
        // If this is the local participant leaving the stage, update the ID but keep
 it around because
        // we want to keep the camera preview active
        participantAdapter.participantUpdated(participantInfo.participantId) {
            it.participantId = null
        }
    } else {
        // If they are not local, have them leave normally
        participantAdapter.participantLeft(participantInfo.participantId)
    }
}

override fun onParticipantPublishStateChanged(
    stage: Stage,
    participantInfo: ParticipantInfo,
    publishState: Stage.PublishState
) {
    // Update the publishing state of this participant
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.publishState = publishState
    }
}

override fun onParticipantSubscribeStateChanged(
```

```
    stage: Stage,
    participantInfo: ParticipantInfo,
    subscribeState: Stage.SubscribeState
) {
    // Update the subscribe state of this participant
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.subscribeState = subscribeState
    }
}

override fun onStreamsAdded(stage: Stage, participantInfo: ParticipantInfo, streams:
 MutableList<StageStream>) {
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, add these new streams to that participant's streams
 array.
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.streams.addAll(streams)
    }
}

override fun onStreamsRemoved(stage: Stage, participantInfo: ParticipantInfo, streams:
 MutableList<StageStream>) {
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, remove these streams from that participant's streams
 array.
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.streams.removeAll(streams)
    }
}

override fun onStreamsMutedChanged(
    stage: Stage,
    participantInfo: ParticipantInfo,
    streams: MutableList<StageStream>
) {
```

```
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, notify the adapter that the participant has been
 updated. There is no need to modify
    // the `streams` property on the `StageParticipant` because it is the same
 `StageStream` instance. Just
    // query the `isMuted` property again.
    participantAdapter.participantUpdated(participantInfo.participantId) {}
}
```

Implementing a Custom RecyclerView LayoutManager

Laying out different numbers of participants can be complex. You want them to take up the entire parent view's frame but you don't want to handle each participant configuration independently. To make this easy, we'll walk through implementing a `RecyclerView.LayoutManager`.

Create another new class, `StageLayoutManager`, which should extend `GridLayoutManager`. This class is designed to calculate the layout for each participant based on the number of participants in a flow-based row/column layout. Each row is the same height as the others, but columns can be different widths per row. See the code comment above the `layouts` variable for a description of how to customize this behavior.

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.content.Context
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView

class StageLayoutManager(context: Context?) : GridLayoutManager(context, 6) {

    companion object {
        /**
         * This 2D array contains the description of how the grid of participants
 should be rendered
         * The index of the 1st dimension is the number of participants needed to
 active that configuration
         * Meaning if there is 1 participant, index 0 will be used. If there are 5
 participants, index 4 will be used.
         *
```

```
         * The 2nd dimension is a description of the layout. The length of the array is
the number of rows that
         * will exist, and then each number within that array is the number of columns
in each row.
         *
         * See the code comments next to each index for concrete examples.
         *
         * This can be customized to fit any layout configuration needed.
         */
        val layouts: List<List<Int>> = listOf(
            // 1 participant
            listOf(1), // 1 row, full width
            // 2 participants
            listOf(1, 1), // 2 rows, all columns are full width
            // 3 participants
            listOf(1, 2), // 2 rows, first row's column is full width then 2nd row's
columns are 1/2 width
            // 4 participants
            listOf(2, 2), // 2 rows, all columns are 1/2 width
            // 5 participants
            listOf(1, 2, 2), // 3 rows, first row's column is full width, 2nd and 3rd
row's columns are 1/2 width
            // 6 participants
            listOf(2, 2, 2), // 3 rows, all column are 1/2 width
            // 7 participants
            listOf(2, 2, 3), // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd
row's columns are 1/3rd width
            // 8 participants
            listOf(2, 3, 3),
            // 9 participants
            listOf(3, 3, 3),
            // 10 participants
            listOf(2, 3, 2, 3),
            // 11 participants
            listOf(2, 3, 3, 3),
            // 12 participants
            listOf(3, 3, 3, 3),
        )
    }

    init {
        spanSizeLookup = object : SpanSizeLookup() {
            override fun getSpanSize(position: Int): Int {
                if (itemCount <= 0) {
```

```
                    return 1
                }
                // Calculate the row we're in
                val config = layouts[itemCount - 1]
                var row = 0
                var curPosition = position
                while (curPosition - config[row] >= 0) {
                    curPosition -= config[row]
                    row++
                }
                // spanCount == max spans, config[row] = number of columns we want
                // So spanCount / config[row] would be something like 6 / 3 if we want
3 columns.
                // So this will take up 2 spans, with a max of 6 is 1/3rd of the view.
                return spanCount / config[row]
            }
        }
    }

    override fun onLayoutChildren(recycler: RecyclerView.Recycler?, state:
RecyclerView.State?) {
        if (itemCount <= 0 || state?.isPreLayout == true) return

        val parentHeight = height
        val itemHeight = parentHeight / layouts[itemCount - 1].size // height divided
by number of rows.

        // Set the height of each view based on how many rows exist for the current
participant count.
        for (i in 0 until childCount) {
            val child = getChildAt(i) ?: continue
            val layoutParams = child.layoutParams as RecyclerView.LayoutParams
            if (layoutParams.height != itemHeight) {
                layoutParams.height = itemHeight
                child.layoutParams = layoutParams
            }
        }
        // After we set the height for all our views, call super.
        // This works because our RecyclerView can not scroll and all views are always
visible with stable IDs.
        super.onLayoutChildren(recycler, state)
    }

    override fun canScrollVertically(): Boolean = false
```

```
    override fun canScrollHorizontally(): Boolean = false
}
```

Back in `MainActivity.kt` we need to set the adapter and layout manager for our `RecyclerView`:

```
// In onCreate after setting recyclerView.
recyclerView.layoutManager = StageLayoutManager(this)
recyclerView.adapter = viewModel.participantAdapter
```

Hooking Up UI Actions

We are getting close; there are just a few UI actions that we need to hook up.

First we'll have our `MainActivity` observe the `StateFlow` changes from `MainViewModel`:

```
// At the end of your onCreate method
lifecycleScope.launch {
    repeatOnLifecycle(Lifecycle.State.CREATED) {
        viewModel.connectionState.collect { state ->
            buttonJoin.setText(if (state == ConnectionState.DISCONNECTED) R.string.join
 else R.string.leave)
            textViewState.text = getString(R.string.state, state.name)
        }
    }
}
```

Next we add listeners to our Join button and Publish checkbox:

```
buttonJoin.setOnClickListener {
    viewModel.joinStage(editTextToken.text.toString())
}
checkboxPublish.setOnCheckedChangeListener { _, isChecked ->
    viewModel.setPublishEnabled(isChecked)
}
```

Both of the above call functionality in our `MainViewModel`, which we implement now:

```
internal fun joinStage(token: String) {
    if (_connectionState.value != Stage.ConnectionState.DISCONNECTED) {
        // If we're already connected to a stage, leave it.
```

```
        stage?.leave()
    } else {
        if (token.isEmpty()) {
            Toast.makeText(getApplication(), "Empty Token", Toast.LENGTH_SHORT).show()
            return
        }
        try {
            // Destroy the old stage first before creating a new one.
            stage?.release()
            val stage = Stage(getApplication(), token, this)
            stage.addRenderer(this)
            stage.join()
            this.stage = stage
        } catch (e: BroadcastException) {
            Toast.makeText(getApplication(), "Failed to join stage
 ${e.localizedMessage}", Toast.LENGTH_LONG).show()
            e.printStackTrace()
        }
    }
}

internal fun setPublishEnabled(enabled: Boolean) {
    publishEnabled = enabled
}
```

Rendering the Participants

Finally, we need to render the data we receive from the SDK onto the participant item that we created earlier. We already have the `RecyclerView` logic finished, so we just need to implement the `bind` API in `ParticipantItem`.

We'll start by adding the empty function and then walk through it step by step:

```
fun bind(participant: StageParticipant) {

}
```

First we'll handle the easy state, the participant ID, publish state, and subscribe state. For these, we just update our `TextViews` directly:

```
val participantId = if (participant.isLocal) {
    "You (${participant.participantId ?: "Disconnected"})"
```

```
} else {
    participant.participantId
}
textViewParticipantId.text = participantId
textViewPublish.text = participant.publishState.name
textViewSubscribe.text = participant.subscribeState.name
```

Next we'll update the audio and video muted states. To get the muted state, we need to find the `ImageDevice` and `AudioDevice` from the streams array. To optimize performance, we remember the last attached device IDs.

```
// This belongs outside the `bind` API.
private var imageDeviceUrn: String? = null
private var audioDeviceUrn: String? = null

// This belongs inside the `bind` API.
val newImageStream = participant
    .streams
    .firstOrNull { it.device is ImageDevice }
textViewVideoMuted.text = if (newImageStream != null) {
    if (newImageStream.muted) "Video muted" else "Video not muted"
} else {
    "No video stream"
}

val newAudioStream = participant
    .streams
    .firstOrNull { it.device is AudioDevice }
textViewAudioMuted.text = if (newAudioStream != null) {
    if (newAudioStream.muted) "Audio muted" else "Audio not muted"
} else {
    "No audio stream"
}
```

Finally we want to render a preview for the `imageDevice`:

```
if (newImageStream?.device?.descriptor?.urn != imageDeviceUrn) {
    // If the device has changed, remove all subviews from the preview container
    previewContainer.removeAllViews()
    (newImageStream?.device as? ImageDevice)?.let {
        val preview = it.getPreviewView(BroadcastConfiguration.AspectMode.FIT)
        previewContainer.addView(preview)
```

```
        preview.layoutParams = FrameLayout.LayoutParams(
            FrameLayout.LayoutParams.MATCH_PARENT,
            FrameLayout.LayoutParams.MATCH_PARENT
        )
    }
}
imageDeviceUrn = newImageStream?.device?.descriptor?.urn
```

And we display audio stats from the `audioDevice`:

```
if (newAudioStream?.device?.descriptor?.urn != audioDeviceUrn) {
    (newAudioStream?.device as? AudioDevice)?.let {
        it.setStatsCallback { _, rms ->
            textViewAudioLevel.text = "Audio Level: ${rms.roundToInt()} dB"
        }
    }
}
audioDeviceUrn = newAudioStream?.device?.descriptor?.urn
```

Publish & Subscribe with the IVS iOS Broadcast SDK

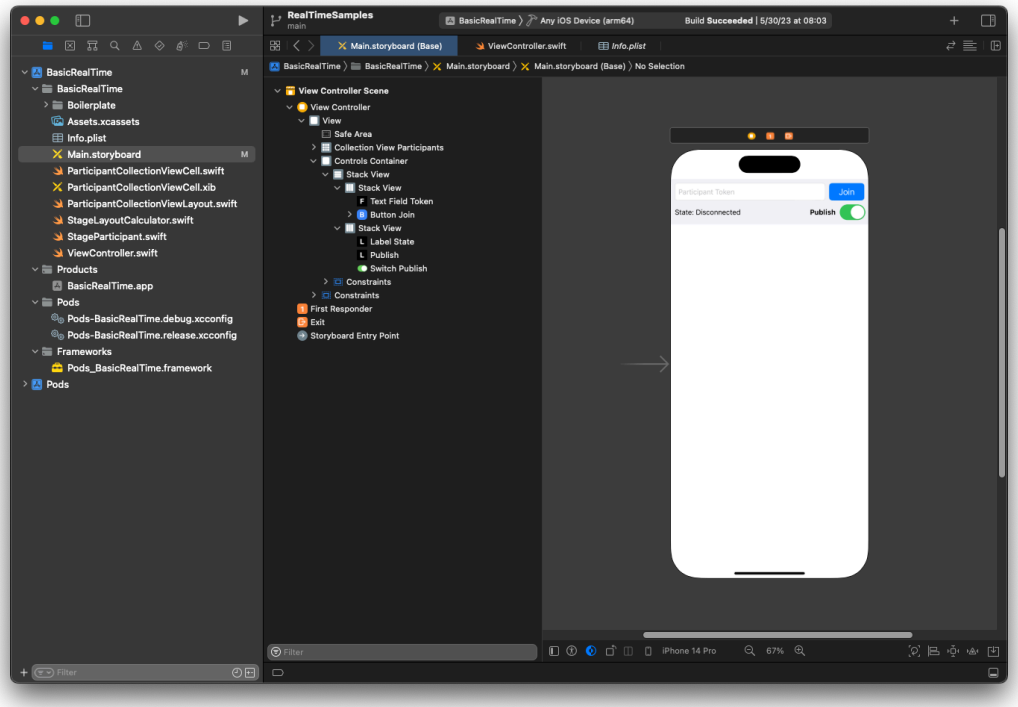
This section takes you through the steps involved in publishing and subscribing to a stage using your iOS app.

Create Views

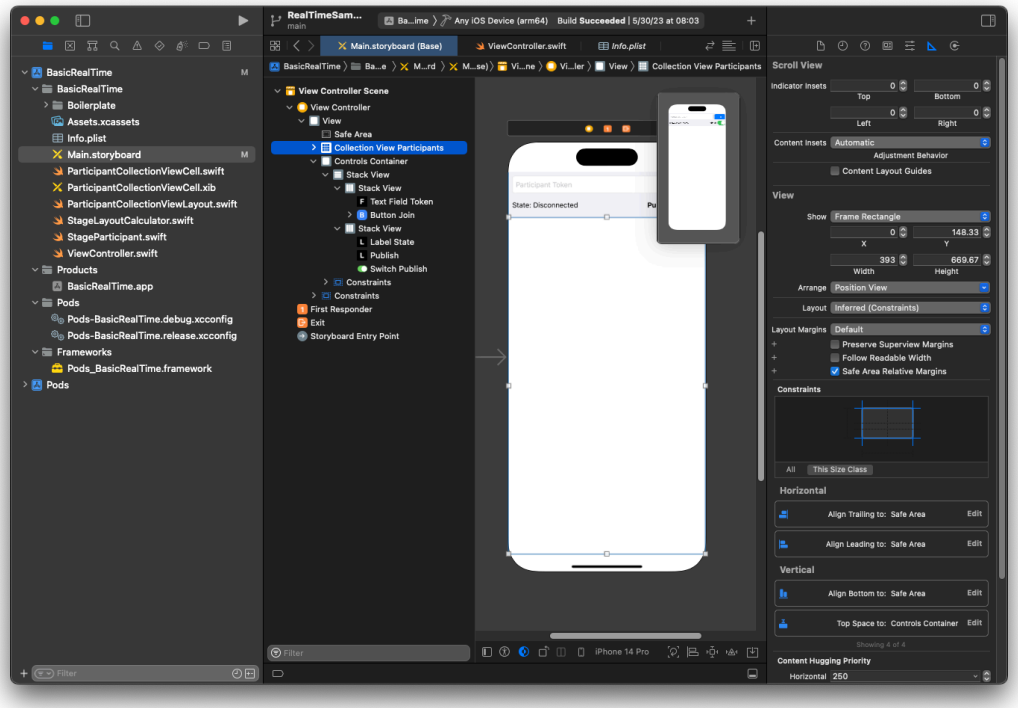
We start by using the auto-created `ViewController.swift` file to import `AmazonIVSBroadcast` and then add some `@IBOutlets` to link:

```
import AmazonIVSBroadcast

class ViewController: UIViewController {

    @IBOutlet private var textFieldToken: UITextField!
    @IBOutlet private var buttonJoin: UIButton!
    @IBOutlet private var labelState: UILabel!
    @IBOutlet private var switchPublish: UISwitch!
    @IBOutlet private var collectionViewParticipants: UICollectionView!
```

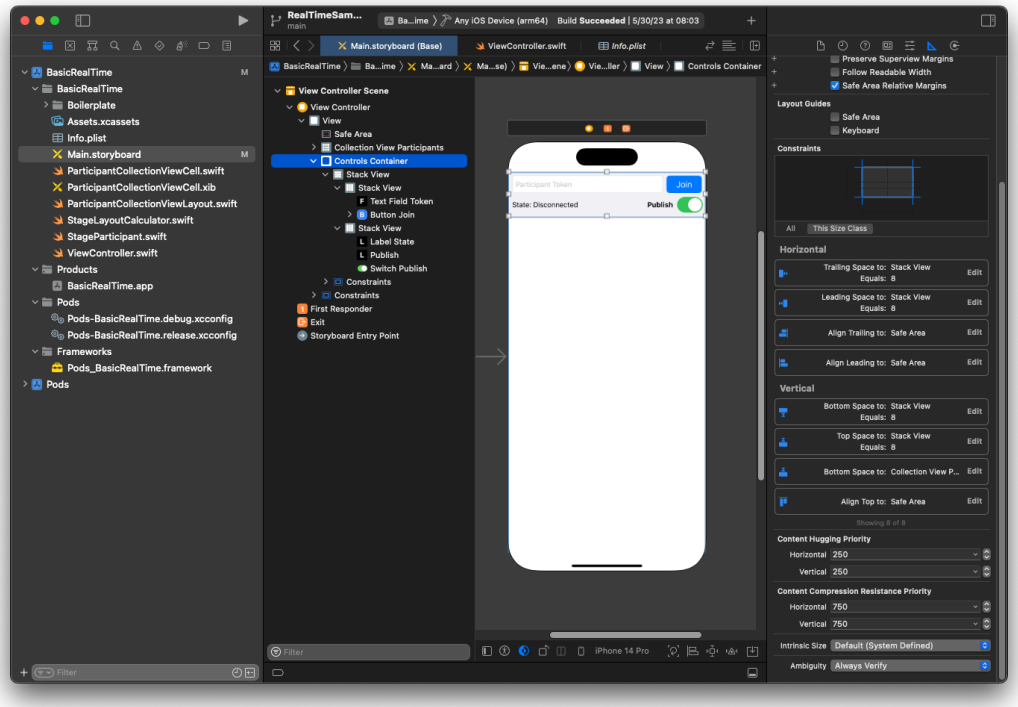
Now we create those views and link them up in `Main.storyboard`. Here is the view structure that we'll use:



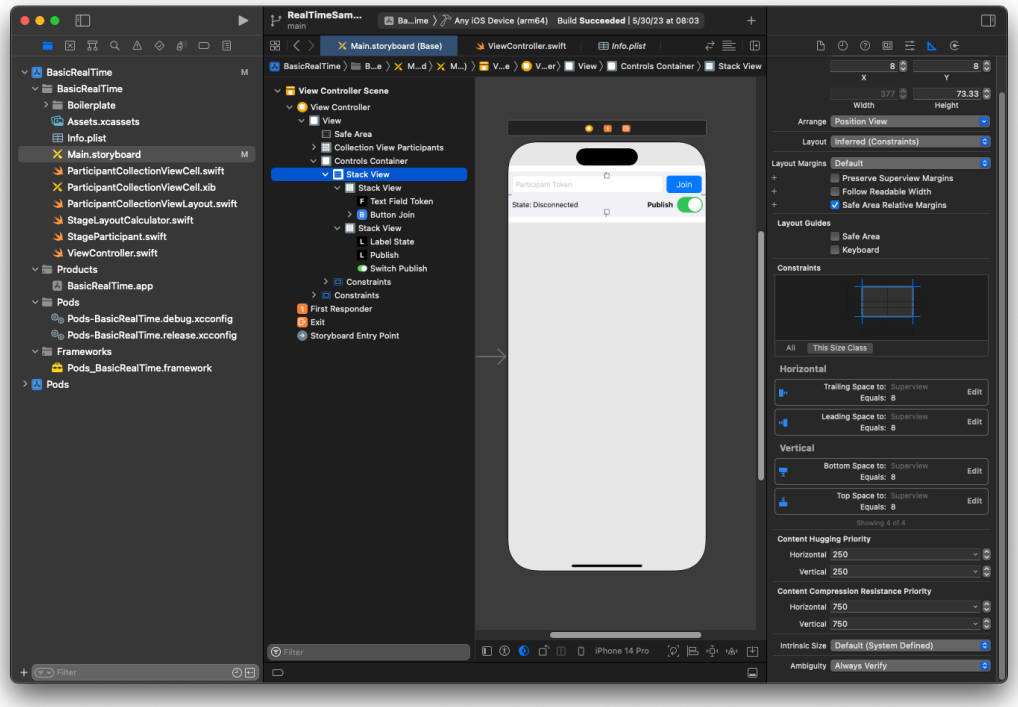
For AutoLayout configuration, we need to customize three views. The first view is **Collection View Participants** (a `UICollectionView`). Bound **Leading**, **Trailing**, and **Bottom** to **Safe Area**. Also bound **Top** to **Controls Container**.



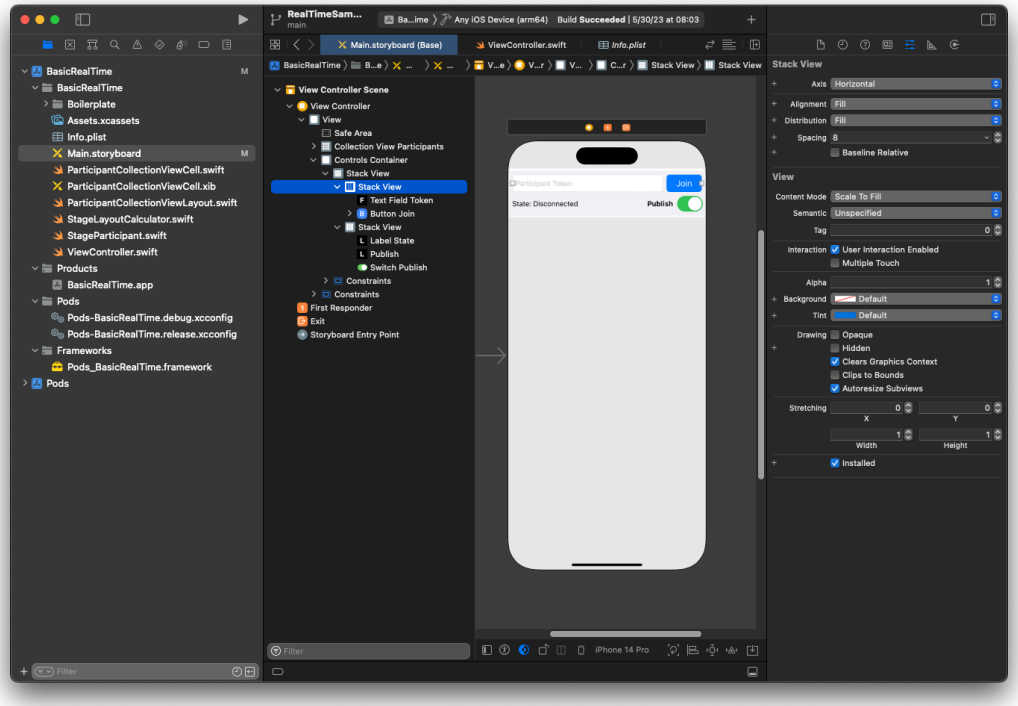
The second view is **Controls Container**. Bound **Leading**, **Trailing**, and **Top** to **Safe Area**:



The third and last view is **Vertical Stack View**. Bound **Top**, **Leading**, **Trailing**, and **Bottom** to **Superview**. For styling, set the spacing to 8 instead of 0.



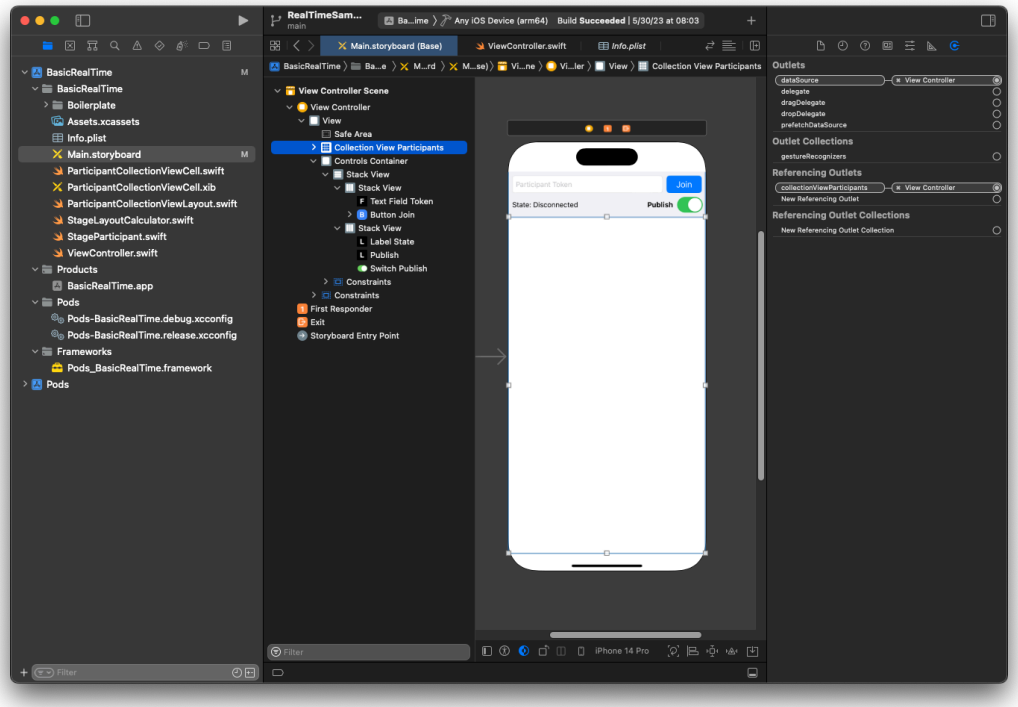
The **UIStackViews** will handle the layout of the remaining views. For all three **UIStackViews**, use **Fill** as the **Alignment** and **Distribution**.



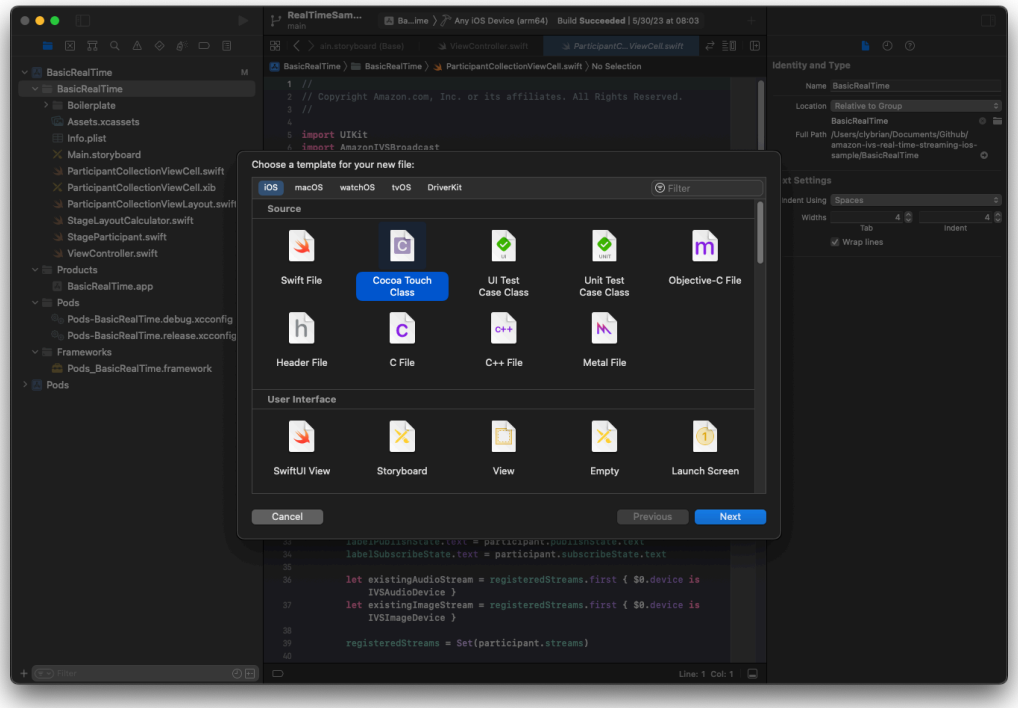
Finally, let's link these views to our `ViewController`. From above, map the following views:

- **Text Field Join** binds to `textFieldToken`.
- **Button Join** binds to `buttonJoin`.
- **Label State** binds to `labelState`.
- **Switch Publish** binds to `switchPublish`.
- **Collection View Participants** binds to `collectionViewParticipants`.

Also use this time to set the `dataSource` of the **Collection View Participants** item to the owning `ViewController`:



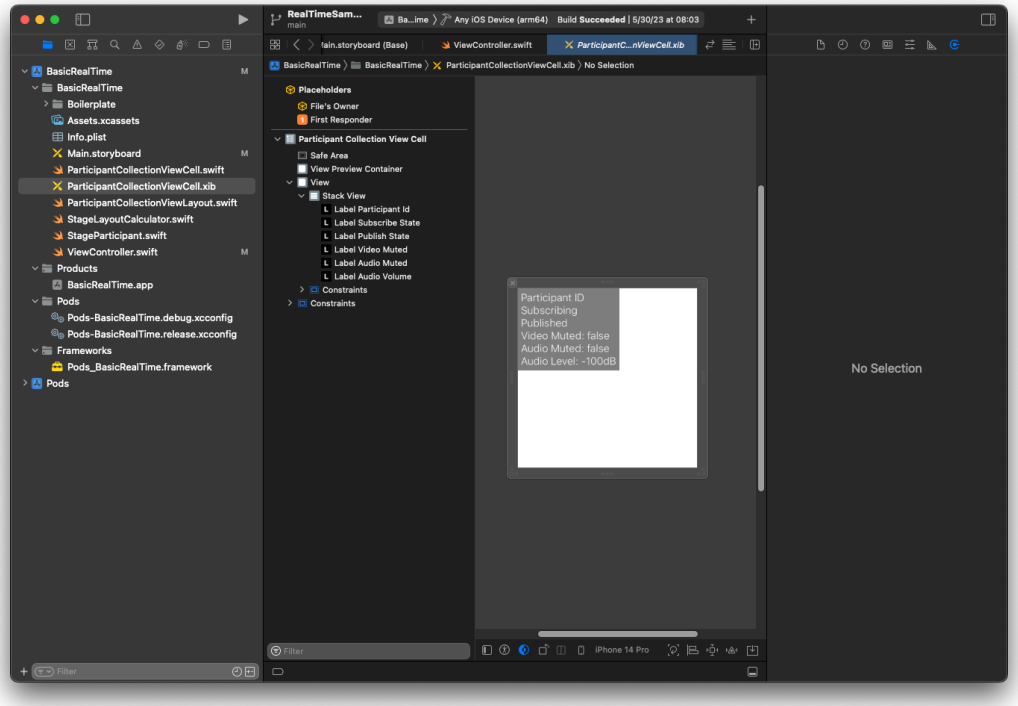
Now we create the `UICollectionViewCell` subclass in which to render the participants. Start by creating a new **Cocoa Touch Class** file:



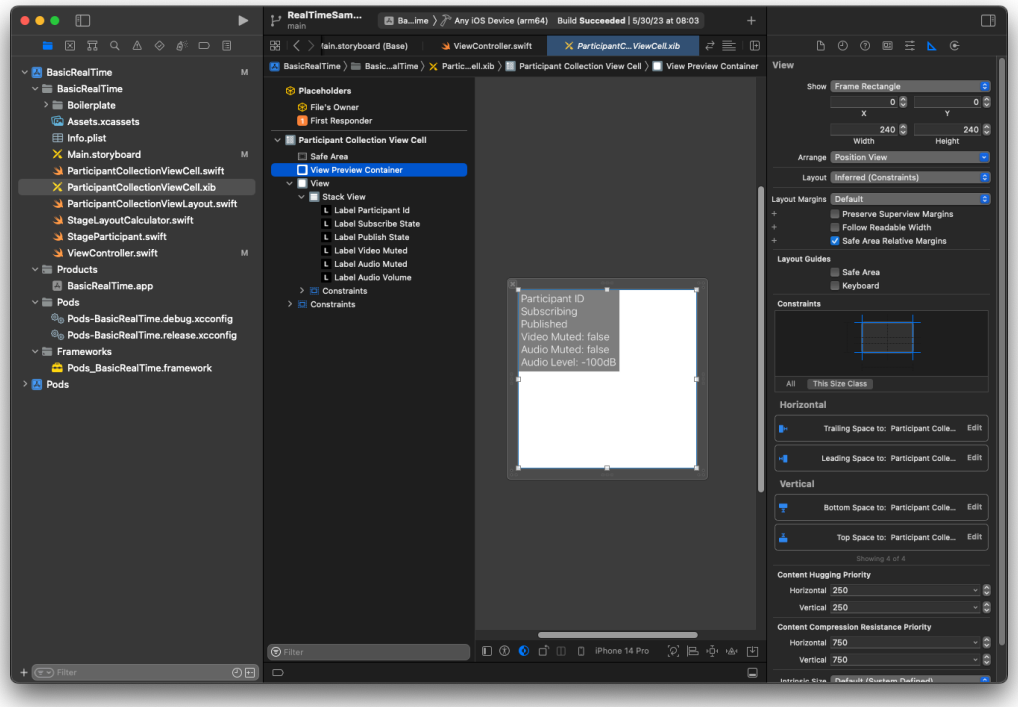
Name it `ParticipantUICollectionViewCell` and make it a subclass of `UICollectionViewCell` in Swift. We start in the Swift file again, creating our `@IBOutlets` to link:

```
import AmazonIVSBroadcast

class ParticipantCollectionViewCell: UICollectionViewCell {

    @IBOutlet private var viewPreviewContainer: UIView!
    @IBOutlet private var labelParticipantId: UILabel!
    @IBOutlet private var labelSubscribeState: UILabel!
    @IBOutlet private var labelPublishState: UILabel!
    @IBOutlet private var labelVideoMuted: UILabel!
    @IBOutlet private var labelAudioMuted: UILabel!
    @IBOutlet private var labelAudioVolume: UILabel!
```

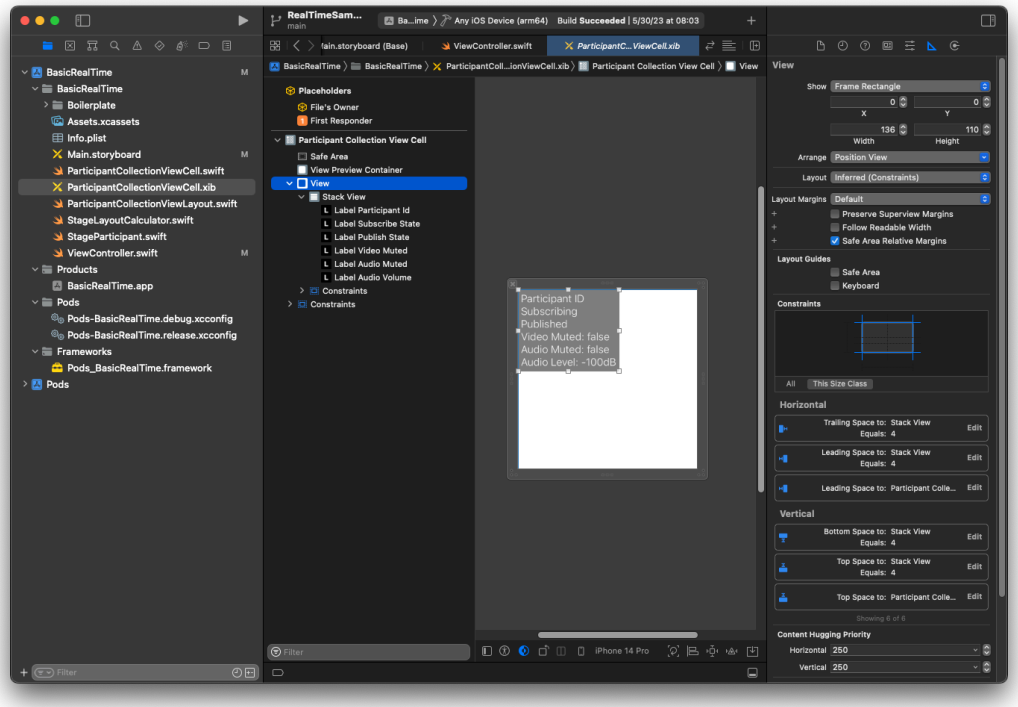
In the associated XIB file, create this view hierarchy:



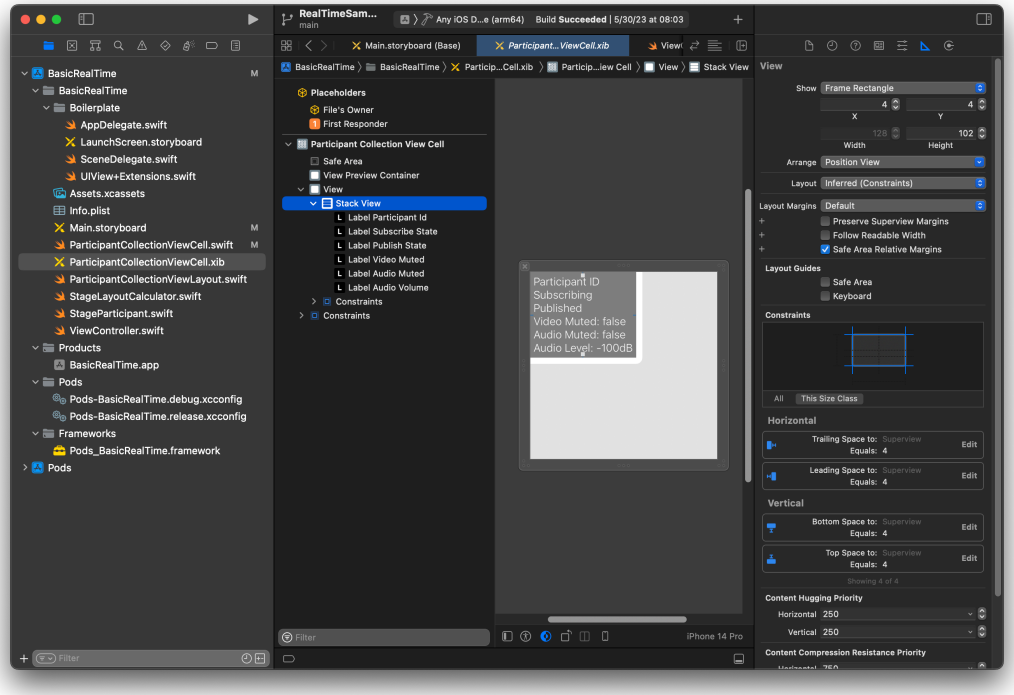
For AutoLayout, we'll modify three views again. The first view is **View Preview Container**. Set **Trailing**, **Leading**, **Top**, and **Bottom** to **Participant Collection View Cell**.



The second view is **View**. Set **Leading** and **Top** to **Participant Collection View Cell** and change the value to 4.



The third view is **Stack View**. Set **Trailing**, **Leading**, **Top**, and **Bottom** to **Superview** and change the value to 4.



Permissions and Idle Timer

Going back to our `ViewController`, we will disable the system idle timer to prevent the device from going to sleep while our application is being used:

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    // Prevent the screen from turning off during a call.
    UIApplication.shared.isIdleTimerDisabled = true
}

override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    UIApplication.shared.isIdleTimerDisabled = false
}
```

Next we request camera and microphone permissions from the system:

```
private func checkPermissions() {
    checkOrGetPermission(for: .video) { [weak self] granted in
```

```
        guard granted else {
            print("Video permission denied")
            return
        }
        self?.checkOrGetPermission(for: .audio) { [weak self] granted in
            guard granted else {
                print("Audio permission denied")
                return
            }
            self?.setupLocalUser() // we will cover this later
        }
    }
}

private func checkOrGetPermission(for mediaType: AVMediaType, _ result: @escaping
 (Bool) -> Void) {
    func mainThreadResult(_ success: Bool) {
        DispatchQueue.main.async {
            result(success)
        }
    }
    switch AVCaptureDevice.authorizationStatus(for: mediaType) {
    case .authorized: mainThreadResult(true)
    case .notDetermined:
        AVCaptureDevice.requestAccess(for: mediaType) { granted in
            mainThreadResult(granted)
        }
    case .denied, .restricted: mainThreadResult(false)
    @unknown default: mainThreadResult(false)
    }
}
```

App State

We need to configure our `collectionViewParticipants` with the layout file that we created earlier:

```
override func viewDidLoad() {
    super.viewDidLoad()
    // We render everything to exactly the frame, so don't allow scrolling.
    collectionViewParticipants.isScrollEnabled = false
    collectionViewParticipants.register(UINib(nibName: "ParticipantCollectionViewCell",
 bundle: .main), forCellWithReuseIdentifier: "ParticipantCollectionViewCell")
```

```
}
```

To represent each participant, we create a simple struct called `StageParticipant`. This can be included in the `ViewController.swift` file, or a new file can be created.

```
import Foundation
import AmazonIVSBroadcast

struct StageParticipant {
    let isLocal: Bool
    var participantId: String?
    var publishState: IVSParticipantPublishState = .notPublished
    var subscribeState: IVSParticipantSubscribeState = .notSubscribed
    var streams: [IVSStageStream] = []

    init(isLocal: Bool, participantId: String?) {
        self.isLocal = isLocal
        self.participantId = participantId
    }
}
```

To track those participants, we keep an array of them as a private property in our `ViewController`:

```
private var participants = [StageParticipant]()
```

This property will be used to power our `UICollectionViewDataSource` that was linked from the storyboard earlier:

```
extension ViewController: UICollectionViewDataSource {

    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
 section: Int) -> Int {
        return participants.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
 IndexPath) -> UICollectionViewCell {
        if let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
 "ParticipantCollectionViewCell", for: indexPath) as? ParticipantCollectionViewCell {
            cell.set(participant: participants[indexPath.row])
            return cell
```

```
        } else {
            fatalError("Couldn't load custom cell type
 'ParticipantCollectionViewCell'")
        }
    }

}
```

To see your own preview before joining a stage, we create a local participant immediately:

```
override func viewDidLoad() {
    /* existing UICollectionView code */
    participants.append(StageParticipant(isLocal: true, participantId: nil))
}
```

This results in a participant cell being rendered immediately once the app is running, representing the local participant.

Users want to be able to see themselves before joining a stage, so next we implement the `setupLocalUser()` method that gets called from the permissions-handling code earlier. We store the camera and microphone reference as `IVSLocalStageStream` objects.

```
private var streams = [IVSLocalStageStream]()
private let deviceDiscovery = IVSDeviceDiscovery()

private func setupLocalUser() {
    // Gather our camera and microphone once permissions have been granted
    let devices = deviceDiscovery.listLocalDevices()
    streams.removeAll()
    if let camera = devices.compactMap({ $0 as? IVSCamera }).first {
        streams.append(IVSLocalStageStream(device: camera))
        // Use a front camera if available.
        if let frontSource = camera.listAvailableInputSources().first(where:
 { $0.position == .front }) {
            camera.setPreferredInputSource(frontSource)
        }
    }
    if let mic = devices.compactMap({ $0 as? IVSMicrophone }).first {
        streams.append(IVSLocalStageStream(device: mic))
    }
    participants[0].streams = streams
    participantsChanged(index: 0, changeType: .updated)
```

```
}
```

Here we've found the device's camera and microphone through the SDK and stored them in our local `streams` object, then assigned the `streams` array of the first participant (the local participant that we created earlier) to our `streams`. Finally we call `participantsChanged` with an `index` of 0 and `changeType` of `updated`. That function is a helper function for updating our `UICollectionView` with nice animations. Here's what it looks like:

```
private func participantsChanged(index: Int, changeType: ChangeType) {
    switch changeType {
    case .joined:
        collectionViewParticipants?.insertItems(at: [IndexPath(item: index, section:
 0)])
    case .updated:
        // Instead of doing reloadItems, just grab the cell and update it ourselves. It
 saves a create/destroy of a cell
        // and more importantly fixes some UI flicker. We disable scrolling so the
 index path per cell
        // never changes.
        if let cell = collectionViewParticipants?.cellForItem(at: IndexPath(item:
 index, section: 0)) as? ParticipantCollectionViewCell {
            cell.set(participant: participants[index])
        }
    case .left:
        collectionViewParticipants?.deleteItems(at: [IndexPath(item: index, section:
 0)])
    }
}
```

Don't worry about `cell.set` yet; we'll get to that later, but that's where we will render the cell's contents based on the participant.

The `ChangeType` is a simple enum:

```
enum ChangeType {
    case joined, updated, left
}
```

Finally, we want to keep track of whether the stage is connected. We use a simple `bool` to track that, which will automatically update our UI when it is updated itself.

```
private var connectingOrConnected = false {
    didSet {
        buttonJoin.setTitle(connectingOrConnected ? "Leave" : "Join", for: .normal)
        buttonJoin.tintColor = connectingOrConnected ? .systemRed : .systemBlue
    }
}
```

Implement the Stage SDK

Three core concepts underlie real-time functionality: stage, strategy, and renderer. The design goal is minimizing the amount of client-side logic necessary to build a working product.

IVSStageStrategy

Our `IVSStageStrategy` implementation is simple:

```
extension ViewController: IVSStageStrategy {
    func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream] {
        // Return the camera and microphone to be published.
        // This is only called if `shouldPublishParticipant` returns true.
        return streams
    }

    func stage(_ stage: IVSStage, shouldPublishParticipant participant:
 IVSParticipantInfo) -> Bool {
        // Our publish status is based directly on the UISwitch view
        return switchPublish.isOn
    }

    func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
        // Subscribe to both audio and video for all publishing participants.
        return .audioVideo
    }
}
```

To summarize, we only publish if the publish switch is in the “on” position, and if we publish we will publish the streams that we collected earlier. Finally, for this sample, we always subscribe to other participants, receiving both their audio and video.

IVSStageRenderer

The `IVSStageRenderer` implementation also is fairly simple, though given the number of functions it contains quite a bit more code. The general approach in this renderer is to update our `participants` array when the SDK notifies us of a change to a participant. There are certain scenarios where we handle local participants differently, because we have decided to manage them ourselves so they can see their camera preview before joining.

```
extension ViewController: IVSStageRenderer {

   func stage(_ stage: IVSStage, didChange connectionState: IVSStageConnectionState,
 withError error: Error?) {
       labelState.text = connectionState.text
       connectingOrConnected = connectionState != .disconnected
   }

   func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo) {
       if participant.isLocal {
           // If this is the local participant joining the Stage, update the first
 participant in our array because we
           // manually added that participant when setting up our preview
           participants[0].participantId = participant.participantId
           participantsChanged(index: 0, changeType: .updated)
       } else {
           // If they are not local, add them to the array as a newly joined
 participant.
           participants.append(StageParticipant(isLocal: false, participantId:
 participant.participantId))
           participantsChanged(index: (participants.count - 1), changeType: .joined)
       }
   }

   func stage(_ stage: IVSStage, participantDidLeave participant: IVSParticipantInfo)
 {
       if participant.isLocal {
           // If this is the local participant leaving the Stage, update the first
 participant in our array because
           // we want to keep the camera preview active
           participants[0].participantId = nil
           participantsChanged(index: 0, changeType: .updated)
       } else {
           // If they are not local, find their index and remove them from the array.
```

```
            if let index = participants.firstIndex(where: { $0.participantId ==
participant.participantId }) {
                participants.remove(at: index)
                participantsChanged(index: index, changeType: .left)
            }
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
publishState: IVSParticipantPublishState) {
        // Update the publishing state of this participant
        mutatingParticipant(participant.participantId) { data in
            data.publishState = publishState
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
subscribeState: IVSParticipantSubscribeState) {
        // Update the subscribe state of this participant
        mutatingParticipant(participant.participantId) { data in
            data.subscribeState = subscribeState
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo,
didChangeMutedStreams streams: [IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
those streams locally
        if participant.isLocal { return }
        // For remote participants, notify the UICollectionView that they have updated.
There is no need to modify
        // the `streams` property on the `StageParticipant` because it is the same
`IVSStageStream` instance. Just
        // query the `isMuted` property again.
        if let index = participants.firstIndex(where: { $0.participantId ==
participant.participantId }) {
            participantsChanged(index: index, changeType: .updated)
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams:
[IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
those streams locally
```

```
        if participant.isLocal { return }
        // For remote participants, add these new streams to that participant's streams
 array.
        mutatingParticipant(participant.participantId) { data in
            data.streams.append(contentsOf: streams)
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didRemove streams:
 [IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
 those streams locally
        if participant.isLocal { return }
        // For remote participants, remove these streams from that participant's
 streams array.
        mutatingParticipant(participant.participantId) { data in
            let oldUrns = streams.map { $0.device.descriptor().urn }
            data.streams.removeAll(where: { stream in
                return oldUrns.contains(stream.device.descriptor().urn)
            })
        }
    }

    // A helper function to find a participant by its ID, mutate that participant, and
 then update the UICollectionView accordingly.
    private func mutatingParticipant(_ participantId: String?, modifier: (inout
 StageParticipant) -> Void) {
        guard let index = participants.firstIndex(where: { $0.participantId ==
 participantId }) else {
            fatalError("Something is out of sync, investigate if this was a sample app
 or SDK issue.")
        }

        var participant = participants[index]
        modifier(&participant)
        participants[index] = participant
        participantsChanged(index: index, changeType: .updated)
    }
}
```

This code uses an extension to convert the connection state into human-friendly text:

```
extension IVSStageConnectionState {
```

```
    var text: String {
        switch self {
        case .disconnected: return "Disconnected"
        case .connecting: return "Connecting"
        case .connected: return "Connected"
        @unknown default: fatalError()
        }
    }
}
```

Implementing a Custom UICollectionViewLayout

Laying out different numbers of participants can be complex. You want them to take up the entire parent view's frame but you don't want to handle each participant configuration independently. To make this easy, we'll walk through implementing a `UICollectionViewLayout`.

Create another new file, `ParticipantCollectionViewLayout.swift`, which should extend `UICollectionViewLayout`. This class will use another class called `StageLayoutCalculator`, which we'll cover soon. The class receives calculated frame values for each participant and then generates the necessary `UICollectionViewLayoutAttributes` objects.

```
import Foundation
import UIKit

/**
 Code modified from https://developer.apple.com/documentation/uikit/views_and_controls/
collection_views/layouts/customizing_collection_view_layouts?language=objc
 */
class ParticipantCollectionViewLayout: UICollectionViewLayout {

    private let layoutCalculator = StageLayoutCalculator()

    private var contentBounds = CGRect.zero
    private var cachedAttributes = [UICollectionViewLayoutAttributes]()

    override func prepare() {
        super.prepare()

        guard let collectionView = collectionView else { return }

        cachedAttributes.removeAll()
        contentBounds = CGRect(origin: .zero, size: collectionView.bounds.size)
```

```
        layoutCalculator.calculateFrames(participantCount:
collectionView.numberOfItems(inSection: 0),
                                         width: collectionView.bounds.size.width,
                                         height: collectionView.bounds.size.height,
                                         padding: 4)
        .enumerated()
        .forEach { (index, frame) in
            let attributes = UICollectionViewLayoutAttributes(forCellWith:
IndexPath(item: index, section: 0))
            attributes.frame = frame
            cachedAttributes.append(attributes)
            contentBounds = contentBounds.union(frame)
        }
    }

    override var collectionViewContentSize: CGSize {
        return contentBounds.size
    }

    override func shouldInvalidateLayout(forBoundsChange newBounds: CGRect) -> Bool {
        guard let collectionView = collectionView else { return false }
        return !newBounds.size.equalTo(collectionView.bounds.size)
    }

    override func layoutAttributesForItem(at indexPath: IndexPath) ->
UICollectionViewLayoutAttributes? {
        return cachedAttributes[indexPath.item]
    }

    override func layoutAttributesForElements(in rect: CGRect) ->
[UICollectionViewLayoutAttributes]? {
        var attributesArray = [UICollectionViewLayoutAttributes]()

        // Find any cell that sits within the query rect.
        guard let lastIndex = cachedAttributes.indices.last, let firstMatchIndex =
binSearch(rect, start: 0, end: lastIndex) else {
            return attributesArray
        }

        // Starting from the match, loop up and down through the array until all the
attributes
        // have been added within the query rect.
        for attributes in cachedAttributes[..<firstMatchIndex].reversed() {
```

```
            guard attributes.frame.maxY >= rect.minY else { break }
            attributesArray.append(attributes)
        }

        for attributes in cachedAttributes[firstMatchIndex...] {
            guard attributes.frame.minY <= rect.maxY else { break }
            attributesArray.append(attributes)
        }

        return attributesArray
    }

    // Perform a binary search on the cached attributes array.
    func binSearch(_ rect: CGRect, start: Int, end: Int) -> Int? {
        if end < start { return nil }

        let mid = (start + end) / 2
        let attr = cachedAttributes[mid]

        if attr.frame.intersects(rect) {
            return mid
        } else {
            if attr.frame.maxY < rect.minY {
                return binSearch(rect, start: (mid + 1), end: end)
            } else {
                return binSearch(rect, start: start, end: (mid - 1))
            }
        }
    }
}
```

More important is the `StageLayoutCalculator.swift` class. It is designed to calculate the frames for each participant based on the number of participants in a flow-based row/column layout. Each row is the same height as the others, but the columns can be different widths per row. See the code comment above the `layouts` variable for a description of how to customize this behavior.

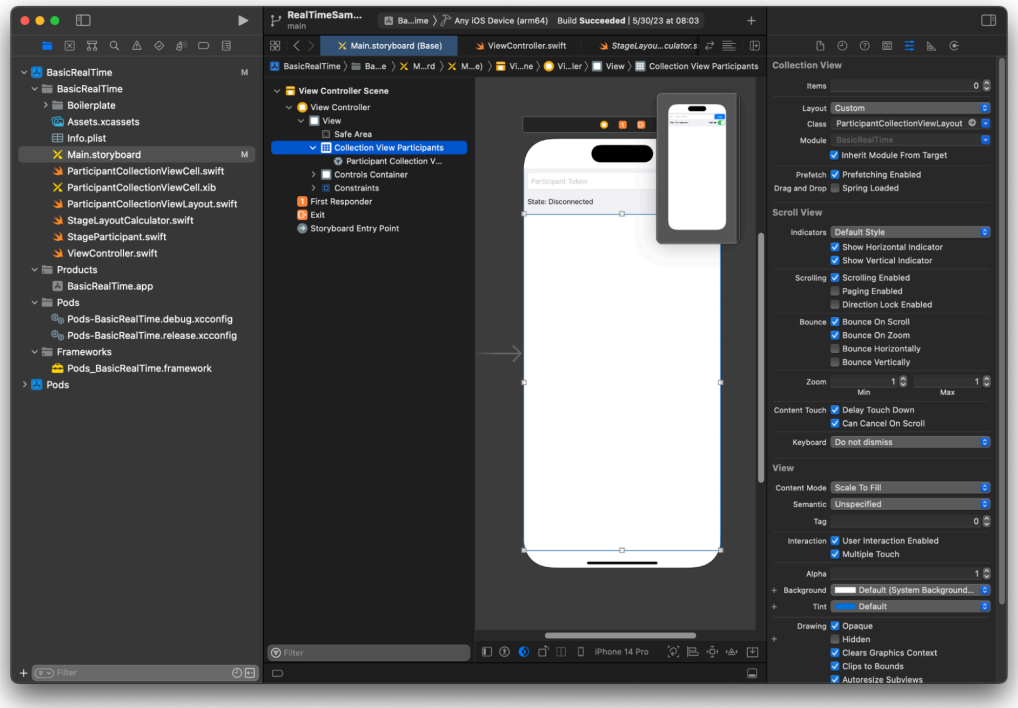
```
import Foundation
import UIKit

class StageLayoutCalculator {
```

```
    /// This 2D array contains the description of how the grid of participants should
be rendered
    /// The index of the 1st dimension is the number of participants needed to active
that configuration
    /// Meaning if there is 1 participant, index 0 will be used. If there are 5
participants, index 4 will be used.
    ///
    /// The 2nd dimension is a description of the layout. The length of the array is
the number of rows that
    /// will exist, and then each number within that array is the number of columns in
each row.
    ///
    /// See the code comments next to each index for concrete examples.
    ///
    /// This can be customized to fit any layout configuration needed.
    private let layouts: [[Int]] = [
        // 1 participant
        [ 1 ], // 1 row, full width
        // 2 participants
        [ 1, 1 ], // 2 rows, all columns are full width
        // 3 participants
        [ 1, 2 ], // 2 rows, first row's column is full width then 2nd row's columns
are 1/2 width
        // 4 participants
        [ 2, 2 ], // 2 rows, all columns are 1/2 width
        // 5 participants
        [ 1, 2, 2 ], // 3 rows, first row's column is full width, 2nd and 3rd row's
columns are 1/2 width
        // 6 participants
        [ 2, 2, 2 ], // 3 rows, all column are 1/2 width
        // 7 participants
        [ 2, 2, 3 ], // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd row's
columns are 1/3rd width
        // 8 participants
        [ 2, 3, 3 ],
        // 9 participants
        [ 3, 3, 3 ],
        // 10 participants
        [ 2, 3, 2, 3 ],
        // 11 participants
        [ 2, 3, 3, 3 ],
        // 12 participants
        [ 3, 3, 3, 3 ],
    ]
```

```
   // Given a frame (this could be for a UICollectionView, or a Broadcast Mixer's
canvas), calculate the frames for each
   // participant, with optional padding.
   func calculateFrames(participantCount: Int, width: CGFloat, height: CGFloat,
padding: CGFloat) -> [CGRect] {
       if participantCount > layouts.count {
           fatalError("Only \(layouts.count) participants are supported at this time")
       }
       if participantCount == 0 {
           return []
       }
       var currentIndex = 0
       var lastFrame: CGRect = .zero

       // If the height is less than the width, the rows and columns will be flipped.
       // Meaning for 6 participants, there will be 2 rows of 3 columns each.
       let isVertical = height > width

       let halfPadding = padding / 2.0

       let layout = layouts[participantCount - 1] // 1 participant is in index 0, so
`-1`.
       let rowHeight = (isVertical ? height : width) / CGFloat(layout.count)

       var frames = [CGRect]()
       for row in 0 ..< layout.count {
           // layout[row] is the number of columns in a layout
           let itemWidth = (isVertical ? width : height) / CGFloat(layout[row])
           let segmentFrame = CGRect(x: (isVertical ? 0 : lastFrame.maxX) +
halfPadding,
                                     y: (isVertical ? lastFrame.maxY : 0) +
halfPadding,
                                     width: (isVertical ? itemWidth : rowHeight) -
padding,
                                     height: (isVertical ? rowHeight : itemWidth) -
padding)

           for column in 0 ..< layout[row] {
               var frame = segmentFrame
               if isVertical {
                   frame.origin.x = (itemWidth * CGFloat(column)) + halfPadding
               } else {
                   frame.origin.y = (itemWidth * CGFloat(column)) + halfPadding
```

```
                }
                frames.append(frame)
                currentIndex += 1
            }

            lastFrame = segmentFrame
            lastFrame.origin.x += halfPadding
            lastFrame.origin.y += halfPadding
        }
        return frames
    }

}
```

Back in `Main.storyboard`, be sure to set the layout class for the `UICollectionView` to the class we just created:



Hooking Up UI Actions

We are getting close, there are a few `IBActions` that we need to create.

First we'll handle the join button. It responds differently based on the value of `connectingOrConnected`. When it is already connected, it just leaves the stage. If it is

disconnected, it reads the text from the token `UITextField` and creates a new `IVSStage` with that text. Then we add our `ViewController` as the `strategy`, `errorDelegate`, and renderer for the `IVSStage`, and finally we join the stage asynchronously.

```
@IBAction private func joinTapped(_ sender: UIButton) {
    if connectingOrConnected {
        // If we're already connected to a Stage, leave it.
        stage?.leave()
    } else {
        guard let token = textFieldToken.text else {
            print("No token")
            return
        }
        // Hide the keyboard after tapping Join
        textFieldToken.resignFirstResponder()
        do {
            // Destroy the old Stage first before creating a new one.
            self.stage = nil
            let stage = try IVSStage(token: token, strategy: self)
            stage.errorDelegate = self
            stage.addRenderer(self)
            try stage.join()
            self.stage = stage
        } catch {
            print("Failed to join stage - \(error)")
        }
    }
}
```

The other UI action we need to hook up is the publish switch:

```
@IBAction private func publishToggled(_ sender: UISwitch) {
    // Because the strategy returns the value of `switchPublish.isOn`, just call
 `refreshStrategy`.
    stage?.refreshStrategy()
}
```

Rendering the Participants

Finally, we need to render the data we receive from the SDK onto the participant cell that we created earlier. We already have the `UICollectionView` logic finished, so we just need to implement the `set` API in `ParticipantCollectionViewCell.swift`.

We'll start by adding the `empty` function and then walk through it step by step:

```
func set(participant: StageParticipant) {

}
```

First we handle the easy state, the participant ID, publish state, and subscribe state. For these, we just update our `UILabels` directly:

```
labelParticipantId.text = participant.isLocal ? "You (\(participant.participantId ??
 "Disconnected"))" : participant.participantId
labelPublishState.text = participant.publishState.text
labelSubscribeState.text = participant.subscribeState.text
```

The text properties of the publish and subscribe enums come from local extensions:

```
extension IVSParticipantPublishState {
    var text: String {
        switch self {
        case .notPublished: return "Not Published"
        case .attemptingPublish: return "Attempting to Publish"
        case .published: return "Published"
        @unknown default: fatalError()
        }
    }
}

extension IVSParticipantSubscribeState {
    var text: String {
        switch self {
        case .notSubscribed: return "Not Subscribed"
        case .attemptingSubscribe: return "Attempting to Subscribe"
        case .subscribed: return "Subscribed"
        @unknown default: fatalError()
        }
    }
}
```

Next we update the audio and video muted states. To get the muted states we need to find the `IVSImageDevice` and `IVSAudioDevice` from the `streams` array. To optimize performance, we will remember the last devices attached.

```
// This belongs outside `set(participant:)`
private var registeredStreams: Set<IVSStageStream> = []
private var imageDevice: IVSImageDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSImageDevice }.first
}
private var audioDevice: IVSAudioDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSAudioDevice }.first
}

// This belongs inside `set(participant:)`
let existingAudioStream = registeredStreams.first { $0.device is IVSAudioDevice }
let existingImageStream = registeredStreams.first { $0.device is IVSImageDevice }

registeredStreams = Set(participant.streams)

let newAudioStream = participant.streams.first { $0.device is IVSAudioDevice }
let newImageStream = participant.streams.first { $0.device is IVSImageDevice }

// `isMuted != false` covers the stream not existing, as well as being muted.
labelVideoMuted.text = "Video Muted: \(newImageStream?.isMuted != false)"
labelAudioMuted.text = "Audio Muted: \(newAudioStream?.isMuted != false)"
```

Finally we want to render a preview for the `imageDevice` and display audio stats from the `audioDevice`:

```
if existingImageStream !== newImageStream {
    // The image stream has changed
    updatePreview() // We'll cover this next
}

if existingAudioStream !== newAudioStream {
    (existingAudioStream?.device as? IVSAudioDevice)?.setStatsCallback(nil)
    audioDevice?.setStatsCallback( { [weak self] stats in
        self?.labelAudioVolume.text = String(format: "Audio Level: %.0f dB", stats.rms)
    })
    // When the audio stream changes, it will take some time to receive new stats.
 Reset the value temporarily.
    self.labelAudioVolume.text = "Audio Level: -100 dB"
}
```

The last function we need to create is `updatePreview()`, which adds a preview of the participant to our view:

```
private func updatePreview() {
    // Remove any old previews from the preview container
    viewPreviewContainer.subviews.forEach { $0.removeFromSuperview() }
    if let imageDevice = self.imageDevice {
        if let preview = try? imageDevice.previewView(with: .fit) {
            viewPreviewContainer.addSubviewMatchFrame(preview)
        }
    }
}
```

The above uses a helper function on `UIView` to make embedding subviews easier:

```
extension UIView {
    func addSubviewMatchFrame(_ view: UIView) {
        view.translatesAutoresizingMaskIntoConstraints = false
        self.addSubview(view)
        NSLayoutConstraint.activate([
            view.topAnchor.constraint(equalTo: self.topAnchor, constant: 0),
            view.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: 0),
            view.leadingAnchor.constraint(equalTo: self.leadingAnchor, constant: 0),
            view.trailingAnchor.constraint(equalTo: self.trailingAnchor, constant: 0),
        ])
    }
}
```

Monitoring Amazon IVS Real-Time Streaming

This document provides details about options available for monitoring your IVS real-time streaming application.

What is a Stage Session?

A stage *session* begins when the first participant joins a stage and ends a few minutes after the last participant stops publishing to the stage. Stage sessions help with debugging long-lived stages by separating out events and participants into short-lived sessions.

View Stage Sessions and Participants

Console Instructions

1. Open the Amazon IVS console.

   (You also can access the Amazon IVS console through the AWS Management Console.)
2. On the navigation pane, choose **Stages**. (If the nav pane is collapsed, first open it by choosing the hamburger icon.)
3. Choose the stage to go to its details page.
4. Scroll down the page until you see the **Stage sessions** section, then select a stage session to view its details page.
5. To view participants in the session, scroll down until you see the **Participants** section, then select a participant to view its details page, including charts for Amazon CloudWatch metrics.

View Events for a Participant

Events are sent when a participant's status in a stage changes, such as joining a stage or encountering an error trying to publish to a stage. Not all errors cause events; e.g., client-side network errors and token-signature errors are not sent as events. To handle these errors in your client application, use the IVS broadcast SDKs.

Console Instructions

1. Navigate to the participant details page as instructed above.

2. Scroll down until you see the **Events** section. This displays an ordered list of participant events. See Using Amazon EventBridge with Amazon IVS for details on events that are emitted for participants.

CLI Instructions

Accessing stage-session events with the AWS CLI is an advanced option and requires that you first download and configure the CLI on your machine. For details, see the AWS Command Line Interface User Guide.

1. List stage sessions to find a stage session:

```
aws ivs-realtime list-stage-sessions --stage-arn <arn>
```

2. List participants for a stage session to find a participant:

```
aws ivs-realtime list-participants --stage-arn <arn> -session-id <sessionId>
```

3. List events for a stage session and participant:

```
aws ivs-realtime list-participant-events --stage-arn <arn> --session-id <sessionId>
 --participant-id <participantId>
```

Here is a sample response to the `list-participant-events` call:

```
{
    "events": [
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "JOINED",
            "participantId": "AdRezBl021t0"
        },
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "SUBSCRIBE_STARTED",
            "participantId": "AdRezBl021t0",
            "remoteParticipantId": "Ou5b5n5XLMdC"
        },
        {
            "eventTime": "2023-04-04T22:49:45+00:00",
```

```
            "name": "SUBSCRIBE_STOPPED",
            "participantId": "AdRezBl021t0",
            "remoteParticipantId": "Ou5b5n5XLMdC"
        },
        {
            "eventTime": "2023-04-04T22:49:45+00:00",
            "name": "LEFT",
            "participantId": "AdRezBl021t0"
        }
    ]
}
```

Access CloudWatch Metrics

For CloudWatch metrics to be available, the following IVS Broadcast SDK versions are required: Web 1.5.0 or later, Android 1.12.0 or later, or iOS 1.12.0 or later.

CloudWatch Console Instructions

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the side navigation, expand the **Metrics** dropdown, then select **All metrics**.
3. On the **Browse** tab, using the unlabeled dropdown at the left, select your "home" region, where your channel(s) was(were) created. For more on regions, see Global Solution, Regional Control. For a list of supported regions, see the Amazon IVS page in the *AWS General Reference*.
4. At the bottom of the **Browse** tab, select the **IVSRealTime** namespace.
5. Do one of the following:
   a. In the search bar, enter your resource ID (part of the ARN, `arn:::ivs:stage/<resource id>`).

      Then select **IVSRealTime > Stage Metrics**.
   b. If **IVSRealTime** appears as a selectable service under **AWS Namespaces**, select it. It will be listed if you use Amazon IVS Real-Time Streaming and it is sending metrics to Amazon CloudWatch. (If **IVSRealTime** is not listed, you do not have any Amazon IVS metrics.)

      Then choose a *dimension* grouping as desired; available dimensions are listed in CloudWatch Metrics below.
6. Choose metrics to add to the graph. Available metrics are listed in CloudWatch Metrics below.

You also can access your stream session's CloudWatch chart from the stream session's details page, by selecting the **View in CloudWatch** button.

CLI Instructions

You also can access the metrics using the AWS CLI. This requires that you first download and configure the CLI on your machine. For details, see the AWS Command Line Interface User Guide.

Then, to access Amazon IVS real-time streaming metrics using the AWS CLI:

- At a command prompt, run:

```
aws cloudwatch list-metrics --namespace AWS/IVSRealTime
```

For more information, see Using Amazon CloudWatch Metrics in the *Amazon CloudWatch User Guide*.

CloudWatch Metrics: IVS Real-Time Streaming

Amazon IVS provides the following metrics in the **AWS/IVSRealTime** namespace.

For CloudWatch metrics to be available, Web Broadcast SDK 1.5.2 or later must be used.

The dimension can have the following valid values:

- The `Stage` dimension is a resource ID (part of the ARN, `arn:::stage/<resource id>`).
- The `Participant` dimension is a `participantID`.
- The `SimulcastLayer` is "hi", "mid", "low", or "none" for a `MediaType` of "video", or "none" for a `MediaType` of "audio." This value also can be empty.
- The `MediaType` dimension is "video" or "audio" (string).

In the case of participant replication, for the destination stage, existing stage-health metrics include all replicated participants (publishers in the source stage who are replica participants in the destination stage).

Metric	Dimensions	Description
ConcurrentPublishers	—	Number of participants publishing across all stages in an AWS Region. Unit: Count Valid statistics: Average, Maximum, Minimum
ConcurrentSubscriptions	—	Number of simultaneous publisher-to-subscriber connections across all stages in an AWS Region. Unit: Count Valid statistics: Average, Maximum, Minimum
DownloadPacketLoss	—	Percentage of packets that were lost while downloading from the IVS server by subscribers. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of packet loss over the configured interval
DownloadPacketLoss	Platform	Filters DownloadPacketLoss by subscriber platform. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of packet loss over the configured interval
DownloadPacketLoss	Platform, SDKVersion	Filters DownloadPacketLoss by subscriber platform and SDK version. Unit: Percent

Metric	Dimensions	Description
		Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of packet loss over the configured interval
`DownloadP acketLoss`	`Stage`	Filters `DownloadPacketLoss` by subscriber stage. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of packet loss over the configured interval
`DownloadP acketLoss`	`Stage, Participant`	Filters `DownloadPacketLoss` by participant, for subscribers who are also publishers. Samples represent the percentage of packets that were lost by the subscriber while downloading from the IVS server. Samples are emitted only when the participant is also a publisher. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of dropped frames over the configured interval
`DownloadP acketLoss`	`Stage, Platform`	Filters `DownloadPacketLoss` by subscriber stage and platform. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of packet loss over the configured interval

Metric	Dimensions	Description
DownloadPacketLoss	Stage, Platform, SDKVersion	Filters DownloadPacketLoss by subscriber stage, platform, and SDK version. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of packet loss over the configured interval
DownloadPacketLoss	Stage, SubscriberCountryCode	Filters DownloadPacketLoss by subscriber stage and country code (ISO 3166). Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of packet loss over the configured interval
DownloadPacketLoss	SubscriberCountryCode	Filters DownloadPacketLoss by subscriber country code (ISO 3166). Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of packet loss over the configured interval
DroppedFrames	–	For subscribers: the percentage of video frames dropped, calculated by aggregating frames received and frames dropped across all publishers the subscriber is subscribed to. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval

Metric	Dimensions	Description
DroppedFrames	Platform	Filters DroppedFrames by subscriber's platform. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	Platform, SDKVersion	Filters DroppedFrames by subscriber's platform and SDK version. Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	Stage	Filters DroppedFrames by stage. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	Stage, Participant	Filters DroppedFrames by stage and participant. Only emitted for subscribers who are also publishers. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval

Metric	Dimensions	Description
DroppedFrames	Stage, Platform	Filters DroppedFrames by stage and subscriber's platform. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	Stage, Platform, SDKVersion	Filters DroppedFrames by stage and subscriber's platform and SDK version. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	Stage, SubscriberCountryCode	Filters DroppedFrames by stage and subscriber's country. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval
DroppedFrames	SubscriberCountryCode	Filters DroppedFrames by subscriber's country. Unit: Percent Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of dropped frames over the configured interval

Metric	Dimensions	Description
PublishBitrate	–	The total rate at which a publisher is sending both video and audio data (summed across all simulcast layers). This includes retransmitted data. The bitrate can be inflated by upload packet loss and retransmissions, since it reflects what the publisher sends and may not match what IVS receives or delivers to subscribers. Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval
PublishBitrate	Platform	Filters PublishBitrate by publisher's platform. Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval
PublishBitrate	Stage	Filters PublishBitrate by stage. Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval

Metric	Dimensions	Description
PublishBitrate	Stage, Participant, SimulcastLayer, MediaType	Filters PublishBitrate by stage, participant, simulcast layer, and media type. The simulcast layer ID is set by the broadcast SDK. When simulcast is disabled, this layer ID will be set to "disabled". The media type is either "video" or "audio". Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval
Publishers	Stage	Number of participants publishing to the stage. Unit: Count Valid statistics: Average, Maximum, Minimum
PublishFramerate	Stage, Participant	How often video frames are received from a given publisher. This metric is available only for participants publishing over RTMP. Unit: Count/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of framerate over the configured interval
PublishFramerate	Stage, Participant, SimulcastLayer, MediaType	How often video frames are received from a given publisher. This metric is available only for participants publishing over RTMP. Unit: Count/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of framerate over the configured interval

Metric	Dimensions	Description
PublishResolution	Stage, Participant, SimulcastLayer, MediaType	Number of pixels across the smaller of the width or height of the frame. For example, for a landscape frame of size 1920x1080, the PublishResolution is 1080. For a portrait frame of size 720x1280, the PublishResolution is 720. Unit: Count Valid statistics: Average, Maximum, Minimum
SubscribeBitrate	–	The total rate at which a subscriber is receiving both video and audio data Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval
SubscribeBitrate	Platform	Filters SubscribeBitrate by subscriber's platform. Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval
SubscribeBitrate	Platform, SDKVersion	Filters SubscribeBitrate by subscriber's platform and SDK version. Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval

Metric	Dimensions	Description
SubscribeBitrate	Stage	Filters SubscribeBitrate by stage. Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval
SubscribeBitrate	Stage, Participant, MediaType	Filters SubscribeBitrate by stage, participant, and media type. The media type is either "video" or "audio". This metric is emitted only while the subscribing participant is also publishing. Unit: Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval
SubscribeBitrate	Stage, Platform	Filters SubscribeBitrate by stage and subscriber's platform. Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval
SubscribeBitrate	Stage, Platform, SDKVersion	Filters SubscribeBitrate by stage and subscriber's platform and SDK version. Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respecti vely) of bitrate over the configured interval

Metric	Dimensions	Description
SubscribeBitrate	Stage, SubscriberCountryCode	Filters SubscribeBitrate by stage and subscriber's country code. Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval
SubscribeBitrate	SubscriberCountryCode	Filters SubscribeBitrate by subscriber's country code (ISO 3166-1 alpha-2). Bits/second Valid statistics: Average, Maximum, Minimum — Average number, largest number, or smallest number (respectively) of bitrate over the configured interval
Subscribers	Stage	Number of participants subscribed to the stage. Note that participants that are actively publishing and subscribing are counted as both publishers and subscribers. Unit: Count Valid statistics: Average, Maximum, Minimum

IVS Broadcast SDK | Real-Time Streaming

The Amazon Interactive Video Services (IVS) Real-Time Streaming broadcast SDK is for developers who are building applications with Amazon IVS. This SDK is designed to leverage the Amazon IVS architecture and will see continual improvement and new features, alongside Amazon IVS. As a native broadcast SDK, it is designed to minimize the performance impact on your application and on the devices with which your users access your application.

Note that the broadcast SDK is used for both sending and receiving video; i.e., you use the same SDK for hosts and viewers. No separate player SDK needed.

Your application can leverage the key features of the Amazon IVS broadcast SDK:

- **High quality streaming** — The broadcast SDK supports high quality streaming. Capture video from your camera and encode it at up to 720p.
- **Automatic Bitrate Adjustments** — Smartphone users are mobile, so their network conditions can change throughout the course of a broadcast. The Amazon IVS broadcast SDK automatically adjusts the video bitrate to accommodate changing network conditions.
- **Portrait and Landscape Support** — No matter how your users hold their devices, the image appears right-side up and properly scaled. The broadcast SDK supports both portrait and landscape canvas sizes. It automatically manages the aspect ratio when the users rotate their device away from the configured orientation.
- **Secure Streaming** — Your user's broadcasts are encrypted using TLS, so they can keep their streams secure.
- **External Audio Devices** — The Amazon IVS broadcast SDK supports audio jack, USB, and Bluetooth SCO external microphones.

Platform Requirements

Native Platforms

Platform	Supported Versions
Android	9.0+ -- note customers can build with versions 6.0+ but will not be able to use real-time streaming functionality.

Platform	Supported Versions
iOS	14+

IVS supports a minimum of 4 major iOS versions and 6 major Android versions. Our current version support may extend beyond these minimums. Customers will be notified via SDK release notes at least 3 months in advance of a major version no longer being supported.

Desktop Browsers

Browser	Supported Platforms	Supported Versions
Chrome	Windows, macOS	Two major versions (current and most recent prior version)
Firefox	Windows, macOS	Two major versions (current and most recent prior version)
Edge	Windows 8.1+	Two major versions (current and most recent prior version) Excludes Edge Legacy
Safari	macOS	Two major versions (current and most recent prior version)

Mobile Browsers (iOS and Android)

Browser	Supported Platforms	Supported Versions
Chrome	iOS, Android	Two major versions (current and most recent prior version)

Browser	Supported Platforms	Supported Versions
Firefox	Android	Two major versions (current and most recent prior version)
Safari	iOS	Two major versions (current and most recent prior version)

Known Limitations

- On all mobile web browsers, we recommend publishing/subscribing with no more than three simultaneous publishers, due to performance constraints which cause video artifacts and black screens. If you require more publishers, configure audio-only publish and subscribe.
- We do not recommend compositing a stage and broadcasting it to a channel on Android Mobile Web, due to performance considerations and potential crashes. If broadcast functionality is required, integrate the IVS real-time streaming Android broadcast SDK.

Webviews

The Web broadcast SDK does not provide support for webviews or weblike environments (TVs, consoles, etc). For mobile implementations, see the Real-Time Streaming Broadcast SDK Guide for Android and for iOS.

Required Device Access

The broadcast SDK requires access to the device's cameras and microphones, both those built into the device and those connected through Bluetooth, USB, or audio jack.

Support

The broadcast SDK is continually improved. See Amazon IVS Release Notes for available versions and fixed issues. If appropriate, before contacting support, update your version of the broadcast SDK and see if that resolves your issue.

Versioning

The Amazon IVS broadcast SDKs use semantic versioning.

For this discussion, suppose:

- The latest release is 4.1.3.
- The latest release of the prior major version is 3.2.4.
- The latest release of version 1.x is 1.5.6.

Backward-compatible new features are added as minor releases of the latest version. In this case, the next set of new features will be added as version 4.2.0.

Backward-compatible, minor bug fixes are added as patch releases of the latest version. Here, the next set of minor bug fixes will be added as version 4.1.4.

Backward-compatible, major bug fixes are handled differently; these are added to several versions:

- Patch release of the latest version. Here, this is version 4.1.4.
- Patch release of the prior minor version. Here, this is version 3.2.5.
- Patch release of the latest version 1.x release. Here, this is version 1.5.7.

Major bug fixes are defined by the Amazon IVS product team. Typical examples are critical security updates and selected other fixes necessary for customers.

Note: In the examples above, released versions increment without skipping any numbers (e.g., from 4.1.3 to 4.1.4). In reality, one or more patch numbers may remain internal and not be released, so the released version could increment from 4.1.3 to, say, 4.1.6.

IVS Broadcast SDK: Web Guide | Real-Time Streaming

The IVS real-time streaming Web broadcast SDK gives developers the tools to build interactive, real-time experiences on the web. This SDK is for developers who are building web applications with Amazon IVS.

The Web broadcast SDK enables participants to send and receive video. The SDK supports the following operations:

- Join a stage
- Publish media to other participants in the stage
- Subscribe to media from other participants in the stage
- Manage and monitor video and audio published to the stage
- Get WebRTC statistics for each peer connection
- All operations from the IVS low-latency streaming Web broadcast SDK

Latest version of Web broadcast SDK: 1.39.0 (Release Notes)

Reference documentation: For information on the most important methods available in the Amazon IVS Web Broadcast SDK, see https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference. Make sure the most current version of the SDK is selected.

Sample code: The samples below are a good place to get started quickly with the SDK:

- Simple Playback
- Simple Publishing and Subscribing
- Comprehensive React Real-Time Collaboration Demo

Platform requirements: See Amazon IVS Broadcast SDK for a list of supported platforms

Note: Publishing from a browser is convenient for end users because it does not require installing additional software. However, browser-based publishing is subject to the constraints and variability of browser environments. If you need to prioritize stability (for example, for event streaming), we generally recommend publishing from a non-browser source (e.g., OBS Studio or other dedicated encoders), which often have direct access to system resources and avoid browser limitations. For more on non-browser publishing options, see the Stream Ingest documentation.

Getting Started with the IVS Web Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in getting started with the IVS real-time streaming Web broadcast SDK.

Imports

The building blocks for real-time are located in a different namespace than the root broadcasting modules.

Using a Script Tag

The Web broadcast SDK is distributed as a JavaScript library and can be retrieved at https://web-broadcast.live-video.net/1.39.0/amazon-ivs-web-broadcast.js.

The classes and enums defined in the examples below can be found on the global object `IVSBroadcastClient`:

```
const { Stage, SubscribeType } = IVSBroadcastClient;
```

Using npm

To install the `npm` package:

```
npm install amazon-ivs-web-broadcast
```

The classes, enums, and types also can be imported from the package module:

```
import { Stage, SubscribeType, LocalStageStream } from 'amazon-ivs-web-broadcast'
```

Server-Side Rendering Support

The Web Broadcast SDK Stages library cannot be loaded in a server-side context, as it references browser primitives necessary to the functioning of the library when loaded. To work around this, load the library dynamically, as demonstrated in the Web Broadcast Demo using Next and React.

Request Permissions

Your app must request permission to access the user's camera and microphone, and it must be served using HTTPS. (This is not specific to Amazon IVS; it is required for any website that needs access to cameras and microphones.)

Here's an example function showing how you can request and capture permissions for both audio and video devices:

```
async function handlePermissions() {
   let permissions = {
       audio: false,
```

```
        video: false,
    };
    try {
        const stream = await navigator.mediaDevices.getUserMedia({ video: true, audio:
 true });
        for (const track of stream.getTracks()) {
            track.stop();
        }
        permissions = { video: true, audio: true };
    } catch (err) {
        permissions = { video: false, audio: false };
        console.error(err.message);
    }
    // If we still don't have permissions after requesting them display the error
 message
    if (!permissions.video) {
        console.error('Failed to get video permissions.');
    } else if (!permissions.audio) {
        console.error('Failed to get audio permissions.');
    }
}
```

For additional information, see the Permissions API and MediaDevices.getUserMedia().

List Available Devices

To see what devices are available to capture, query the browser's MediaDevices.enumerateDevices() method:

```
const devices = await navigator.mediaDevices.enumerateDevices();
window.videoDevices = devices.filter((d) => d.kind === 'videoinput');
window.audioDevices = devices.filter((d) => d.kind === 'audioinput');
```

Retrieve a MediaStream from a Device

After acquiring the list of available devices, you can retrieve a stream from any number of devices. For example, you can use the `getUserMedia()` method to retrieve a stream from a camera.

If you'd like to specify which device to capture the stream from, you can explicitly set the `deviceId` in the `audio` or `video` section of the media constraints. Alternately, you can omit the `deviceId` and have users select their devices from the browser prompt.

You also can specify an ideal camera resolution using the `width` and `height` constraints. (Read more about these constraints here.) The SDK automatically applies width and height constraints that correspond to your maximum broadcast resolution; however, it's a good idea to also apply these yourself to ensure that the source aspect ratio is not changed after you add the source to the SDK.

For real-time streaming, ensure that media is constrained to 720p resolution. Specifically, your `getUserMedia` and `getDisplayMedia` constraint values for width and height must not exceed 921600 (1280*720) when multiplied together.

```
const videoConfiguration = {
  maxWidth: 1280,
  maxHeight: 720,
  maxFramerate: 30,
}

window.cameraStream = await navigator.mediaDevices.getUserMedia({
   video: {
       deviceId: window.videoDevices[0].deviceId,
       width: {
           ideal: videoConfiguration.maxWidth,
       },
       height: {
           ideal:videoConfiguration.maxHeight,
       },
   },
});
window.microphoneStream = await navigator.mediaDevices.getUserMedia({
   audio: { deviceId: window.audioDevices[0].deviceId },
});
```

Publishing & Subscribing with the IVS Web Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in publishing and subscribing to a stage using the IVS real-time streaming Web broadcast SDK.

Concepts

Three core concepts underlie real-time functionality: stage, strategy, and events. The design goal is minimizing the amount of client-side logic necessary to build a working product.

Stage

The `Stage` class is the main point of interaction between the host application and the SDK. It represents the stage itself and is used to join and leave the stage. Creating and joining a stage requires a valid, unexpired token string from the control plane (represented as `token`). Joining and leaving a stage are simple:

```
const stage = new Stage(token, strategy)

try {
   await stage.join();
} catch (error) {
   // handle join exception
}

stage.leave();
```

Strategy

The `StageStrategy` interface provides a way for the host application to communicate the desired state of the stage to the SDK. Three functions need to be implemented: `shouldSubscribeToParticipant`, `shouldPublishParticipant`, and `stageStreamsToPublish`. All are discussed below.

To use a defined strategy, pass it to the `Stage` constructor. The following is a complete example of an application using a strategy to publish a participant's webcam to the stage and subscribe to all participants. Each required strategy function's purpose is explained in detail in the subsequent sections.

```
const devices = await navigator.mediaDevices.getUserMedia({
   audio: true,
   video: {
        width: { max: 1280 },
        height: { max: 720 },
    }
});
const myAudioTrack = new LocalStageStream(devices.getAudioTracks()[0]);
const myVideoTrack = new LocalStageStream(devices.getVideoTracks()[0]);

// Define the stage strategy, implementing required functions
const strategy = {
```

```
    audioTrack: myAudioTrack,
    videoTrack: myVideoTrack,

    // optional
    updateTracks(newAudioTrack, newVideoTrack) {
        this.audioTrack = newAudioTrack;
        this.videoTrack = newVideoTrack;
    },

    // required
    stageStreamsToPublish() {
        return [this.audioTrack, this.videoTrack];
    },

    // required
    shouldPublishParticipant(participant) {
        return true;
    },

    // required
    shouldSubscribeToParticipant(participant) {
        return SubscribeType.AUDIO_VIDEO;
    }
};

// Initialize the stage and start publishing
const stage = new Stage(token, strategy);
await stage.join();


// To update later (e.g. in an onClick event handler)
strategy.updateTracks(myNewAudioTrack, myNewVideoTrack);
stage.refreshStrategy();
```

Subscribing to Participants

```
shouldSubscribeToParticipant(participant: StageParticipantInfo): SubscribeType
```

When a remote participant joins the stage, the SDK queries the host application about the desired subscription state for that participant. The options are `NONE`, `AUDIO_ONLY`, and `AUDIO_VIDEO`. When returning a value for this function, the host application does not need to worry about the publish state, current subscription state, or stage connection state. If `AUDIO_VIDEO` is returned,

the SDK waits until the remote participant is publishing before it subscribes, and it updates the host application by emitting events throughout the process.

Here is a sample implementation:

```
const strategy = {

   shouldSubscribeToParticipant: (participant) => {
      return SubscribeType.AUDIO_VIDEO;
   }

   // ... other strategy functions
}
```

This is the complete implementation of this function for a host application that always wants all participants to see each other; e.g., a video chat application.

More advanced implementations also are possible. For example, assume the application provides a `role` attribute when creating the token with CreateParticipantToken. The application could use the `attributes` property on `StageParticipantInfo` to selectively subscribe to participants based on the server-provided attributes:

```
const strategy = {

   shouldSubscribeToParticipant(participant) {
      switch (participant.attributes.role) {
         case 'moderator':
            return SubscribeType.NONE;
         case 'guest':
            return SubscribeType.AUDIO_VIDEO;
         default:
            return SubscribeType.NONE;
      }
   }
   // . . . other strategies properties
}
```

This can be used to create a stage where moderators can monitor all guests without being seen or heard themselves. The host application could use additional business logic to let moderators see each other but remain invisible to guests.

Configuration for Subscribing to Participants

```
subscribeConfiguration(participant: StageParticipantInfo): SubscribeConfiguration
```

If a remote participant is being subscribed to (see Subscribing to Participants), the SDK queries the host application about a custom subscribe configuration for that participant. This configuration is optional and allows the host application to control certain aspects of subscriber behavior. For information on what can be configured, see SubscribeConfiguration in the SDK reference documentation.

Here is a sample implementation:

```
const strategy = {

   subscribeConfiguration: (participant) => {
      return {
         jitterBuffer: {
            minDelay: JitterBufferMinDelay.MEDIUM
         }
      }

   // ... other strategy functions
}
```

This implementation updates the jitter-buffer minimum delay for all subscribed participants to a preset of `MEDIUM`.

As with `shouldSubscribeToParticipant`, more advanced implementations are possible. The given `ParticipantInfo` can be used to selectively update the subscribe configuration for specific participants.

We recommend using the default behaviors. Specify custom configuration only if there is a particular behavior you want to change.

Publishing

```
shouldPublishParticipant(participant: StageParticipantInfo): boolean
```

Once connected to the stage, the SDK queries the host application to see if a particular participant should publish. This is invoked only on local participants that have permission to publish based on the provided token.

Here is a sample implementation:

```
const strategy = {

   shouldPublishParticipant: (participant) => {
      return true;
   }

   // . . . other strategies properties
}
```

This is for a standard video chat application where users always want to publish. They can mute and unmute their audio and video, to instantly be hidden or seen/heard. (They also can use publish/unpublish, but that is much slower. Mute/unmute is preferable for use cases where changing visibility often is desirable.)

Choosing Streams to Publish

```
stageStreamsToPublish(): LocalStageStream[];
```

When publishing, this is used to determine what audio and video streams should be published. This is covered in more detail later in Publish a Media Stream.

Updating the Strategy

The strategy is intended to be dynamic: the values returned from any of the above functions can be changed at any time. For example, if the host application does not want to publish until the end user taps a button, you could return a variable from `shouldPublishParticipant` (something like `hasUserTappedPublishButton`). When that variable changes based on an interaction by the end user, call `stage.refreshStrategy()` to signal to the SDK that it should query the strategy for the latest values, applying only things that have changed. If the SDK observes that the `shouldPublishParticipant` value has changed, it starts the publish process. If the SDK queries and all functions return the same value as before, the `refreshStrategy` call does not modify the stage.

If the return value of `shouldSubscribeToParticipant` changes from `AUDIO_VIDEO` to `AUDIO_ONLY`, the video stream is removed for all participants with changed returned values, if a video stream existed previously.

Generally, the stage uses the strategy to most efficiently apply the difference between the previous and current strategies, without the host application needing to worry about all the state required to manage it properly. Because of this, think of calling `stage.refreshStrategy()` as a cheap operation, because it does nothing unless the strategy changes.

Events

A `Stage` instance is an event emitter. Using `stage.on()`, the state of the stage is communicated to the host application. Updates to the host application's UI usually can be supported entirely by the events. The events are as follows:

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participant, state) =>
 {})
stage.on(StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED, (participant, state) =>
 {})
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_REMOVED, (participant, streams) => {})
stage.on(StageEvents.STAGE_STREAM_ADAPTION_CHANGED, (participant, stream, isAdapting)
 => ())
stage.on(StageEvents.STAGE_STREAM_LAYERS_CHANGED, (participant, stream, layers) => ())
stage.on(StageEvents.STAGE_STREAM_LAYER_SELECTED, (participant, stream, layer, reason)
 => ())
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {})
stage.on(StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED, (participant, stream) => {})
```

For most of these events, the corresponding `ParticipantInfo` is provided.

It is not expected that the information provided by the events impacts the return values of the strategy. For example, the return value of `shouldSubscribeToParticipant` is not expected to change when `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` is called. If the host application wants to subscribe to a particular participant, it should return the desired subscription type regardless of that participant's publish state. The SDK is responsible for ensuring that the desired state of the strategy is acted on at the correct time based on the state of the stage.

Publish a Media Stream

Local devices like microphones and cameras are retrieved using the same steps as outlined above in Retrieve a MediaStream from a Device. In the example we use `MediaStream` to create a list of `LocalStageStream` objects used for publishing by the SDK:

```
try {
    // Get stream using steps outlined in document above
    const stream = await getMediaStreamFromDevice();

    let streamsToPublish = stream.getTracks().map(track => {
        new LocalStageStream(track)
    });

    // Create stage with strategy, or update existing strategy
    const strategy = {
        stageStreamsToPublish: () => streamsToPublish
    }
}
```

Publish a Screenshare

Applications often need to publish a screenshare in addition to the user's web camera. Publishing a screenshare necessitates creating an additional token for the stage, specifically for publishing the screenshare's media. Use `getDisplayMedia` and constrain the resolution to a maximum of 720p. After that, the steps are similar to publishing a camera to the stage.

```
// Invoke the following lines to get the screenshare's tracks
const media = await navigator.mediaDevices.getDisplayMedia({
   video: {
      width: {
         max: 1280,
      },
      height: {
         max: 720,
      }
   }
});
const screenshare = { videoStream: new LocalStageStream(media.getVideoTracks()[0]) };
const screenshareStrategy = {
   stageStreamsToPublish: () => {
      return [screenshare.videoStream];
```

```
   },
   shouldPublishParticipant: (participant) => {
      return true;
   },
   shouldSubscribeToParticipant: (participant) => {
      return SubscribeType.AUDIO_VIDEO;
   }
}
const screenshareStage = new Stage(screenshareToken, screenshareStrategy);
await screenshareStage.join();
```

Display and Remove Participants

After subscribing is completed, you receive an array of `StageStream` objects through the `STAGE_PARTICIPANT_STREAMS_ADDED` event. The event also gives you participant info to help when displaying media streams:

```
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    let streamsToDisplay = streams;

    if (participant.isLocal) {
        // Ensure to exclude local audio streams, otherwise echo will occur
        streamsToDisplay = streams.filter(stream => stream.streamType ===
 StreamType.VIDEO)
    }

    // Create or find video element already available in your application
    const videoEl = getParticipantVideoElement(participant.id);

    // Attach the participants streams
    videoEl.srcObject = new MediaStream();
    streamsToDisplay.forEach(stream =>
 videoEl.srcObject.addTrack(stream.mediaStreamTrack));
})
```

When a participant stops publishing or is unsubscribed from a stream, the `STAGE_PARTICIPANT_STREAMS_REMOVED` function is called with the streams that were removed. Host applications should use this as a signal to remove the participant's video stream from the DOM.

`STAGE_PARTICIPANT_STREAMS_REMOVED` is invoked for all scenarios in which a stream might be removed, including:

- The remote participant stops publishing.
- A local device unsubscribes or changes subscription from `AUDIO_VIDEO` to `AUDIO_ONLY`.
- The remote participant leaves the stage.
- The local participant leaves the stage.

Because `STAGE_PARTICIPANT_STREAMS_REMOVED` is invoked for all scenarios, no custom business logic is required around removing participants from the UI during remote or local leave operations.

Mute and Unmute Media Streams

`LocalStageStream` objects have a `setMuted` function that controls whether the stream is muted. This function can be called on the stream before or after it is returned from the `stageStreamsToPublish` strategy function.

Important: If a new `LocalStageStream` object instance is returned by `stageStreamsToPublish` after a call to `refreshStrategy`, the mute state of the new stream object is applied to the stage. Be careful when creating new `LocalStageStream` instances to make sure the expected mute state is maintained.

Monitor Remote Participant Media Mute State

When participants change the mute state of their video or audio, the `STAGE_STREAM_MUTE_CHANGED` event is triggered with a list of streams that have changed. Use the `isMuted` property on `StageStream` to update your UI accordingly:

```
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {
   if (stream.streamType === 'video' && stream.isMuted) {
       // handle UI changes for video track getting muted
   }
})
```

Also, you can look at StageParticipantInfo for state information on whether audio or video is muted:

```
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {
   if (participant.videoStopped || participant.audioMuted) {
       // handle UI changes for either video or audio
   }
```

```
})
```

Get WebRTC Statistics

The `requestQualityStats()` method provides access to detailed WebRTC statistics for both local and remote streams. This is available on both LocalStageStream and RemoteStageStream objects. It returns comprehensive quality metrics including network quality, packet statistics, bitrate information, and frame-related metrics.

This is an asynchronous method with which you can retrieve statistics either via await or by chaining a promise. It returns `undefined` when statistics are not available; e.g., the stream is not active or internal statistics are unavailable. If statistics are available, and depending on the stream (remote or local, video or audio), the method returns a LocalVideoStats, LocalAudioStats, RemoteVideoStats, or RemoteAudioStats object.

Note that for video streams with simulcast, the array contains multiple stat objects (one per layer).

Best Practices

- Polling frequency — Call `requestQualityStats()` at reasonable intervals (1-5 seconds) to avoid performance impact
- Error handling — Always check if the returned value is `undefined` before processing
- Memory management — Clear intervals/timeouts when streams are no longer needed
- Network quality — Use `networkQuality` for user feedback regarding possible degradations caused by the network. For details, see NetworkQuality.

Example Usage

```
// For local streams
const localStats = await localVideoStream.requestQualityStats();
const audioStats = await localAudioStream.requestQualityStats();

// For remote streams
const remoteVideoStats = await remoteVideoStream.requestQualityStats();
const remoteAudioStats = await remoteAudioStream.requestQualityStats();

// Example: Monitor stats every 10 seconds
const statsInterval = setInterval(async () => {
   const stats = await localVideoStream.requestQualityStats();
   if (stats) {
```

```
      // Note: If simulcast is enabled, you may receive multiple
      // stats records for each layer
      stats.forEach(layer => {
         const rid = layer.rid || 'default';
         console.log(`Layer ${rid}:`, {
            active: layer.active,
            networkQuality: layer.networkQuality,
            packetsSent: layer.packetsSent,
            bytesSent: layer.bytesSent,
            resolution: `${layer.frameWidth}x${layer.frameHeight}`,
            fps: layer.framesPerSecond
         });
      });
   }
}, 10000);
```

Optimizing Media

It's recommended to limit `getUserMedia` and `getDisplayMedia` calls to the following constraints for the best performance:

```
const CONSTRAINTS = {
    video: {
        width: { ideal: 1280 }, // Note: flip width and height values if portrait is
 desired
        height: { ideal: 720 },
        framerate: { ideal: 30 },
    },
};
```

You can further constrain the media through additional options passed to the `LocalStageStream` constructor:

```
const localStreamOptions = {
    minBitrate?: number;
    maxBitrate?: number;
    maxFramerate?: number;
    simulcast: {
        enabled: boolean
    }
}
const localStream = new LocalStageStream(track, localStreamOptions)
```

In the code above:

- `minBitrate` sets a minimum bitrate that the browser should be expected to use. However, a low complexity video stream may push the encoder to go lower than this bitrate.
- `maxBitrate` sets a maximum bitrate that the browser should be expected to not exceed for this stream.
- `maxFramerate` sets a maximum frame rate that the browser should be expected to not exceed for this stream.
- The `simulcast` option is usable only on Chromium-based browsers. It enables sending three rendition layers of the stream.
  - This allows the server to choose which rendition to send to other participants, based on their networking limitations.
  - When `simulcast` is specified along with a `maxBitrate` and/or `maxFramerate` value, it is expected that the highest rendition layer will be configured with these values in mind, provided the `maxBitrate` does not go below the internal SDK's second highest layer's default `maxBitrate` value of 900 kbps.
  - If `maxBitrate` is specified as too low compared to the second highest layer's default value, `simulcast` will be disabled.
  - `simulcast` cannot be toggled on and off without republishing the media through a combination of having `shouldPublishParticipant` return `false`, calling `refreshStrategy`, having `shouldPublishParticipant` return `true` and calling `refreshStrategy` again.

Get Participant Attributes

If you specify attributes in the `CreateParticipantToken` operation request, you can see the attributes in `StageParticipantInfo` properties:

```
stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
   console.log(`Participant ${participant.id} info:`, participant.attributes);
})
```

Supplemental Enhancement Information (SEI)

The Supplemental Enhancement Information (SEI) NAL unit is used to store frame-aligned metadata alongside the video. It can be used when publishing and subscribing to H.264 video

streams. SEI payloads are not guaranteed to arrive to subscribers, especially in bad network conditions. As the SEI payload stores data directly within the H.264 frame structure, this capability cannot be leveraged for audio-only streams.

Inserting SEI Payloads

Publishing clients can insert SEI payloads to a stage stream that is being published by configuring their video's LocalStageStream to enable `inBandMessaging` and subsequently invoking the `insertSeiMessage` method. Note that enabling `inBandMessaging` increases SDK memory usage.

Payloads must be of the ArrayBuffer type. The payload size must be greater than 0KB and less than 1KB. The number of SEI messages inserted per second must not exceed 10KB per second.

```
const config = {
    inBandMessaging: { enabled: true }
};
const vidStream = new LocalStageStream(videoTrack, config);
const payload = new TextEncoder().encode('hello world').buffer;
vidStream.insertSeiMessage(payload);
```

Repeating SEI Payloads

Optionally provide a `repeatCount` to repeat the insertion of SEI payloads for the next N frames sent. This could be helpful to mitigate the inherent loss that may occur due to the underlying UDP transport protocol used to send video. Note this value must be between 0 and 30. Receiving clients must have logic to de-duplicate the message.

```
vidStream.insertSeiMessage(payload, { repeatCount: 5 }); // Optional config,
 repeatCount must be between 0 and 30
```

Reading SEI Payloads

Subscribing clients can read SEI payloads from a publisher who is publishing H.264 video if present by configuring the subscriber(s) `SubscribeConfiguration` to enable `inBandMessaging` and listening to the `StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED` event, as shown in the following example:

```
const strategy = {
```

```
    subscribeConfiguration: (participant) => {
        return {
            inBandMessaging: {
                enabled: true
            }
        }
    }
    // ... other strategy functions
}

stage.on(StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED, (participant, seiMessage) => {
    console.log(seiMessage.payload, seiMessage.uuid);
});
```

Layered Encoding with Simulcast

Layered encoding with simulcast is an IVS real-time streaming feature that allows publishers to send multiple different quality layers of video, and subscribers to dynamically or manually change those layers. The feature is described more in the Streaming Optimizations document.

Configuring Layered Encoding (Publisher)

As a publisher, to enable layered encoding with simulcast, add the following configuration to your `LocalStageStream` on instantiation:

```
// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})
```

Depending on the input resolution of your camera device, a set number of layers will be encoded and sent as defined in the Default Layers, Qualities, and Framerates section of *Streaming Optimizations*.

Also, you can optionally configure individual layers from within the simulcast configuration:

```
import { SimulcastLayerPresets } from 'amazon-ivs-web-broadcast'

// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: {
```

```
      enabled: true,
      layers: [
         SimulcastLayerPresets.DEFAULT_720,
          SimulcastLayerPresets.DEFAULT_360,
          SimulcastLayerPresets.DEFAULT_180,
   }
})
```

Alternately, you can create your own custom layer configurations for up to three layers. If you provide an empty array or no value, the defaults described above are used. Layers are described with the following required properties:

- `height: number;`
- `width: number;`
- `maxBitrateKbps: number;`
- `maxFramerate: number;`

Starting from the presets, you can either override individual properties or create an entirely new configuration:

```
import { SimulcastLayerPresets } from 'amazon-ivs-web-broadcast'

const custom720pLayer = {
   ...SimulcastLayerPresets.DEFAULT_720,
   maxFramerate: 15,
}

const custom360pLayer = {
       maxBitrateKbps: 600,
       maxFramerate: 15,
       width: 640,
       height: 360,
}

// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: {
      enabled: true,
      layers: [
         custom720pLayer,
         custom360pLayer,
```

```
    }
})
```

For maximum values, limits, and errors which can be triggered when configuring individual layers, see the SDK reference documentation.

Configuring Layered Encoding (Subscriber)

As a subscriber, there is nothing needed to enable layered encoding. If a publisher is sending simulcast layers, then by default the server dynamically adapts between the layers to choose the optimal quality based on the subscriber's device and network conditions.

Alternatively, to pick explicit layers that the publisher is sending, there are several options, described below.

Option 1: Initial Layer Quality Preference

Using the `subscribeConfiguration` strategy, it is possible to choose what initial layer you want to receive as a subscriber:

```
const strategy = {
    subscribeConfiguration: (participant) => {
        return {
            simulcast: {
                initialLayerPreference: InitialLayerPreference.LOWEST_QUALITY
            }
        }
    }
    // ... other strategy functions
}
```

By default, subscribers always are sent the lowest quality layer first; this slowly ramps up to the highest quality layer. This optimizes end-user bandwidth consumption and provides the best time to video, reducing initial video freezes for users on weaker networks.

These options are available for `InitialLayerPreference`:

- `LOWEST_QUALITY` — The server delivers the lowest quality layer of video first. This optimizes bandwidth consumption, as well as time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 720p video is lower quality than 1080p video.

- `HIGHEST_QUALITY` — The server delivers the highest quality layer of video first. This optimizes quality but may increase the time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 1080p video is higher quality than 720p video.

Note: For initial layer preferences (the `initialLayerPreference` call) to take effect, a re-subscribe is necessary as these updates do not apply to the active subscription.

Option 2: Preferred Layer for Stream

Once a stream has started, you can use the `preferredLayerForStream` strategy method. This strategy method exposes the participant and the stream information.

The strategy method can be returned with the following:

- The layer object directly, based on what `RemoteStageStream.getLayers` returns
- The layer object label string, based on `StageStreamLayer.label`
- Undefined or null, which indicates that no layer should be selected, and dynamic adaption is preferred

For example, the following strategy will always have the users selecting the lowest quality layer of video available:

```
const strategy = {
    preferredLayerForStream: (participant, stream) => {
        return stream.getLowestQualityLayer();
    }
    // ... other strategy functions
}
```

To reset the layer selection and return to dynamic adaption, return null or undefined in the strategy. In this example `appState` is a dummy variable that represents the possible application state.

```
const strategy = {
    preferredLayerForStream: (participant, stream) => {
        if (appState.isAutoMode) {
            return null;
        } else {
            return appState.layerChoice
```

```
        }
    }
    // ... other strategy functions
}
```

Option 3: RemoteStageStream Layer Helpers

`RemoteStageStream` has several helpers which can be used to make decisions about layer selection and display the corresponding selections to end users:

- **Layer Events** — Alongside `StageEvents`, the `RemoteStageStream` object itself has events which communicate layer and simulcast adaption changes:
  - `stream.on(RemoteStageStreamEvents.ADAPTION_CHANGED, (isAdapting) => {})`
  - `stream.on(RemoteStageStreamEvents.LAYERS_CHANGED, (layers) => {})`
  - `stream.on(RemoteStageStreamEvents.LAYER_SELECTED, (layer, reason) => {})`
- **Layer Methods** — `RemoteStageStream` has several helper methods which can be used to get information about the stream and the layers being presented. These methods are available on the remote stream provided in the `preferredLayerForStream` strategy, as well as remote streams exposed via `StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED`.
  - `stream.getLayers`
  - `stream.getSelectedLayer`
  - `stream.getLowestQualityLayer`
  - `stream.getHighestQualityLayer`

For details, see the `RemoteStageStream` class in the SDK reference documentation. For the `LAYER_SELECTED` reason, if `UNAVAILABLE` is returned, this indicates that the requested layer could not be selected. A best-effort selection is made in its place, which typically is a lower quality layer to maintain stream stability.

Handling Network Issues

When the local device's network connection is lost, the SDK internally tries to reconnect without any user action. In some cases, the SDK is not successful and user action is needed.

Broadly the state of the stage can be handled via the `STAGE_CONNECTION_STATE_CHANGED` event:

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
   switch (state) {
      case StageConnectionState.DISCONNECTED:
         // handle disconnected UI
         return;
      case StageConnectionState.CONNECTING:
         // handle establishing connection UI
         return;
      case StageConnectionState.CONNECTED:
         // SDK is connected to the Stage
         return;
      case StageConnectionState.ERRORED:
         // SDK encountered an error and lost its connection to the stage. Wait for
 CONNECTED.
         return;
    }
})
```

In general, you can ignore an errored state that is encountered after successfully joining a stage, as the SDK will try to recover internally. If the SDK reports an `ERRORED` state and the stage remains in the `CONNECTING` state for an extended period of time (e.g., 30 seconds or longer), you probably are disconnected from the network.

Broadcast the Stage to an IVS Channel

To broadcast a stage, create a separate `IVSBroadcastClient` session and then follow the usual instructions for broadcasting with the SDK, described above. The list of `StageStream` exposed via `STAGE_PARTICIPANT_STREAMS_ADDED` can be used to retrieve the participant media streams which can be applied to the broadcast stream composition, as follows:

```
// Setup client with preferred settings
const broadcastClient = getIvsBroadcastClient();

stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    streams.forEach(stream => {
        const inputStream = new MediaStream([stream.mediaStreamTrack]);
        switch (stream.streamType) {
            case StreamType.VIDEO:
                broadcastClient.addVideoInputDevice(inputStream, `video-
${participant.id}`, {
                    index: DESIRED_LAYER,
```

```
                    width: MAX_WIDTH,
                    height: MAX_HEIGHT
                });
                break;
            case StreamType.AUDIO:
                broadcastClient.addAudioInputDevice(inputStream, `audio-
${participant.id}`);
                break;
        }
    })
})
```

Optionally, you can composite a stage and broadcast it to an IVS low-latency channel, to reach a larger audience. See Enabling Multiple Hosts on an Amazon IVS Stream in the IVS Low-Latency Streaming User Guide.

Error Handling in the IVS Web Broadcast SDK | Real-Time Streaming

This section is an overview of error conditions, how the Web broadcast SDK reports them to the application, and what an application should do when those errors are encountered. Errors are reported by the SDK to listeners of the `StageEvents.ERROR` event:

```
stage.on(StageEvents.ERROR, (error: StageError) => {
    // log or handle errors here
    console.log(`${error.code}, ${error.category}, ${error.message}`);
});
```

Stage Errors

A StageError is reported when the SDK encounters a problem it cannot recover from and generally requires app intervention and/or network reconnection to recover.

Each reported `StageError` has a code (or `StageErrorCode`), message (string), and category (`StageErrorCategory`). Each is related to an underlying operation category.

The operation category of the error is determined based on whether it is related to the connection to the stage (`JOIN_ERROR`), sending media to the stage (`PUBLISH_ERROR`), or receiving an incoming media stream from the stage (`SUBSCRIBE_ERROR`).

The code property of a `StageError` reports the specific problem:

Name	Code	Recommended Action
TOKEN_MALFORMED	1	Create a valid token and retry instantiating the stage.
TOKEN_EXPIRED	2	Create an unexpired token and retry instantiating the stage.
TIMEOUT	3	The operation timed out. If the stage exists and the token is valid, this failure likely is a network issue. In that case, wait for the device's connectivity to recover.
FAILED	4	A fatal condition was encountered when attempting an operation. Check error details. If the stage exists and the token is valid, this failure likely is a network issue. In that case, wait for the device's connectivity to recover. For most failures related to network stability, the SDK will retry internally for a period of up to 30 seconds before emitting a FAILED error.
CANCELED	5	Check application code and ensure there are no repeated `join`, `refreshStrategy` , or `replaceStrategy` invocations, which may cause repeated operations to be started and canceled before completion.
STAGE_AT_CAPACITY	6	This error indicates that the stage or your account is at capacity. If the stage has reached its participant limit, try the operation again when the stage is no longer at capacity, by refreshing the strategy. If your account has reached its concurrent subscriptions or concurrent publishers quota, reduce usage or request a quota increase through the AWS Service Quotas console.
CODEC_MISMATCH	7	The codec is not supported by the stage. Check the browser and platform for codec support. For IVS real-

Name	Code	Recommended Action
		time streaming, browsers must support the H.264 codec for video and the Opus codec for audio.
TOKEN_NOT_ALLOWED	8	The token does not have permission for the operation . Recreate the token with the correct permission(s) and try again.
STAGE_DELETED	9	None; attempting to join a deleted stage triggers this error.
PARTICIPANT_DISCONNECTED	10	None; attempting to join with a token of a disconnected participant triggers this error.

Handling StageError Example

Use the StageError code to determine if the error is due to an expired token:

```
stage.on(StageEvents.ERROR, (error: StageError) => {
    if (error.code === StageError.TOKEN_EXPIRED) {
        // recreate the token and stage instance and re-join
    }
});
```

Network Errors when Already Joined

If the device's network connection goes down, the SDK may lose its connection to stage servers. You may see errors in the console because the SDK can no longer reach backend services. POSTs to https://broadcast.stats.live-video.net will fail.

If you are publishing and/or subscribing, you will see errors in the console related to attempts to publish/subscribe.

Internally the SDK will try to reconnect with an exponential backoff strategy.

Action: Wait for the device's connectivity to recover.

Errored States

We recommend you use these states for application logging and to display messaging to users that alerts them of connectivity issues to the stage for a particular participant.

Publish

The SDK reports ERRORED when a publish fails.

```
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participantInfo, state)
 => {
  if (state === StageParticipantPublishState.ERRORED) {
      // Log and/or display message to user
  }
});
```

Subscribe

The SDK reports ERRORED when a subscribe fails. This can occur due to network conditions or if a stage is at capacity for subscribers.

```
stage.on(StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED, (participantInfo,
 state) => {
  if (state === StageParticipantSubscribeState.ERRORED) {
    // Log and/or display message to user
  }
});
```

IVS Broadcast SDK: Android Guide | Real-Time Streaming

The IVS real-time streaming Android broadcast SDK enables participants to send and receive video on Android.

The `com.amazonaws.ivs.broadcast` package implements the interface described in this document. The SDK supports the following operations:

- Join a stage
- Publish media to other participants in the stage
- Subscribe to media from other participants in the stage

- Manage and monitor video and audio published to the stage
- Get WebRTC statistics for each peer connection
- All operations from the IVS low-latency streaming Android broadcast SDK

Latest version of Android broadcast SDK: 1.46.0 (Release Notes)

Reference documentation: For information on the most important methods available in the Amazon IVS Android broadcast SDK, see the reference documentation at https://aws.github.io/amazon-ivs-broadcast-docs/1.46.0/android/.

Sample code: See the Android sample repository on GitHub: https://github.com/aws-samples/amazon-ivs-real-time-streaming-android-samples.

Platform requirements: Android 9.0+

Getting Started with the IVS Android Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in getting started with the IVS real-time streaming Android broadcast SDK.

Install the Library

There are several ways to add the Amazon IVS Android broadcast library to your Android development environment: use Gradle directly, use Gradle version catalogs, or install the SDK manually.

Use Gradle directly: Add the library to your module's `build.gradle` file, as shown here (for the latest version of the IVS broadcast SDK):

```
repositories {
    mavenCentral()
}

dependencies {
     implementation 'com.amazonaws:ivs-broadcast:1.46.0:stages@aar'
}
```

Use Gradle version catalogs: First include this in your module's `build.gradle` file:

```
implementation(libs.ivs){
   artifact {
      classifier = "stages"
      type = "aar"
   }
}
```

Then include the following in the `libs.version.toml` file (for the latest version of the IVS broadcast SDK):

```
[versions]
ivs="1.46.0"

[libraries]
ivs = {module = "com.amazonaws:ivs-broadcast", version.ref = "ivs"}
```

Install the SDK manually: Download the latest version from this location:

https://search.maven.org/artifact/com.amazonaws/ivs-broadcast

Be sure to download the `aar` with `-stages` appended.

Also allow SDK control over the speakerphone: Regardless of which installation method you choose, also add the following permission to your manifest, to allow the SDK to enable and disable the speakerphone:

```
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
```

Using the SDK with Debug Symbols

We also publish a version of the Android broadcast SDK which includes debug symbols. You can use this version to improve the quality of debug reports (stack traces) in Firebase Crashlytics, if you run into crashes in the IVS broadcast SDK; i.e., `libbroadcastcore.so`. When you report these crashes to the IVS SDK team, the higher quality stack traces make it easier to fix the issues.

To use this version of the SDK, put the following in your Gradle build files:

```
implementation "com.amazonaws:ivs-broadcast:$version:stages-unstripped@aar"
```

Use the above line instead of this:

```
implementation "com.amazonaws:ivs-broadcast:$version:stages@aar"
```

Uploading Symbols to Firebase Crashlytics

Ensure that your Gradle build files are set up for Firebase Crashlytics. Follow Google's instructions here:

https://firebase.google.com/docs/crashlytics/ndk-reports

Be sure to include `com.google.firebase:firebase-crashlytics-ndk` as a dependency.

When building your app for release, the Firebase Crashlytics plugin should upload symbols automatically. To upload symbols manually, run either of the following:

```
gradle uploadCrashlyticsSymbolFileRelease
```

```
./gradlew uploadCrashlyticsSymbolFileRelease
```

(It will not hurt if symbols are uploaded twice, both automatically and manually.)

Preventing your Release .apk from Becoming Larger

Before packaging the release `.apk` file, the Android Gradle Plugin automatically tries to strip debug information from shared libraries (including the IVS broadcast SDK's `libbroadcastcore.so` library). However, sometimes this does not happen. As a result, your `.apk` file could become larger and you could get a warning message from the Android Gradle Plugin that it's unable to strip debug symbols and is packaging `.so` files as is. If this happens, do the following:

- Install an Android NDK. Any recent version will work.
- Add `ndkVersion <your_installed_ndk_version_number>` to your application's `build.gradle` file. Do this even if your application itself does not contain native code.

For more information, see this issue report.

Request Permissions

Your app must request permission to access the user's camera and mic. (This is not specific to Amazon IVS; it is required for any application that needs access to cameras and microphones.)

Here, we check whether the user has already granted permissions and, if not, ask for them:

```
final String[] requiredPermissions =
         { Manifest.permission.CAMERA, Manifest.permission.RECORD_AUDIO };

for (String permission : requiredPermissions) {
    if (ContextCompat.checkSelfPermission(this, permission)
                != PackageManager.PERMISSION_GRANTED) {
        // If any permissions are missing we want to just request them all.
        ActivityCompat.requestPermissions(this, requiredPermissions, 0x100);
        break;
    }
}
```

Here, we get the user's response:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode,
               permissions, grantResults);
    if (requestCode == 0x100) {
        for (int result : grantResults) {
            if (result == PackageManager.PERMISSION_DENIED) {
                return;
            }
        }
        setupBroadcastSession();
    }
}
```

Publishing & Subscribing with the IVS Android Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in publishing and subscribing to a stage using the IVS real-time streaming Android broadcast SDK.

Concepts

Three core concepts underlie real-time functionality: stage, strategy, and renderer. The design goal is minimizing the amount of client-side logic necessary to build a working product.

Stage

The `Stage` class is the main point of interaction between the host application and the SDK. It represents the stage itself and is used to join and leave the stage. Creating and joining a stage requires a valid, unexpired token string from the control plane (represented as `token`). Joining and leaving a stage are simple.

```
Stage stage = new Stage(context, token, strategy);

try {
 stage.join();
} catch (BroadcastException exception) {
 // handle join exception
}

stage.leave();
```

The `Stage` class is also where the `StageRenderer` can be attached:

```
stage.addRenderer(renderer); // multiple renderers can be added
```

Strategy

The `Stage.Strategy` interface provides a way for the host application to communicate the desired state of the stage to the SDK. Three functions need to be implemented: `shouldSubscribeToParticipant`, `shouldPublishFromParticipant`, and `stageStreamsToPublishForParticipant`. All are discussed below.

Subscribing to Participants

```
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo);
```

When a remote participant joins the stage, the SDK queries the host application about the desired subscription state for that participant. The options are `NONE`, `AUDIO_ONLY`, and `AUDIO_VIDEO`. When returning a value for this function, the host application does not need to worry about the publish state, current subscription state, or stage connection state. If `AUDIO_VIDEO` is returned, the SDK waits until the remote participant is publishing before subscribing, and it updates the host application through the renderer throughout the process.

Here is a sample implementation:

```
@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 return Stage.SubscribeType.AUDIO_VIDEO;
}
```

This is the complete implementation of this function for a host application that always wants all participants to see each other; e.g., a video chat application.

More advanced implementations also are possible. Use the `userInfo` property on `ParticipantInfo` to selectively subscribe to participants based on server-provided attributes:

```
@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 switch(participantInfo.userInfo.get("role")) {
  case "moderator":
   return Stage.SubscribeType.NONE;
  case "guest":
   return Stage.SubscribeType.AUDIO_VIDEO;
  default:
   return Stage.SubscribeType.NONE;
 }
}
```

This can be used to create a stage where moderators can monitor all guests without being seen or heard themselves. The host application could use additional business logic to let moderates see each other but remain invisible to guests.

Configuration for Subscribing to Participants

```
SubscribeConfiguration subscribeConfigurationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo);
```

If a remote participant is being subscribed to (see Subscribing to Participants), the SDK queries the host application about a custom subscribe configuration for that participant. This configuration is optional and allows the host application to control certain aspects of subscriber behavior. For information on what can be configured, see SubscribeConfiguration in the SDK reference documentation.

Here is a sample implementation:

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.jitterBuffer.setMinDelay(JitterBufferConfiguration.JitterBufferDelay.MEDIUM());

    return config;
}
```

This implementation updates the jitter-buffer minimum delay for all subscribed participants to a preset of `MEDIUM`.

As with `shouldSubscribeToParticipant`, more advanced implementations are possible. The given `ParticipantInfo` can be used to selectively update the subscribe configuration for specific participants.

We recommend using the default behaviors. Specify custom configuration only if there is a particular behavior you want to change.

Publishing

```
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo);
```

Once connected to the stage, the SDK queries the host application to see if a particular participant should publish. This is invoked only on local participants that have permission to publish based on the provided token.

Here is a sample implementation:

```
@Override
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 return true;
}
```

This is for a standard video chat application where users always want to publish. They can mute and unmute their audio and video, to instantly be hidden or seen/heard. (They also can use

publish/unpublish, but that is much slower. Mute/unmute is preferable for use cases where changing visibility often is desirable.)

Choosing Streams to Publish

```
@Override
List<LocalStageStream> stageStreamsToPublishForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo);
}
```

When publishing, this is used to determine what audio and video streams should be published. This is covered in more detail later in Publish a Media Stream.

Updating the Strategy

The strategy is intended to be dynamic: the values returned from any of the above functions can be changed at any time. For example, if the host application does not want to publish until the end user taps a button, you could return a variable from `shouldPublishFromParticipant` (something like `hasUserTappedPublishButton`). When that variable changes based on an interaction by the end user, call `stage.refreshStrategy()` to signal to the SDK that it should query the strategy for the latest values, applying only things that have changed. If the SDK observes that the `shouldPublishFromParticipant` value has changed, it will start the publish process. If the SDK queries and all functions return the same value as before, the `refreshStrategy` call will not perform any modifications to the stage.

If the return value of `shouldSubscribeToParticipant` changes from `AUDIO_VIDEO` to `AUDIO_ONLY`, the video stream will be removed for all participants with changed returned values, if a video stream existed previously.

Generally, the stage uses the strategy to most efficiently apply the difference between the previous and current strategies, without the host application needing to worry about all the state required to manage it properly. Because of this, think of calling `stage.refreshStrategy()` as a cheap operation, because it does nothing unless the strategy changes.

Renderer

The `StageRenderer` interface communicates the state of the stage to the host application. Updates to the host application's UI usually can be powered entirely by the events provided by the renderer. The renderer provides the following functions:

```
void onParticipantJoined(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo);

void onParticipantLeft(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo);

void onParticipantPublishStateChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull Stage.PublishState publishState);

void onParticipantSubscribeStateChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull Stage.SubscribeState subscribeState);

void onStreamsAdded(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo,
 @NonNull List<StageStream> streams);

void onStreamsRemoved(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo,
 @NonNull List<StageStream> streams);

void onStreamsMutedChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull List<StageStream> streams);

void onError(@NonNull BroadcastException exception);

void onConnectionStateChanged(@NonNull Stage stage, @NonNull Stage.ConnectionState
 state, @Nullable BroadcastException exception);

void onStreamAdaptionChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, boolean adaption);

void onStreamLayersChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, @NonNull
 List<RemoteStageStream.Layer> layers);

void onStreamLayerSelected(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, @Nullable RemoteStageStream.Layer
 layer, @NonNull RemoteStageStream.LayerSelectedReason reason);
```

For most of these methods, the corresponding `Stage` and `ParticipantInfo` are provided.

It is not expected that the information provided by the renderer impacts the return values of the strategy. For example, the return value of `shouldSubscribeToParticipant` is not expected to change when on`ParticipantPublishStateChanged` is called. If the host application wants to subscribe to a particular participant, it should return the desired subscription type regardless of

that participant's publish state. The SDK is responsible for ensuring that the desired state of the strategy is acted on at the correct time based on the state of the stage.

The `StageRenderer` can be attached to the stage class:

```
stage.addRenderer(renderer); // multiple renderers can be added
```

Note that only publishing participants trigger on`ParticipantJoined`, and whenever a participant stops publishing or leaves the stage session, on`ParticipantLeft` is triggered.

Publish a Media Stream

Local devices such as built-in microphones and cameras are discovered via `DeviceDiscovery`. Here is an example of selecting the front-facing camera and default microphone, then return them as `LocalStageStreams` to be published by the SDK:

```
DeviceDiscovery deviceDiscovery = new DeviceDiscovery(context);

List<Device> devices = deviceDiscovery.listLocalDevices();
List<LocalStageStream> publishStreams = new ArrayList<LocalStageStream>();

Device frontCamera = null;
Device microphone = null;

// Create streams using the front camera, first microphone
for (Device device : devices) {
 Device.Descriptor descriptor = device.getDescriptor();
 if (!frontCamera && descriptor.type == Device.Descriptor.DeviceType.Camera &&
 descriptor.position = Device.Descriptor.Position.FRONT) {
  front Camera = device;
 }
 if (!microphone && descriptor.type == Device.Descriptor.DeviceType.Microphone) {
  microphone = device;
 }
}

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera);
AudioLocalStageStream microphoneStream = new AudioLocalStageStream(microphoneDevice);

publishStreams.add(cameraStream);
publishStreams.add(microphoneStream);
```

```
// Provide the streams in Stage.Strategy
@Override
@NonNull List<LocalStageStream> stageStreamsToPublishForParticipant(@NonNull Stage
 stage, @NonNull ParticipantInfo participantInfo) {
 return publishStreams;
}
```

Display and Remove Participants

After subscribing is completed, you will receive an array of `StageStream` objects through the renderer's on`StreamsAdded` function. You can retrieve the preview from an `ImageStageStream`:

```
ImagePreviewView preview = ((ImageStageStream)stream).getPreview();

// Add the view to your view hierarchy
LinearLayout previewHolder = findViewById(R.id.previewHolder);
preview.setLayoutParams(new LinearLayout.LayoutParams(
  LinearLayout.LayoutParams.MATCH_PARENT,
  LinearLayout.LayoutParams.MATCH_PARENT));
previewHolder.addView(preview);
```

You can retrieve the audio-level stats from an `AudioStageStream`:

```
((AudioStageStream)stream).setStatsCallback((peak, rms) -> {
 // handle statistics
});
```

When a participant stops publishing or is unsubscribed from, the on`StreamsRemoved` function is called with the streams that were removed. Host applications should use this as a signal to remove the participant's video stream from the view hierarchy.

on`StreamsRemoved` is invoked for all scenarios in which a stream might be removed, including:

- The remote participant stops publishing.
- A local device unsubscribes or changes subscription from `AUDIO_VIDEO` to `AUDIO_ONLY`.
- The remote participant leaves the stage.
- The local participant leaves the stage.

Because on`StreamsRemoved` is invoked for all scenarios, no custom business logic is required around removing participants from the UI during remote or local leave operations.

Mute and Unmute Media Streams

`LocalStageStream` objects have a `setMuted` function that controls whether the stream is muted. This function can be called on the stream before or after it is returned from the `streamsToPublishForParticipant` strategy function.

Important: If a new `LocalStageStream` object instance is returned by `streamsToPublishForParticipant` after a call to `refreshStrategy`, the mute state of the new stream object is applied to the stage. Be careful when creating new `LocalStageStream` instances to make sure the expected mute state is maintained.

Monitor Remote Participant Media Mute State

When a participant changes the mute state of their video or audio stream, the renderer on`StreamMutedChanged` function is invoked with a list of streams that have changed. Use the `getMuted` method on `StageStream` to update your UI accordingly.

```
@Override
void onStreamsMutedChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull List<StageStream> streams) {
 for (StageStream stream : streams) {
  boolean muted = stream.getMuted();
  // handle UI changes
 }
}
```

Get WebRTC Statistics

To get the latest WebRTC statistics for a publishing stream or a subscribing stream, use `requestRTCStats` on `StageStream`. When a collection is completed, you will receive statistics through the `StageStream.Listener` which can be set on `StageStream`.

```
stream.requestRTCStats();

@Override
void onRTCStats(Map<String, Map<String, String>> statsMap) {
 for (Map.Entry<String, Map<String, string>> stat : statsMap.entrySet()) {
  for(Map.Entry<String, String> member : stat.getValue().entrySet()) {
   Log.i(TAG, stat.getKey() + " has member " + member.getKey() + " with value " +
 member.getValue());
```

```
  }
 }
}
```

Get Participant Attributes

If you specify attributes in the `CreateParticipantToken` operation request, you can see the attributes in `ParticipantInfo` properties:

```
@Override
void onParticipantJoined(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 for (Map.Entry<String, String> entry : participantInfo.userInfo.entrySet()) {
  Log.i(TAG, “attribute: “ + entry.getKey() + “ = “ + entry.getValue());
 }
}
```

Embed Messages

The `embedMessage` method on ImageDevice allows you to insert metadata payloads directly into video frames during publishing. This enables frame-synchronized messaging for real-time applications. Message embedding is available only when using the SDK for real-time publishing (not low-latency publishing).

Embedded messages are not guaranteed to arrive to subscribers because they are embedded directly within video frames and transmitted over UDP, which does not guarantee packet delivery. Packet loss during transmission can result in lost messages, especially in poor network conditions. To mitigate this, the `embedMessage` method includes a `repeatCount` parameter that duplicates the message across multiple consecutive frames, increasing delivery reliability. This capability is available only for video streams.

Using embedMessage

Publishing clients can embed message payloads into their video stream using the `embedMessage` method on ImageDevice. The payload size must be greater than 0KB and less than 1KB. The number of embedded messages inserted per second must not exceed 10KB per second.

```
val surfaceSource: SurfaceSource = imageStream.device as SurfaceSource
val message = "hello world"
```

```
val messageBytes = message.toByteArray(StandardCharsets.UTF_8)

try {
    surfaceSource.embedMessage(messageBytes, 0)
} catch (e: BroadcastException) {
    Log.e("EmbedMessage", "Failed to embed message: ${e.message}")
}
```

Repeating Message Payloads

Use `repeatCount` to duplicate the message across multiple frames for improved reliability. This value must be between 0 and 30. Receiving clients must have logic to de-duplicate the message.

```
try {
    surfaceSource.embedMessage(messageBytes, 5)
    // repeatCount: 0-30, receiving clients should handle duplicates
} catch (e: BroadcastException) {
    Log.e("EmbedMessage", "Failed to embed message: ${e.message}")
}
```

Reading Embedded Messages

See "Get Supplemental Enhancement Information (SEI)" below for how to read embedded messages from incoming streams.

Get Supplemental Enhancement Information (SEI)

The Supplemental Enhancement Information (SEI) NAL unit is used to store frame-aligned metadata alongside the video. Subscribing clients can read SEI payloads from a publisher who is publishing H.264 video by inspecting the `embeddedMessages` property on the `ImageDeviceFrame` objects coming out of the publisher's `ImageDevice`. To do this, acquire a publisher's `ImageDevice`, then observe each frame via a callback provided to `setOnFrameCallback`, as shown in the following example:

```
// in a StageRenderer's onStreamsAdded function, after acquiring the new ImageStream

val imageDevice = imageStream.device as ImageDevice
imageDevice.setOnFrameCallback(object : ImageDevice.FrameCallback {
 override fun onFrame(frame: ImageDeviceFrame) {
      for (message in frame.embeddedMessages) {
```

```
          if (message is UserDataUnregisteredSeiMessage) {
              val seiMessageBytes = message.data
              val seiMessageUUID = message.uuid

              // interpret the message's data based on the UUID
          }
      }
 }
})
```

Continue Session in the Background

When the app enters the background, you may want to stop publishing or subscribe only to other remote participants' audio. To accomplish this, update your `Strategy` implementation to stop publishing, and subscribe to `AUDIO_ONLY` (or `NONE`, if applicable).

```
// Local variables before going into the background
boolean shouldPublish = true;
Stage.SubscribeType subscribeType = Stage.SubscribeType.AUDIO_VIDEO;

// Stage.Strategy implementation
@Override
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 return shouldPublish;
}

@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 return subscribeType;
}

// In our Activity, modify desired publish/subscribe when we go to background, then
 call refreshStrategy to update the stage
@Override
void onStop() {
 super.onStop();
 shouldPublish = false;
 subscribeTpye = Stage.SubscribeType.AUDIO_ONLY;
 stage.refreshStrategy();
}
```

Layered Encoding with Simulcast

Layered encoding with simulcast is an IVS real-time streaming feature that allows publishers to send multiple different quality layers of video, and subscribers to dynamically or manually configure those layers. The feature is described more in the Streaming Optimizations document.

Configuring Layered Encoding (Publisher)

As a publisher, to enable layered encoding with simulcast, add the following configuration to your `LocalStageStream` on instantiation:

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

Depending on the resolution you set on video configuration, a set number of layers will be encoded and sent as defined in the Default Layers, Qualities, and Framerates section of *Streaming Optimizations*.

Also, you can optionally configure individual layers from within the simulcast configuration:

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

List<StageVideoConfiguration.Simulcast.Layer> simulcastLayers = new ArrayList<>();
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_720);
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_180);

config.simulcast.setLayers(simulcastLayers);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

Alternately, you can create your own custom layer configurations for up to three layers. If you provide an empty array or no value, the defaults described above are used. Layers are described with the following required property setters:

- `setSize: Vec2;`
- `setMaxBitrate: integer;`
- `setMinBitrate: integer;`
- `setTargetFramerate: integer;`

Starting from the presets, you can either override individual properties or create an entirely new configuration:

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

List<StageVideoConfiguration.Simulcast.Layer> simulcastLayers = new ArrayList<>();

// Configure high quality layer with custom framerate
StageVideoConfiguration.Simulcast.Layer customHiLayer =
 StagePresets.SimulcastLocalLayer.DEFAULT_720;
customHiLayer.setTargetFramerate(15);

// Add layers to the list
simulcastLayers.add(customHiLayer);
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_180);

config.simulcast.setLayers(simulcastLayers);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

For maximum values, limits, and errors which can be triggered when configuring individual layers, see the SDK reference documentation.

Configuring Layered Encoding (Subscriber)

As a subscriber, there is nothing needed to enable layered encoding. If a publisher is sending simulcast layers, then by default the server dynamically adapts between the layers to choose the optimal quality based on the subscriber's device and network conditions.

Alternatively, to pick explicit layers that the publisher is sending, there are several options, described below.

Option 1: Initial Layer Quality Preference

Using the `subscribeConfigurationForParticipant` strategy, it is possible to choose what initial layer you want to receive as a subscriber:

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.simulcast.setInitialLayerPreference(SubscribeSimulcastConfiguration.InitialLayerPrefere

    return config;
}
```

By default, subscribers always are sent the lowest quality layer first; this slowly ramps up to the highest quality layer. This optimizes end-user bandwidth consumption and provides the best time to video, reducing initial video freezes for users on weaker networks.

These options are available for `InitialLayerPreference`:

- `LOWEST_QUALITY` — The server delivers the lowest quality layer of video first. This optimizes bandwidth consumption, as well as time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 720p video is lower quality than 1080p video.
- `HIGHEST_QUALITY` — The server delivers the highest quality layer of video first. This optimizes quality but may increase the time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 1080p video is higher quality than 720p video.

Note: For initial layer preferences (the `setInitialLayerPreference` call) to take effect, a re-subscribe is necessary as these updates do not apply to the active subscription.

Option 2: Preferred Layer for Stream

The `preferredLayerForStream` strategy method lets you select a layer after the stream has started. This strategy method receives the participant and the stream information, so you can select a layer on a participant-by-participant basis. The SDK calls this method in response to specific events, such as when stream layers change, the participant state changes, or the host application refreshes the strategy.

The strategy method returns a `RemoteStageStream.Layer` object, which can be one of the following:

- A layer object, such as one returned by `RemoteStageStream.getLayers`.
- null, which indicates that no layer should be selected and dynamic adaption is preferred.

For example, the following strategy will always have the users selecting the lowest quality layer of video available:

```
@Nullable
@Override
public RemoteStageStream.Layer preferredLayerForStream(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo, @NonNull RemoteStageStream stream) {
    return stream.getLowestQualityLayer();
}
```

To reset the layer selection and return to dynamic adaption, return null or undefined in the strategy. In this example, `appState` is a placeholder variable that represents the host application's state.

```
@Nullable
@Override
public RemoteStageStream.Layer preferredLayerForStream(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo, @NonNull RemoteStageStream stream) {
    if (appState.isAutoMode) {
        return null;
    } else {
        return appState.layerChoice;
    }
}
```

Option 3: RemoteStageStream Layer Helpers

`RemoteStageStream` has several helpers which can be used to make decisions about layer selection and display the corresponding selections to end users:

- **Layer Events** — Alongside `StageRenderer`, the `RemoteStageStream.Listener` has events which communicate layer and simulcast adaption changes:
  - `void onAdaptionChanged(boolean adaption)`

  - `void onLayersChanged(@NonNull List<Layer> layers)`
  - `void onLayerSelected(@Nullable Layer layer, @NonNull LayerSelectedReason reason)`
- **Layer Methods** — `RemoteStageStream` has several helper methods which can be used to get information about the stream and the layers being presented. These methods are available on the remote stream provided in the `preferredLayerForStream` strategy, as well as remote streams exposed via `StageRenderer.onStreamsAdded`.
  - `stream.getLayers`
  - `stream.getSelectedLayer`
  - `stream.getLowestQualityLayer`
  - `stream.getHighestQualityLayer`
  - `stream.getLayersWithConstraints`

For details, see the `RemoteStageStream` class in the SDK reference documentation. For the `LayerSelected` reason, if `UNAVAILABLE` is returned, this indicates that the requested layer could not be selected. A best-effort selection is made in its place, which typically is a lower quality layer to maintain stream stability.

Video-Configuration Limitations

The SDK does not support forcing portrait mode or landscape mode using `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. In portrait orientation, the smaller dimension is used as the width; in landscape orientation, the height. This means that the following two calls to `setSize` have the same effect on the video configuration:

```
StageVideo Configuration config = new StageVideo Configuration();

config.setSize(BroadcastConfiguration.Vec2(720f, 1280f);
config.setSize(BroadcastConfiguration.Vec2(1280f, 720f);
```

Handling Network Issues

When the local device's network connection is lost, the SDK internally tries to reconnect without any user action. In some cases, the SDK is not successful and user action is needed. There are two main errors related to losing the network connection:

- Error code 1400, message: "PeerConnection is lost due to unknown network error"

- Error code 1300, message: "Retry attempts are exhausted"

If the first error is received but the second is not, the SDK is still connected to the stage and will try to reestablish its connections automatically. As a safeguard, you can call `refreshStrategy` without any changes to the strategy method's return values, to trigger a manual reconnect attempt.

If the second error is received, the SDK's reconnect attempts have failed and the local device is no longer connected to the stage. In this case, try to rejoin the stage by calling `join` after your network connection has been reestablished.

In general, encountering errors after joining a stage successfully indicates that the SDK was unsuccessful in reestablishing a connection. Create a new `Stage` object and try to join when network conditions improve.

Using Bluetooth Microphones

To publish using Bluetooth microphone devices, you must start a Bluetooth SCO connection:

```
Bluetooth.startBluetoothSco(context);
// Now bluetooth microphones can be used
…
// Must also stop bluetooth SCO
Bluetooth.stopBluetoothSco(context);
```

Error Handling in the IVS Android Broadcast SDK | Real-Time Streaming

This section is an overview of error conditions, how the IVS real-time streaming Android broadcast SDK reports them to the application, and what an application should do when those errors are encountered.

Fatal vs. Non-Fatal Errors

The error object has an "is fatal" boolean field of `BroadcastException`.

In general, fatal errors are related to connection to the Stages server (either a connection cannot be established or is lost and cannot be recovered). The application should re-create the stage and re-join, possibly with a new token or when the device's connectivity recovers.

Non-fatal errors generally are related to the publish/subscribe state and are handled by the SDK, which retries the publish/subscribe operation.

You can check this property:

```
try {
  stage.join(...)
} catch (e: BroadcastException) {
  If (e.isFatal) {
    // the error is fatal
```

Join Errors

Malformed Token

This happens when the stage token is malformed.

The SDK throws a Java exception from a call to `stage.join`, with error code = 1000 and fatal = true.

Action: Create a valid token and retry joining.

Expired Token

This happens when the stage token is expired.

The SDK throws a Java exception from a call to `stage.join`, with error code = 1001 and fatal = true.

Action: Create a new token and retry joining.

Invalid or Revoked Token

This happens when the stage token is not malformed but is rejected by the Stages server. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls `onConnectionStateChanged` with an exception, with error code = 1026 and fatal = true.

Action: Create a valid token and retry joining.

Network Errors for Initial Join

This happens when the SDK cannot contact the Stages server to establish a connection. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls onConnectionStateChanged with an exception, with error code = 1300 and fatal = true.

Action: Wait for the device's connectivity to recover and retry joining.

Network Errors when Already Joined

If the device's network connection goes down, the SDK may lose its connection to Stage servers. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls onConnectionStateChanged with an exception, with error code = 1300 and fatal = true.

Action: Wait for the device's connectivity to recover and retry joining.

Publish/Subscribe Errors

Initial

There are several errors:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)
- MultihostSessionNoIceCandidates (1022)
- MultihostSessionStageAtCapacity (1024)
- SignallingSessionCannotRead (1201)
- SignallingSessionCannotSend (1202)
- SignallingSessionBadResponse (1203)

These are reported asynchronously through the application-supplied stage renderer.

The SDK retries the operation for a limited number of times. During retries, the publish/subscribe state is ATTEMPTING_PUBLISH / ATTEMPTING_SUBSCRIBE. If the retry attempts succeed, the state changes to PUBLISHED / SUBSCRIBED.

The SDK calls onError with the relevant error code and fatal = false.

Action: No action is needed, as the SDK retries automatically. Optionally, the application can refresh the strategy to force more retries.

Already Established, Then Fail

A publish or subscribe can fail after it is established, most likely due to a network error. The error code for a "peer connection lost due to network error" is 1400.

This is reported asynchronously through the application-supplied stage renderer.

The SDK retries the publish/subscribe operation. During retries, the publish/subscribe state is ATTEMPTING_PUBLISH / ATTEMPTING_SUBSCRIBE. If the retry attempts succeed, the state changes to PUBLISHED / SUBSCRIBED.

The SDK calls onError with the error code = 1400 and fatal = false.

Action: No action is needed, as the SDK retries automatically. Optionally, the application can refresh the strategy to force more retries. In the event of total connectivity loss, it's likely that the connection to Stages will fail too.

IVS Broadcast SDK: iOS Guide | Real-Time Streaming

The IVS real-time streaming iOS broadcast SDK enables participants to send and receive video on iOS.

The AmazonIVSBroadcast module implements the interface described in this document. The following operations are supported:

- Join a stage
- Publish media to other participants in the stage
- Subscribe to media from other participants in the stage
- Manage and monitor video and audio published to the stage
- Get WebRTC statistics for each peer connection
- All operations from the IVS low-latency streaming iOS broadcast SDK

Latest version of iOS broadcast SDK: 1.46.0 (Release Notes)

Reference documentation: For information on the most important methods available in the Amazon IVS iOS broadcast SDK, see the reference documentation at https://aws.github.io/amazon-ivs-broadcast-docs/1.46.0/ios/.

Sample code: See the iOS sample repository on GitHub: https://github.com/aws-samples/amazon-ivs-real-time-streaming-ios-samples.

Platform requirements: iOS 14+

Getting Started with the IVS iOS Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in getting started with the IVS real-time streaming iOS broadcast SDK.

Install the Library

We recommend that you integrate broadcast SDK via Swift Package Manager. (Alternatively, you can manually add the framework to your project.)

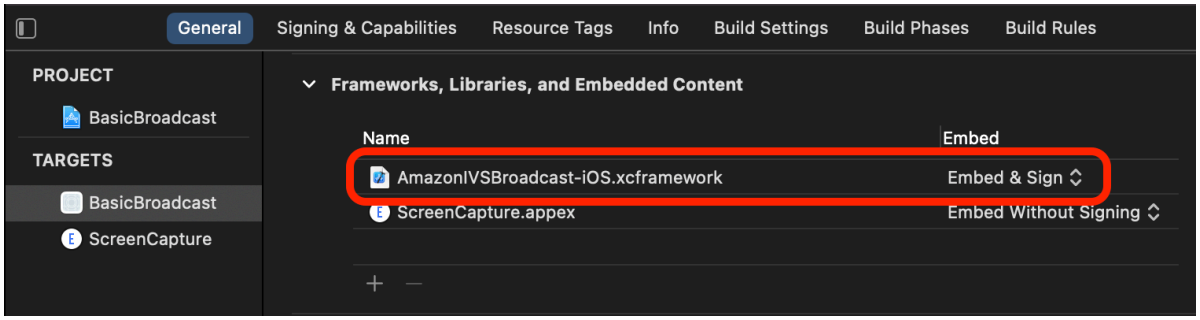
Recommended: Integrate the Broadcast SDK (Swift Package Manager)

1. Download the Package.swift file from https://broadcast.live-video.net/1.46.0/Package.swift.
2. In your project, create a new directory named AmazonIVSBroadcast and add it to version control.
3. Place the downloaded Package.swift file in the new directory.
4. In Xcode, go to **File > Add Package Dependencies** and select **Add Local...**
5. Navigate to and select the AmazonIVSBroadcast directory that you created, and select **Add Package**.
6. When prompted to **Choose Package Products for AmazonIVSBroadcast**, select **AmazonIVSBroadcastStages** as your **Package Product** by setting your application target in the **Add to Target** section.
7. Select **Add Package**.

Important: The IVS real-time streaming broadcast SDK includes all features of the IVS low-latency streaming broadcast SDK. It is not possible to integrate both SDKs in the same project.

Alternate Approach: Install the Framework Manually

1. Download the latest version from https://broadcast.live-video.net/1.46.0/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extract the contents of the archive. `AmazonIVSBroadcast.xcframework` contains the SDK for both device and simulator.
3. Embed `AmazonIVSBroadcast.xcframework` by dragging it into the **Frameworks, Libraries, and Embedded Content** section of the **General** tab for your application target.



Request Permissions

Your app must request permission to access the user's camera and mic. (This is not specific to Amazon IVS; it is required for any application that needs access to cameras and microphones.)

Here, we check whether the user has already granted permissions and, if not, we ask for them:

```
switch AVCaptureDevice.authorizationStatus(for: .video) {
case .authorized: // permission already granted.
case .notDetermined:
   AVCaptureDevice.requestAccess(for: .video) { granted in
       // permission granted based on granted bool.
   }
case .denied, .restricted: // permission denied.
@unknown default: // permissions unknown.
}
```

You need to do this for both `.video` and `.audio` media types, if you want access to cameras and microphones, respectively.

You also need to add entries for `NSCameraUsageDescription` and `NSMicrophoneUsageDescription` to your `Info.plist`. Otherwise, your app will crash when trying to request permissions.

Disable the Application Idle Timer

This is optional but recommended. It prevents your device from going to sleep while using the broadcast SDK, which would interrupt the broadcast.

```
override func viewDidAppear(_ animated: Bool) {
   super.viewDidAppear(animated)
   UIApplication.shared.isIdleTimerDisabled = true
}
override func viewDidDisappear(_ animated: Bool) {
   super.viewDidDisappear(animated)
   UIApplication.shared.isIdleTimerDisabled = false
}
```

Publishing & Subscribing with the IVS iOS Broadcast SDK | Real-Time Streaming

This document takes you through the steps involved in publishing and subscribing to a stage using the IVS real-time streaming iOS broadcast SDK.

Concepts

Three core concepts underlie real-time functionality: stage, strategy, and renderer. The design goal is minimizing the amount of client-side logic necessary to build a working product.

Stage

The `IVSStage` class is the main point of interaction between the host application and the SDK. The class represents the stage itself and is used to join and leave the stage. Creating or joining a stage requires a valid, unexpired token string from the control plane (represented as `token`). Joining and leaving a stage are simple.

```
let stage = try IVSStage(token: token, strategy: self)

try stage.join()

stage.leave()
```

The `IVSStage` class also is where the `IVSStageRenderer` and `IVSErrorDelegate` can be attached:

```
let stage = try IVSStage(token: token, strategy: self)
stage.errorDelegate = self
stage.addRenderer(self) // multiple renderers can be added
```

Strategy

The `IVSStageStrategy` protocol provides a way for the host application to communicate the desired state of the stage to the SDK. Three functions need to be implemented: `shouldSubscribeToParticipant`, `shouldPublishParticipant`, and `streamsToPublishForParticipant`. All are discussed below.

Subscribing to Participants

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType
```

When a remote participant joins a stage, the SDK queries the host application about the desired subscription state for that participant. The options are `.none`, `.audioOnly`, and `.audioVideo`. When returning a value for this function, the host application does not need to worry about the publish state, current subscription state, or stage connection state. If `.audioVideo` is returned, the SDK waits until the remote participant is publishing before subscribing, and it updates the host application through the renderer throughout the process.

Here is a sample implementation:

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    return .audioVideo
}
```

This is the complete implementation of this function for a host application that always wants all participants to see each other; e.g., a video-chat application.

More advanced implementations also are possible. Use the `attributes` property on `IVSParticipantInfo` to selectively subscribe to participants based on server-provided attributes:

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    switch participant.attributes["role"] {
```

```
    case "moderator": return .none
    case "guest": return .audioVideo
    default: return .none
    }
}
```

This can be used to create a stage where moderators can monitor all guests without being seen or heard themselves. The host application could use additional business logic to let moderators see each other but remain invisible to guests.

Configuration for Subscribing to Participants

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration
```

If a remote participant is being subscribed to (see Subscribing to Participants), the SDK queries the host application about a custom subscribe configuration for that participant. This configuration is optional and allows the host application to control certain aspects of subscriber behavior. For information on what can be configured, see SubscribeConfiguration in the SDK reference documentation.

Here is a sample implementation:

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()

    try! config.jitterBuffer.setMinDelay(.medium())

    return config
}
```

This implementation updates the jitter-buffer minimum delay for all subscribed participants to a preset of `MEDIUM`.

As with `shouldSubscribeToParticipant`, more advanced implementations are possible. The given `ParticipantInfo` can be used to selectively update the subscribe configuration for specific participants.

We recommend using the default behaviors. Specify custom configuration only if there is a particular behavior you want to change.

Publishing

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool
```

Once connected to the stage, the SDK queries the host application to see if a particular participant should publish. This is invoked only on local participants that have permission to publish based on the provided token.

Here is a sample implementation:

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return true
}
```

This is for a standard video chat application where users always want to publish. They can mute and unmute their audio and video, to instantly be hidden or seen/heard. (They also can use publish/unpublish, but that is much slower. Mute/unmute is preferable for use cases where changing visibility often is desirable.)

Choosing Streams to Publish

```
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream]
```

When publishing, this is used to determine what audio and video streams should be published. This is covered in more detail later in Publish a Media Stream.

Updating the Strategy

The strategy is intended to be dynamic: the values returned from any of the above functions can be changed at any time. For example, if the host application does not want to publish until the end user taps a button, you could return a variable from `shouldPublishParticipant` (something like `hasUserTappedPublishButton`). When that variable changes based on an interaction by the end user, call `stage.refreshStrategy()` to signal to the SDK that it should query the strategy for the latest values, applying only things that have changed. If the SDK observes that the `shouldPublishParticipant` value has changed, it will start the publish process. If the SDK

queries and all functions return the same value as before, the `refreshStrategy` call will not make any modifications to the stage.

If the return value of `shouldSubscribeToParticipant` changes from `.audioVideo` to `.audioOnly`, the video stream will be removed for all participants with changed returned values, if a video stream existed previously.

Generally, the stage uses the strategy to most efficiently apply the difference between the previous and current strategies, without the host application needing to worry about all the state required to manage it properly. Because of this, think of calling `stage.refreshStrategy()` as a cheap operation, because it does nothing unless the strategy changes.

Renderer

The `IVSStageRenderer` protocol communicates the state of the stage to the host application. Updates to the host application's UI usually can be powered entirely by the events provided by the renderer. The renderer provides the following functions:

```
func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo)

func stage(_ stage: IVSStage, participantDidLeave participant: IVSParticipantInfo)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange publishState:
 IVSParticipantPublishState)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
 subscribeState: IVSParticipantSubscribeState)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams:
 [IVSStageStream])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didRemove streams:
 [IVSStageStream])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChangeMutedStreams
 streams: [IVSStageStream])

func stage(_ stage: IVSStage, didChange connectionState: IVSStageConnectionState,
 withError error: Error?)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didChangeStreamAdaption adaption: Bool)
```

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didChange layers: [IVSRemoteStageStreamLayer])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didSelect layer: IVSRemoteStageStreamLayer?, reason:
 IVSRemoteStageStream.LayerSelectedReason)
```

It is not expected that the information provided by the renderer impacts the return values of the strategy. For example, the return value of `shouldSubscribeToParticipant` is not expected to change when `participant:didChangePublishState` is called. If the host application wants to subscribe to a particular participant, it should return the desired subscription type regardless of that participant's publish state. The SDK is responsible for ensuring that the desired state of the strategy is acted on at the correct time based on the state of the stage.

Note that only publishing participants trigger `participantDidJoin`, and whenever a participant stops publishing or leaves the stage session, `participantDidLeave` is triggered.

Publish a Media Stream

Local devices such as built-in microphones and cameras are discovered via `IVSDeviceDiscovery`. Here is an example of selecting the front-facing camera and default microphone, then returning them as `IVSLocalStageStreams` to be published by the SDK:

```
let devices = IVSDeviceDiscovery().listLocalDevices()

// Find the camera virtual device, choose the front source, and create a stream
let camera = devices.compactMap({ $0 as? IVSCamera }).first!
let frontSource = camera.listAvailableInputSources().first(where: { $0.position
 == .front })!
camera.setPreferredInputSource(frontSource)
let cameraStream = IVSLocalStageStream(device: camera)

// Find the microphone virtual device and create a stream
let microphone = devices.compactMap({ $0 as? IVSMicrophone }).first!
let microphoneStream = IVSLocalStageStream(device: microphone)

// Configure the audio manager to use the videoChat preset, which is optimized for bi-
directional communication, including echo cancellation.
IVSStageAudioManager.sharedInstance().setPreset(.videoChat)

// This is a function on IVSStageStrategy
```

```
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream] {
    return [cameraStream, microphoneStream]
}
```

Display and Remove Participants

After subscribing is completed, you will receive an array of `IVSStageStream` objects through the renderer's `didAddStreams` function. To preview or receive audio level stats about this participant, you can access the underlying `IVSDevice` object from the stream:

```
if let imageDevice = stream.device as? IVSImageDevice {
    let preview = imageDevice.previewView()
    /* attach this UIView subclass to your view */
} else if let audioDevice = stream.device as? IVSAudioDevice {
    audioDevice.setStatsCallback( { stats in
        /* process stats.peak and stats.rms */
    })
}
```

When a participant stops publishing or is unsubscribed from, the `didRemoveStreams` function is called with the streams that were removed. Host applications should use this as a signal to remove the participant's video stream from the view hierarchy.

`didRemoveStreams` is invoked for all scenarios in which a stream might be removed, including:

- The remote participant stops publishing.
- A local device unsubscribes or changes subscription from `.audioVideo` to `.audioOnly`.
- The remote participant leaves the stage.
- The local participant leaves the stage.

Because `didRemoveStreams` is invoked for all scenarios, no custom business logic is required around removing participants from the UI during remote or local leave operations.

Mute and Unmute Media Streams

`IVSLocalStageStream` objects have a `setMuted` function that controls whether the stream is muted. This function can be called on the stream before or after it is returned from the `streamsToPublishForParticipant` strategy function.

Important: If a new `IVSLocalStageStream` object instance is returned by `streamsToPublishForParticipant` after a call to `refreshStrategy`, the mute state of the new stream object is applied to the stage. Be careful when creating new `IVSLocalStageStream` instances to make sure the expected mute state is maintained.

Monitor Remote Participant Media Mute State

When a participant changes the mute state of its video or audio stream, the renderer `didChangeMutedStreams` function is invoked with an array of streams that have changed. Use the `isMuted` property on `IVSStageStream` to update your UI accordingly:

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChangeMutedStreams
 streams: [IVSStageStream]) {
    streams.forEach { stream in
        /* stream.isMuted */
    }
}
```

Create a Stage Configuration

To customize the values of a stage's video configuration, use `IVSLocalStageStreamVideoConfiguration`:

```
let config = IVSLocalStageStreamVideoConfiguration()
try config.setMaxBitrate(900_000)
try config.setMinBitrate(100_000)
try config.setTargetFramerate(30)
try config.setSize(CGSize(width: 360, height: 640))
config.degradationPreference = .balanced
```

Get WebRTC Statistics

To get the latest WebRTC statistics for a publishing stream or a subscribing stream, use `requestRTCStats` on `IVSStageStream`. When a collection is completed, you will receive statistics through the `IVSStageStreamDelegate` which can be set on `IVSStageStream`. To continually collect WebRTC statistics, call this function on a `Timer`.

```
func stream(_ stream: IVSStageStream, didGenerateRTCStats stats: [String : [String :
 String]]) {
    for stat in stats {
```

```
      for member in stat.value {
         print("stat \(stat.key) has member \(member.key) with value \(member.value)")
      }
   }
}
```

Get Participant Attributes

If you specify attributes in the `CreateParticipantToken` operation request, you can see the attributes in `IVSParticipantInfo` properties:

```
func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo) {
    print("ID: \(participant.participantId)")
    for attribute in participant.attributes {
        print("attribute: \(attribute.key)=\(attribute.value)")
    }
}
```

Embed Messages

The `embedMessage` method on IVSImageDevice allows you to insert metadata payloads directly into video frames during publishing. This enables frame-synchronized messaging for real-time applications. Message embedding is available only when using the SDK for real-time publishing (not low-latency publishing).

Embedded messages are not guaranteed to arrive to subscribers because they are embedded directly within video frames and transmitted over UDP, which does not guarantee packet delivery. Packet loss during transmission can result in lost messages, especially in poor network conditions. To mitigate this, the `embedMessage` method includes a `repeatCount` parameter that duplicates the message across multiple consecutive frames, increasing delivery reliability. This capability is available only for video streams.

Using embedMessage

Publishing clients can embed message payloads into their video stream using the `embedMessage` method on IVSImageDevice. The payload size must be greater than 0KB and less than 1KB. The number of embedded messages inserted per second must not exceed 10KB per second.

```
let imageDevice: IVSImageDevice = imageStream.device as! IVSImageDevice
let messageData = Data("hello world".utf8)
```

```
do {
    try imageDevice.embedMessage(messageData, withRepeatCount: 0)
} catch {
    print("Failed to embed message: \(error)")
}
```

Repeating Message Payloads

Use `repeatCount` to duplicate the message across multiple frames for improved reliability. This value must be between 0 and 30. Receiving clients must have logic to de-duplicate the message.

```
try imageDevice.embedMessage(messageData, withRepeatCount: 5)

// repeatCount: 0-30, receiving clients should handle duplicates
```

Reading Embedded Messages

See "Get Supplemental Enhancement Information (SEI)" below for how to read embedded messages from incoming streams.

Get Supplemental Enhancement Information (SEI)

The Supplemental Enhancement Information (SEI) NAL unit is used to store frame-aligned metadata alongside the video. Subscribing clients can read SEI payloads from a publisher who is publishing H.264 video by inspecting the `embeddedMessages` property on the `IVSImageDeviceFrame` objects coming out of the publisher's `IVSImageDevice`. To do this, acquire a publisher's `IVSImageDevice`, then observe each frame via a callback provided to `setOnFrameCallback`, as shown in the following example:

```
// in an IVSStageRenderer's stage:participant:didAddStreams: function, after acquiring
 the new IVSImageStream

let imageDevice: IVSImageDevice? = imageStream.device as? IVSImageDevice
imageDevice?.setOnFrameCallback { frame in
 for message in frame.embeddedMessages {
      if let seiMessage = message as? IVSUserDataUnregisteredSEIMessage {
          let seiMessageData = seiMessage.data
          let seiMessageUUID = seiMessage.UUID

          // interpret the message's data based on the UUID
```

```
        }
    }
}
```

Continue Session in the Background

When the app enters the background, you can continue to be in the stage while hearing remote audio, though it is not possible to continue to send your own image and audio. You will need to update your `IVSStrategy` implementation to stop publishing and subscribe to `.audioOnly` (or `.none`, if applicable):

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return false
}
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    return .audioOnly
}
```

Then make a call to `stage.refreshStrategy()`.

Layered Encoding with Simulcast

Layered encoding with simulcast is an IVS real-time streaming feature that allows publishers to send multiple different quality layers of video, and subscribers to dynamically or manually configure those layers. The feature is described more in the Streaming Optimizations document.

Configuring Layered Encoding (Publisher)

As a publisher, to enable layered encoding with simulcast, add the following configuration to your `IVSLocalStageStream` on instantiation:

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

Depending on the resolution you set on video configuration, a set number of layers will be encoded and sent as defined in the Default Layers, Qualities, and Framerates section of *Streaming Optimizations*.

Also, you can optionally configure individual layers from within the simulcast configuration:

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let layers = [
    IVSStagePresets.simulcastLocalLayer().default720(),
    IVSStagePresets.simulcastLocalLayer().default180()
]

try config.simulcast.setLayers(layers)

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

Alternately, you can create your own custom layer configurations for up to three layers. If you provide an empty array or no value, the defaults described above are used. Layers are described with the following required property setters:

- `setSize: CGSize;`
- `setMaxBitrate: integer;`
- `setMinBitrate: integer;`
- `setTargetFramerate: float;`

Starting from the presets, you can either override individual properties or create an entirely new configuration:

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let customHiLayer = IVSStagePresets.simulcastLocalLayer().default720()
try customHiLayer.setTargetFramerate(15)
```

```
let layers = [
    customHiLayer,
    IVSStagePresets.simulcastLocalLayer().default180()
]

try config.simulcast.setLayers(layers)

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

For maximum values, limits, and errors which can be triggered when configuring individual layers, see the SDK reference documentation.

Configuring Layered Encoding (Subscriber)

As a subscriber, there is nothing needed to enable layered encoding. If a publisher is sending simulcast layers, then by default the server dynamically adapts between the layers to choose the optimal quality based on the subscriber's device and network conditions.

Alternatively, to pick explicit layers that the publisher is sending, there are several options, described below.

Option 1: Initial Layer Quality Preference

Using the `subscribeConfigurationForParticipant` strategy, it is possible to choose what initial layer you want to receive as a subscriber:

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()

    config.simulcast.initialLayerPreference = .lowestQuality

    return config
}
```

By default, subscribers always are sent the lowest quality layer first; this slowly ramps up to the highest quality layer. This optimizes end-user bandwidth consumption and provides the best time to video, reducing initial video freezes for users on weaker networks.

These options are available for `InitialLayerPreference`:

- `lowestQuality` — The server delivers the lowest quality layer of video first. This optimizes bandwidth consumption, as well as time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 720p video is lower quality than 1080p video.
- `highestQuality` — The server delivers the highest quality layer of video first. This optimizes quality but may increase the time to media. Quality is defined as the combination of size, bitrate, and framerate of the video. For example, 1080p video is higher quality than 720p video.

Note: For initial layer preferences (the `initialLayerPreference` call) to take effect, a re-subscribe is necessary as these updates do not apply to the active subscription.

Option 2: Preferred Layer for Stream

The `preferredLayerForStream` strategy method lets you select a layer after the stream has started. This strategy method receives the participant and the stream information, so you can select a layer on a participant-by-participant basis. The SDK calls this method in response to specific events, such as when stream layers change, the participant state changes, or the host application refreshes the strategy.

The strategy method returns an `IVSRemoteStageStreamLayer` object, which can be one of the following:

- A layer object, such as one returned by `IVSRemoteStageStream.layers`.
- null, which indicates that no layer should be selected and dynamic adaption is preferred.

For example, the following strategy will always have the users selecting the lowest quality layer of video available:

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, preferredLayerFor
 stream: IVSRemoteStageStream) -> IVSRemoteStageStreamLayer? {
    return stream.lowestQualityLayer
}
```

To reset the layer selection and return to dynamic adaption, return null or undefined in the strategy. In this example, `appState` is a placeholder variable that represents the host application's state.

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, preferredLayerFor
 stream: IVSRemoteStageStream) -> IVSRemoteStageStreamLayer? {
```

```
    If appState.isAutoMode {
        return nil
    } else {
        return appState.layerChoice
    }
}
```

Option 3: RemoteStageStream Layer Helpers

`IVSRemoteStageStream` has several helpers which can be used to make decisions about layer selection and display the corresponding selections to end users:

- **Layer Events** — Alongside `IVSStageRenderer`, the `IVSRemoteStageStreamDelegate` has events which communicate layer and simulcast adaption changes:
  - `func stream(_ stream: IVSRemoteStageStream, didChangeAdaption adaption: Bool)`
  - `func stream(_ stream: IVSRemoteStageStream, didChange layers: [IVSRemoteStageStreamLayer])`
  - `func stream(_ stream: IVSRemoteStageStream, didSelect layer: IVSRemoteStageStreamLayer?, reason: IVSRemoteStageStream.LayerSelectedReason)`
- **Layer Methods** — `IVSRemoteStageStream` has several helper methods which can be used to get information about the stream and the layers being presented. These methods are available on the remote stream provided in the `preferredLayerForStream` strategy, as well as remote streams exposed via `func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams: [IVSStageStream])`.
  - `stream.layers`
  - `stream.selectedLayer`
  - `stream.lowestQualityLayer`
  - `stream.highestQualityLayer`
  - `stream.layers(with: IVSRemoteStageStreamLayerConstraints)`

For details, see the `IVSRemoteStageStream` class in the SDK reference documentation. For the `LayerSelected` reason, if `UNAVAILABLE` is returned, this indicates that the requested layer could not be selected. A best-effort selection is made in its place, which typically is a lower quality layer to maintain stream stability.

Broadcast the Stage to an IVS Channel

To broadcast a stage, create a separate `IVSBroadcastSession` and then follow the usual instructions for broadcasting with the SDK, described above. The `device` property on `IVSStageStream` will be either an `IVSImageDevice` or `IVSAudioDevice` as shown in the snippet above; these can be connected to the `IVSBroadcastSession.mixer` to broadcast the entire stage in a customizable layout.

Optionally, you can composite a stage and broadcast it to an IVS low-latency channel, to reach a larger audience. See Enabling Multiple Hosts on an Amazon IVS Stream in the IVS Low-Latency Streaming User Guide.

How iOS Chooses Camera Resolution and Frame Rate

The camera managed by the broadcast SDK optimizes its resolution and frame rate (frames-per-second, or FPS) to minimize heat production and energy consumption. This section explains how the resolution and frame rate are selected to help host applications optimize for their use cases.

When creating an `IVSLocalStageStream` with an `IVSCamera`, the camera is optimized for a frame rate of `IVSLocalStageStreamVideoConfiguration.targetFramerate` and a resolution of `IVSLocalStageStreamVideoConfiguration.size`. Calling `IVSLocalStageStream.setConfiguration` updates the camera with newer values.

Camera Preview

If you create a preview of an `IVSCamera` without attaching it to a `IVSBroadcastSession` or `IVSStage`, it defaults to a resolution of 1080p and a frame rate of 60 fps.

Broadcasting a Stage

When using an `IVSBroadcastSession` to broadcast an `IVSStage`, the SDK tries to optimize the camera with a resolution and frame rate that meet the criteria of both sessions.

For example, if the broadcast configuration is set to have a frame rate of 15 FPS and a resolution of 1080p, while the Stage has a frame rate of 30 FPS and a resolution of 720p, the SDK will select a camera configuration with a frame rate of 30 FPS and a resolution of 1080p. The `IVSBroadcastSession` will drop every other frame from the camera, and the `IVSStage` will scale the 1080p image down to 720p.

If a host application plans on using both `IVSBroadcastSession` and `IVSStage` together, with a camera, we recommend that the `targetFramerate` and `size` properties of the respective configurations match. A mismatch could cause the camera to reconfigure itself while capturing video, which will cause a brief delay in video-sample delivery.

If having identical values does not meet the host application's use case, creating the higher quality camera first will prevent the camera from reconfiguring itself when the lower quality session is added. For example, if you broadcast at 1080p and 30 FPS and then later join a Stage set to 720p and 30 FPS, the camera will not reconfigure itself and video will continue uninterrupted. This is because 720p is less than or equal to 1080p and 30 FPS is less than or equal to 30 FPS.

Arbitrary Frame Rates, Resolutions, and Aspect Ratios

Most camera hardware can exactly match common formats, such as 720p at 30 FPS or 1080p at 60 FPS. However, it is not possible to exactly match all formats. The broadcast SDK chooses the camera configuration based on the following rules (in priority order):

1. The width and height of the resolution are greater than or equal to the desired resolution, but within this constraint, width and height are as small as possible.
2. The frame rate is greater than or equal to the desired frame rate, but within this constraint, frame rate is as low as possible.
3. The aspect ratio matches the desired aspect ratio.
4. If there are multiple matching formats, the format with the greatest field of view is used.

Here are two examples:

- The host application is trying to broadcast in 4k at 120 FPS. The selected camera supports only 4k at 60 FPS or 1080p at 120 FPS. The selected format will be 4k at 60 FPS, because the resolution rule is higher priority than the frame-rate rule.
- An irregular resolution is requested, 1910x1070. The camera will use 1920x1080. *Be careful: choosing a resolution like 1921x1080 will cause the camera to scale up to the next available resolution (such as 2592x1944), which incurs a CPU and memory-bandwidth penalty*.

What about Android?

Android does not adjust its resolution or frame rate on the fly like iOS does, so this does not impact the Android broadcast SDK.

Error Handling in the IVS iOS Broadcast SDK | Real-Time Streaming

This section is an overview of error conditions, how the IVS real-time streaming iOS broadcast SDK reports them to the application, and what an application should do when those errors are encountered.

Fatal vs. Non-Fatal Errors

The error object has an "is fatal" boolean. This is a dictionary entry under `IVSBroadcastErrorIsFatalKey` which contains a boolean.

In general, fatal errors are related to connection to the Stages server (either a connection cannot be established or is lost and cannot be recovered). The application should re-create the stage and re-join, possibly with a new token or when the device's connectivity recovers.

Non-fatal errors generally are related to the publish/subscribe state and are handled by the SDK, which retries the publish/subscribe operation.

You can check this property:

```
let nsError = error as NSError
if nsError.userInfo[IVSBroadcastErrorIsFatalKey] as? Bool == true {
  // the error is fatal
}
```

Join Errors

Malformed Token

This happens when the stage token is malformed.

The SDK throws a Swift exception with error code = 1000 and IVSBroadcastErrorIsFatalKey = YES.

Action: Create a valid token and retry joining.

Expired Token

This happens when the stage token is expired.

The SDK throws a Swift exception with error code = 1001 and IVSBroadcastErrorIsFatalKey = YES.

Action: Create a new token and retry joining.

Invalid or Revoked Token

This happens when the stage token is not malformed but is rejected by the Stages server. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls `stage(didChange connectionState, withError error)` with error code = 1026 and IVSBroadcastErrorIsFatalKey = YES.

Action: Create a valid token and retry joining.

Network Errors for Initial Join

This happens when the SDK cannot contact the Stages server to establish a connection. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls `stage(didChange connectionState, withError error)` with error code = 1300 and IVSBroadcastErrorIsFatalKey = YES.

Action: Wait for the device's connectivity to recover and retry joining.

Network Errors when Already Joined

If the device's network connection goes down, the SDK may lose its connection to Stage servers. This is reported asynchronously through the application-supplied stage renderer.

The SDK calls `stage(didChange connectionState, withError error)` with error code = 1300 and IVSBroadcastErrorIsFatalKey value = YES.

Action: Wait for the device's connectivity to recover and retry joining.

Publish/Subscribe Errors

Initial

There are several errors:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)
- MultihostSessionNoIceCandidates (1022)
- MultihostSessionStageAtCapacity (1024)
- SignallingSessionCannotRead (1201)
- SignallingSessionCannotSend (1202)

- SignallingSessionBadResponse (1203)

These are reported asynchronously through the application-supplied stage renderer.

The SDK retries the operation for a limited number of times. During retries, the publish/subscribe state is `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. If the retry attempts succeed, the state changes to `PUBLISHED` / `SUBSCRIBED`.

The SDK calls `IVSErrorDelegate:didEmitError` with the relevant error code and `IVSBroadcastErrorIsFatalKey == NO`.

Action: No action is needed, as the SDK retries automatically. Optionally, the application can refresh the strategy to force more retries.

Already Established, Then Fail

A publish or subscribe can fail after it is established, most likely due to a network error. The error code for a "peer connection lost due to network error" is 1400.

This is reported asynchronously through the application-supplied stage renderer.

The SDK retries the publish/subscribe operation. During retries, the publish/subscribe state is `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. If the retry attempts succeed, the state changes to `PUBLISHED` / `SUBSCRIBED`.

The SDK calls `didEmitError` with error code = 1400 and IVSBroadcastErrorIsFatalKey = NO.

Action: No action is needed, as the SDK retries automatically. Optionally, the application can refresh the strategy to force more retries. In the event of total connectivity loss, it's likely that the connection to Stages will fail too.

IVS Broadcast SDK: Mixed Devices

Mixed devices are audio and video devices that take multiple input sources and generate a single output. Mixing devices is a powerful feature that lets you define and manage multiple on-screen (video) elements and audio tracks. You can combine video and audio from multiple sources such as cameras, microphones, screen captures, and audio and video generated by your app. You can use transitions to move these sources around the video that you stream to IVS, and add to and remove sources mid-stream.

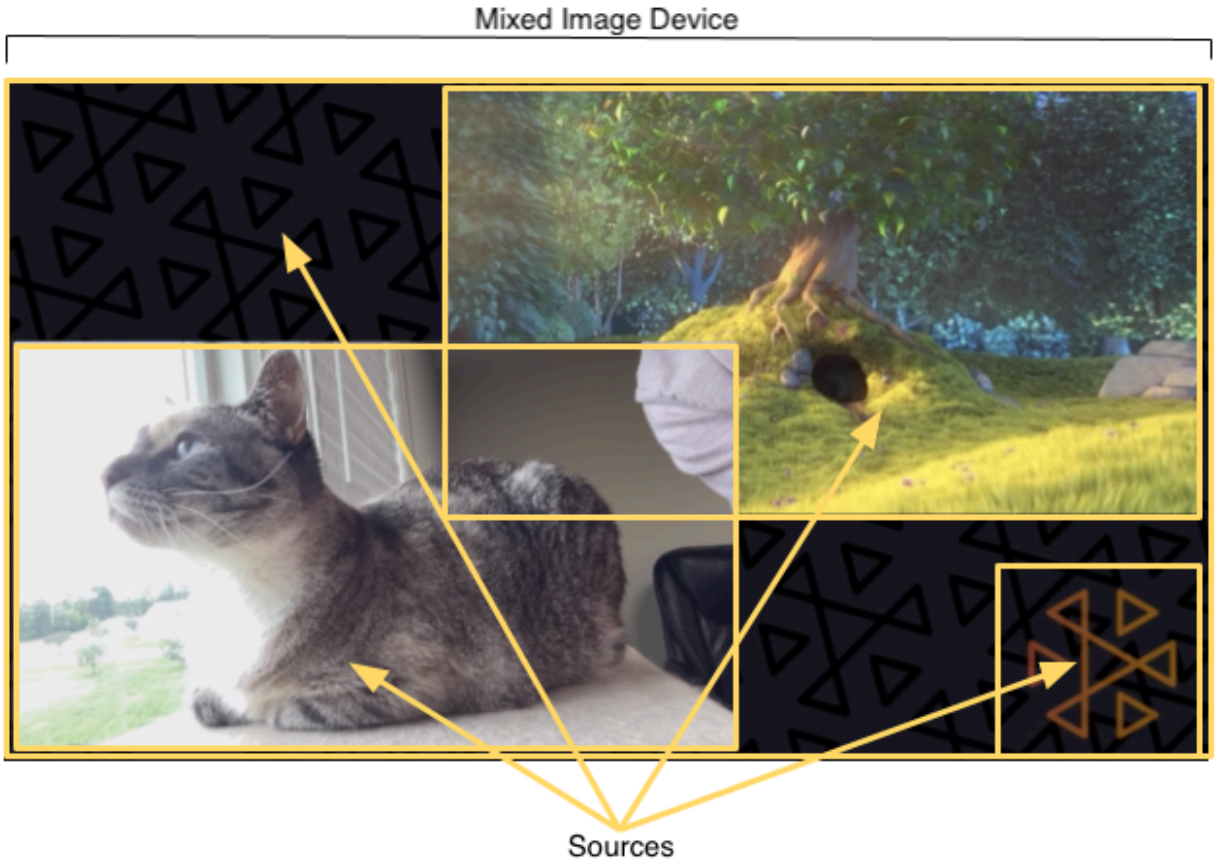
Mixed devices come in image and audio flavors. To create a mixed image device, call:

`DeviceDiscovery.createMixedImageDevice()` on Android

`IVSDeviceDiscovery.createMixedImageDevice()` on iOS

The returned device can be attached to a `BroadcastSession` (low-latency streaming) or `Stage` (real-time streaming), like any other device.

Terminology



Term	Description
Device	A hardware or software component that produces audio or image input. Examples of devices are microphones, cameras, Bluetooth headsets, and virtual devices such as screen captures or custom-image inputs.
Mixed Device	A `Device` that can be attached to a `BroadcastSession` like any other `Device`, but with additional APIs that allow `Source` objects to be added. Mixed devices have internal mixers that composite audio or images, producing a single output audio and image stream.

Term	Description
	Mixed devices come in either audio or image flavors.
Mixed device configuration	A configuration object for the mixed device. For mixed image devices, this configures properties like dimensions and framerate. For mixed audio devices, this configures the channel count.
Source	A container that defines a visual element's position on screen and an audio track's properties in the audio mix. A mixed device can be configured with zero or more sources. Sources are given a configuration that affects how the source's media are used. The image above shows four image sources: • Bottom left with camera input • Top right with movie input • Bottom right with the Amazon IVS logo • A full-screen background image
Source Configuration	A configuration object for the source going into a mixed device. The full configuration objects are described below..
Transition	To move a slot to a new position or change some of its properties, use `MixedDevice.transitionToConfiguration()` . This method takes: • A new source configuration that represents the next state for the source. • A duration that specifies how long the animation should take, relative to the timeline of the video. If the duration is set to 0, the transition happens on the next frame that is mixed. • An optional callback that informs you when the animation is complete. The callback may be useful for chaining animations.

Mixed Audio Device

Configuration

`MixedAudioDeviceConfiguration` on Android

`IVSMixedAudioDeviceConfiguration` on iOS

Name	Type	Description
`channels`	Integer	Number of output channels from the audio mixer. Valid values: 1, 2. 1 is mono audio; 2, stereo audio. Default: 2.

Source Configuration

`MixedAudioDeviceSourceConfiguration` on Android

`IVSMixedAudioDeviceSourceConfiguration` on iOS

Name	Type	Description
`gain`	Float	Audio gain. This is a multiplier, so any value above 1 increases the gain; any value below 1, decreases it. Valid values: 0-2. Default: 1.

Mixed Image Device

Configuration

`MixedImageDeviceConfiguration` on Android

`IVSMixedImageDeviceConfiguration` on iOS

Name	Type	Description
`size`	Vec2	Size of the video canvas.
`targetFramerate`	Integer	Number of target frames per second for the mixed device. On average, this value should be met, but the system may drop frames under certain circumstances (e.g., high CPU or GPU load).

Name	Type	Description
`transparencyEnabled`	Boolean	This enables blending using the `alpha` property on image source configurations. Setting this to `true` increases memory and CPU consumption. Default: `false`.

Source Configuration

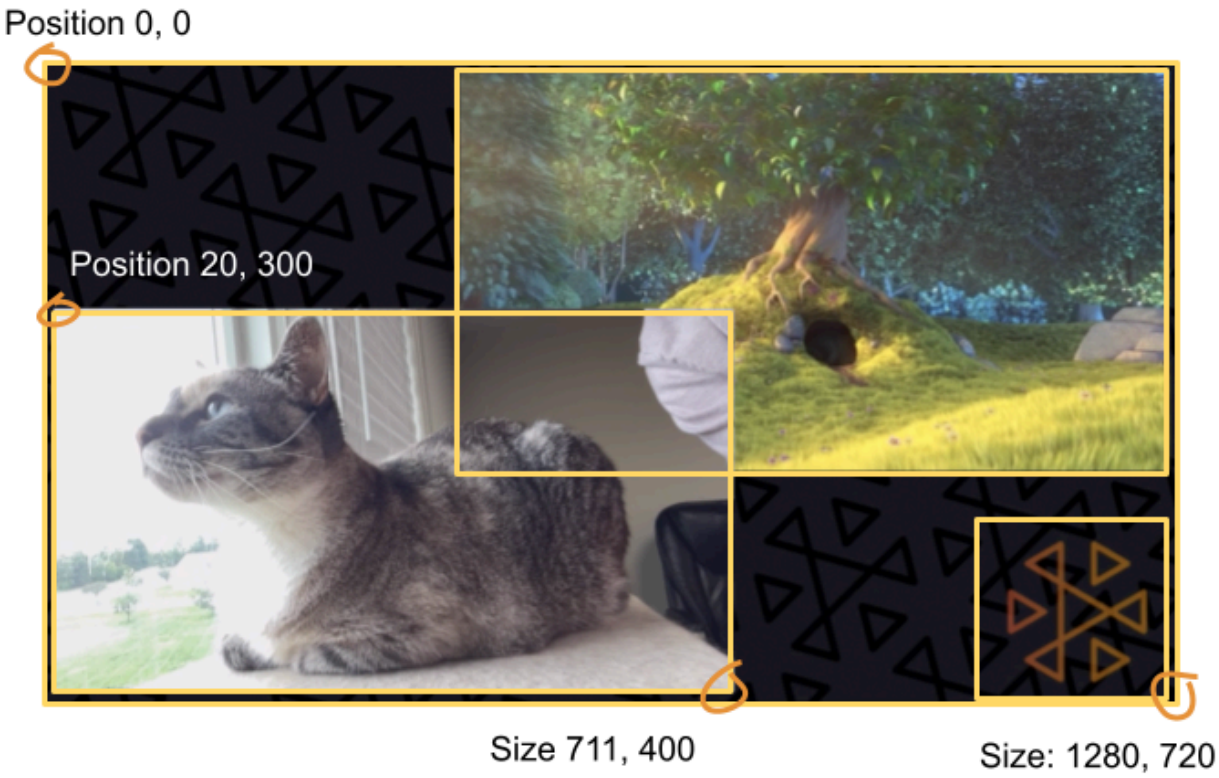
`MixedImageDeviceSourceConfiguration` on Android

`IVSMixedImageDeviceSourceConfiguration` on iOS

Name	Type	Description
`alpha`	Float	Alpha of the slot. This is multiplicative with any alpha values in the image. Valid values: 0-1. 0 is fully transparent and 1 is fully opaque. Default: 1.
`aspect`	AspectMode	Aspect-ratio mode for any image rendered in the slot. Valid values: • `Fill` — Maintain the aspect ratio of the image but fill the slot. The image is cropped if needed. • `Fit` — Maintain the aspect ratio of the image but fit the entire image into the slot. The slot may have a letterbox or pillarbox if necessary. The letter/pillarbox is in the `fillColor` if that value was set; otherwise, transparent (which may appear black if the canvas color behind the image is black). • None — Do not maintain the aspect ratio of the image. The image is scaled to match the dimensions of the slot. Default: `Fit`

Name	Type	Description
`fillColor`	Vec4	Fill color to be used with `aspect Fit` when the slot and image aspect ratios do not match. The format is (red, green, blue, alpha). Valid value (for each channel): 0-1. Default: (0, 0, 0, 0).
`position`	Vec2	Slot position (in pixels), relative to the top-left corner of the canvas. The origin of the slot also is top-left.
`size`	Vec2	Size of the slot, in pixels. Setting this value also sets `matchCanvasSize` to `false`. Default: (0, 0); however, because `matchCanvasSize` defaults to `true`, the rendered size of the slot is the canvas size, not (0, 0).
`zIndex`	Float	Relative ordering of slots. Slots with higher `zIndex` values are drawn on top of slots with lower `zIndex` values.

Creating and Configuring a Mixed Image Device



Here, we create a scene similar to the one at the beginning of this guide, with three on-screen elements:

- Bottom-left slot for a camera.
- Bottom-right slot for a logo overlay.
- Top-right slot for a movie.

Note that the origin for the canvas is the top-left corner and this is the same for the slots. Hence, positioning a slot at (0, 0) puts it in the top-left corner with the entire slot visible.

iOS

```
let deviceDiscovery = IVSDeviceDiscovery()
let mixedImageConfig = IVSMixedImageDeviceConfiguration()
mixedImageConfig.size = CGSize(width: 1280, height: 720)
try mixedImageConfig.setTargetFramerate(60)
mixedImageConfig.isTransparencyEnabled = true
let mixedImageDevice = deviceDiscovery.createMixedImageDevice(with: mixedImageConfig)
```

```
// Bottom Left
let cameraConfig = IVSMixedImageDeviceSourceConfiguration()
cameraConfig.size = CGSize(width: 320, height: 180)
cameraConfig.position = CGPoint(x: 20, y: mixedImageConfig.size.height -
 cameraConfig.size.height - 20)
cameraConfig.zIndex = 2
let camera = deviceDiscovery.listLocalDevices().first(where: { $0 is IVSCamera }) as?
 IVSCamera
let cameraSource = IVSMixedImageDeviceSource(configuration: cameraConfig, device:
 camera)
mixedImageDevice.add(cameraSource)

// Top Right
let streamConfig = IVSMixedImageDeviceSourceConfiguration()
streamConfig.size = CGSize(width: 640, height: 320)
streamConfig.position = CGPoint(x: mixedImageConfig.size.width -
 streamConfig.size.width - 20, y: 20)
streamConfig.zIndex = 1
let streamDevice = deviceDiscovery.createImageSource(withName: "stream")
let streamSource = IVSMixedImageDeviceSource(configuration: streamConfig, device:
 streamDevice)
mixedImageDevice.add(streamSource)

// Bottom Right
let logoConfig = IVSMixedImageDeviceSourceConfiguration()
logoConfig.size = CGSize(width: 320, height: 180)
logoConfig.position = CGPoint(x: mixedImageConfig.size.width - logoConfig.size.width -
 20,
                              y: mixedImageConfig.size.height - logoConfig.size.height
 - 20)
logoConfig.zIndex = 3
let logoDevice = deviceDiscovery.createImageSource(withName: "logo")
let logoSource = IVSMixedImageDeviceSource(configuration: logoConfig, device:
 logoDevice)
mixedImageDevice.add(logoSource)
```

Android

```
val deviceDiscovery = DeviceDiscovery(this /* context */)
val mixedImageConfig = MixedImageDeviceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(1280f, 720f))
    setTargetFramerate(60)
    setEnableTransparency(true)
```

```
}
val mixedImageDevice = deviceDiscovery.createMixedImageDevice(mixedImageConfig)

// Bottom Left
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, 180f))
    setPosition(BroadcastConfiguration.Vec2(20f, mixedImageConfig.size.y - size.y -
 20))
    setZIndex(2)
}
val camera = deviceDiscovery.listLocalDevices().firstNotNullOf { it as? CameraSource }
val cameraSource = MixedImageDeviceSource(cameraConfig, camera)
mixedImageDevice.addSource(cameraSource)

// Top Right
val streamConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(640f, 320f))
    setPosition(BroadcastConfiguration.Vec2(mixedImageConfig.size.x - size.x - 20,
 20f))
    setZIndex(1)
}
val streamDevice = deviceDiscovery.createImageInputSource(streamConfig.size)
val streamSource = MixedImageDeviceSource(streamConfig, streamDevice)
mixedImageDevice.addSource(streamSource)

// Bottom Right
val logoConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, 180f))
    setPosition(BroadcastConfiguration.Vec2(mixedImageConfig.size.x - size.x - 20,
 mixedImageConfig.size.y - size.y - 20))
    setZIndex(1)
}
val logoDevice = deviceDiscovery.createImageInputSource(logoConfig.size)
val logoSource = MixedImageDeviceSource(logoConfig, logoDevice)
mixedImageDevice.addSource(logoSource)
```

Removing Sources

To remove a source, call `MixedDevice.remove` with the `Source` object you want to remove.

Animations with Transitions

The transition method replaces a source's configuration with a new configuration. This replacement can be animated over time by setting a duration higher than 0, in seconds.

Which Properties Can Be Animated?

Not all properties in the slot structure can be animated. Any properties based on Float types can be animated; other properties take effect at either the start or end of the animation.

Name	Can It Be Animated?	Impact Point
`Audio.gain`	Yes	Interpolated
`Image.alpha`	Yes	Interpolated
`Image.aspect`	No	End
`Image.fillColor`	Yes	Interpolated
`Image.position`	Yes	Interpolated
`Image.size`	Yes	Interpolated
`Image.zIndex` Note: The `zIndex` moves 2D planes through 3D space, so the transition happens when the two planes cross at some point in the middle of the animation. This could be computed, but it depends on the starting and ending `zIndex` values. For a smoother transition, combine this with `alpha`.	Yes	Unknown

Simple Examples

Below are examples of a full-screen camera takeover using the configuration defined above in Creating and Configuring a Mixed Image Device. This is animated over 0.5 seconds.

iOS

```
// Continuing the example from above, modifying the existing cameraConfig object.
cameraConfig.size = CGSize(width: 1280, height: 720)
cameraConfig.position = CGPoint.zero
cameraSource.transition(to: cameraConfig, duration: 0.5) { completed in
    if completed {
        print("Animation completed")
    } else {
        print("Animation interrupted")
    }
}
```

Android

```
// Continuing the example from above, modifying the existing cameraConfig object.
cameraConfig.setSize(BroadcastConfiguration.Vec2(1280f, 720f))
cameraConfig.setPosition(BroadcastConfiguration.Vec2(0f, 0f))
cameraSource.transitionToConfiguration(cameraConfig, 500) { completed ->
    if (completed) {
        print("Animation completed")
    } else {
        print("Animation interrupted")
    }
}
```

Mirroring the Broadcast

To mirror an attached image device in the broadcast in this direction ...	Use a negative value for ...
Horizontally	The width of the slot
Vertically	The height of the slot
Both horizontally and vertically	Both the width and height of the slot

The position will need to be adjusted by the same value, to put the slot in the correct position when mirrored.

Below are examples for mirroring the broadcast horizontally and vertically.

iOS

Horizontal mirroring:

```
let cameraSource = IVSMixedImageDeviceSourceConfiguration()
cameraSource.size = CGSize(width: -320, height: 720)
// Add 320 to position x since our width is -320
cameraSource.position = CGPoint(x: 320, y: 0)
```

Vertical mirroring:

```
let cameraSource = IVSMixedImageDeviceSourceConfiguration()
cameraSource.size = CGSize(width: 320, height: -720)
// Add 720 to position y since our height is -720
cameraSource.position = CGPoint(x: 0, y: 720)
```

Android

Horizontal mirroring:

```
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(-320f, 180f))
   // Add 320f to position x since our width is -320f
    setPosition(BroadcastConfiguration.Vec2(320f, 0f))
}
```

Vertical mirroring:

```
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, -180f))
    // Add 180f to position y since our height is -180f
    setPosition(BroadcastConfiguration.Vec2(0f, 180f))
}
```

Note: This mirroring is different than the `setMirrored` method on `ImagePreviewView` (Android) and `IVSImagePreviewView` (iOS). That method affects only the local preview view on the device and does not impact the broadcast.

IVS Broadcast SDK: Token Exchange | Real-Time Streaming

Token exchange enables you to upgrade or downgrade participant-token capabilities and update token attributes within the broadcast SDK, without requiring participants to reconnect. This is useful for scenarios like co-hosting, where participants may start with subscribe-only capabilities and later need publish capabilities.

Token exchange is supported in both the mobile and web broadcast SDKs. When a participant exchanges a token, server-side composition detects the updated attributes in real time and automatically adjusts the layout — for example, reassigning the featured slot, reordering participants, or moving a participant into the picture-in-picture overlay — without requiring a reconnect.

Limitation: Token exchange only works with tokens created on your server using a key pair. It does not work with tokens created via the CreateParticipantToken API.

Exchanging Tokens

Exchanging tokens is straightforward: call the `exchangeToken` API on the `Stage` / `IVSStage` object and provide the new token. If the `capabilities` of the new token are different than those of the previous token, the new token's capabilities are evaluated immediately. For example, if the previous token did not have the `publish` capability and the new token does, the stage strategy functions for publishing are invoked, allowing the host application to decide if they want to publish right away with the new capability, or wait. The same is true for removed capabilities: if the previous token had the `publish` capability and the new token does not, the participant immediately unpublishes without invoking the stage strategy functions for publishing.

When exchanging a token, the previous and new token must have the same values for the following payload fields:

- `topic`
- `resource`
- `jti`
- `whip_url`
- `events_url`

These fields are immutable. Exchanging a token that modifies an immutable field results in the SDK immediately rejecting the exchange.

The remaining fields can be changed, including:

- `attributes`
- `capabilities`
- `user`
- `_id`
- `iat`
- `exp`

iOS

```
let stage = try IVSStage(token: originalToken, strategy: self)
stage.join()
stage.exchangeToken(newToken)
```

Android

```
val stage = Stage(context, originalToken, strategy)
stage.join()
stage.exchangeToken(newToken)
```

Web

```
const stage = new Stage(originalToken, strategy);
await stage.join();
await stage.exchangeToken(newToken);
```

Receiving Updates

A function in `StageRenderer` / `IVSStageRenderer` receives updates about already-published, remote participants that exchange their tokens to update their `userId` or `attributes`. Remote participants that are not already publishing will have their updated `userId` and `attributes` exposed via the existing on`ParticipantJoined` / `participantDidJoin` renderer functions if they eventually publish.

iOS

```
class MyStageRenderer: NSObject, IVSStageRenderer {
    func stage(_ stage: IVSStage, participantMetadataDidUpdate participant:
 IVSParticipantInfo) {
        // participant will be a new IVSParticipantInfo instance with updated
 properties.
    }
}
```

Android

```
private val stageRenderer = object : StageRenderer {
    override fun onParticipantMetadataUpdated(stage: Stage, participantInfo:
 ParticipantInfo) {
        // participantInfo will be a new ParticipantInfo instance with updated
 properties.
    }
}
```

Web

```
stage.on(StageEvents.STAGE_PARTICIPANT_METADATA_CHANGED, (participantInfo:
 StageParticipantInfo) => {
    // participantInfo properties will be updated with the changed properties
    }
);
```

Visibility of Updates

When a participant exchanges a token to update their `userId` or `attributes`, the visibility of these changes depends on their current publishing state:

- **If the participant is *not* publishing:** The update is processed silently. If they eventually publish, all SDKs will receive the updated `userId` and `attributes` as part of the initial publish event.
- **If the participant *is* already publishing:** The update is broadcast immediately for participants using mobile SDKs v1.37.0+, the web SDK, and server-side composition. Participants using older mobile SDKs do not see the change until the participant unpublishes and republishes.

This table clarifies the matrix of support:

Participant State	Observer: Mobile SDK 1.37.0+, Web SDK, Server-Side Composition	Observer: Older Mobile SDKs
Not publishing (then starts)	# Visible (on publish through participant joined event)	# Visible (on publish through participant joined event)
Already publishing (never republishes)	# Visible (immediately through participant metadata updated event)	# Not Visible
Already publishing (unpublishes and republishes)	# Visible (immediately through participant metadata updated event)	## Eventually Visible (on republish through participant joined event)

IVS Broadcast SDK: Custom Image Sources | Real-Time Streaming

Custom image-input sources allow an application to provide its own image input to the broadcast SDK, instead of being limited to the preset cameras. A custom image source can be as simple as a semi-transparent watermark or static “be right back” scene, or it can allow the app to do additional custom processing like adding beauty filters to the camera.

When you use a custom image-input source for custom control of the camera (such as using beauty-filter libraries that require camera access), the broadcast SDK is no longer responsible for managing the camera. Instead, the application is responsible for handling the camera’s lifecycle correctly. See official platform documentation on how your application should manage the camera.

Android

After you create a `DeviceDiscovery` session, create an image-input source:

```
CustomImageSource imageSource = deviceDiscovery.createImageInputSource(new
 BroadcastConfiguration.Vec2(1280, 720));
```

This method returns a `CustomImageSource`, which is an image source backed by a standard Android Surface. The sublcass `SurfaceSource` can be resized and rotated. You also can create an `ImagePreviewView` to display a preview of its contents.

To retrieve the underlying `Surface`:

```
Surface surface = surfaceSource.getInputSurface();
```

This `Surface` can be used as the output buffer for image producers like Camera2, OpenGL ES, and other libraries. The simplest use case is directly drawing a static bitmap or color into the Surface's Canvas. However, many libraries (such as beauty-filter libraries) provide a method that allows an application to specify an external `Surface` for rendering. You can use such a method to pass this `Surface` to the filter library, which allows the library to output processed frames for the broadcast session to stream.

This `CustomImageSource` can be wrapped in a `LocalStageStream` and returned by the `StageStrategy` to publish to a `Stage`.

iOS

After you create a `DeviceDiscovery` session, create an image-input source:

```
let customSource = broadcastSession.createImageSource(withName: "customSourceName")
```

This method returns an `IVSCustomImageSource`, which is an image source that allows the application to submit `CMSampleBuffers` manually. For supported pixel formats, see the iOS Broadcast SDK Reference; a link to the most current version is in the Amazon IVS Release Notes for the latest broadcast SDK release.

Samples submitted to the custom source will be streamed to the Stage:

```
customSource.onSampleBuffer(sampleBuffer)
```

For streaming video, use this method in a callback. For example, if you're using the camera, then every time a new sample buffer is received from an `AVCaptureSession`, the application can forward the sample buffer to the custom image source. If desired, the application can apply further processing (like a beauty filter) before submitting the sample to the custom image source.

The `IVSCustomImageSource` can be wrapped in an `IVSLocalStageStream` and returned by the `IVSStageStrategy` to publish to a `Stage`.

IVS Broadcast SDK: Custom Audio Sources | Real-Time Streaming

Note: This guide only applies to the IVS real-time streaming Android broadcast SDK. Information for the iOS and web SDKs will be published in the future.

Custom audio-input sources allow an application to provide its own audio input to the broadcast SDK, instead of being limited to the device's built-in microphone. A custom audio source enables applications to stream processed audio with effects, mix multiple audio streams, or integrate with third-party audio processing libraries.

When you use a custom audio-input source, the broadcast SDK is no longer responsible for managing the microphone directly. Instead, your application is responsible for capturing, processing, and submitting audio data to the custom source.

The custom-audio-source workflow follows these steps:

1. Audio input — Create a custom audio source with specified audio format (sample rate, channels, format).
2. Your processing — Capture or generate audio data from your audio processing pipeline.
3. Custom audio source — Submit audio buffers to the custom source using `appendBuffer()`.
4. Stage — Wrap in `LocalStageStream` and publish to the stage via your `StageStrategy`.
5. Participants — Stage participants receive the processed audio in real time.

Android

Creating a Custom Audio Source

After you create a `DeviceDiscovery` session, create a custom audio-input source:

```
DeviceDiscovery deviceDiscovery = new DeviceDiscovery(context);

// Create custom audio source with specific format
CustomAudioSource customAudioSource = deviceDiscovery.createAudioInputSource(
   2,  // Number of channels (1 = mono, 2 = stereo)
   BroadcastConfiguration.AudioSampleRate.RATE_48000,  // Sample rate
   AudioDevice.Format.INT16  // Audio format (16-bit PCM)
);
```

This method returns a `CustomAudioSource`, which accepts raw PCM audio data. The custom audio source must be configured with the same audio format that your audio-processing pipeline produces.

Supported Audio Formats

Parameter	Options	Description
Channels	1 (mono), 2 (stereo)	Number of audio channels.
Sample rate	RATE_16000, RATE_44100, RATE_48000	Audio sample rate in Hz. 48kHz recommended for high quality.
Format	INT16, FLOAT32	Audio sample format. INT16 is 16-bit fixed-point PCM, FLOAT32 is 32-bit floating-point PCM. Both interleaved and planar formats are available.

Submitting Audio Data

To submit audio data to the custom source, use the `appendBuffer()` method:

```
// Prepare audio data in a ByteBuffer
ByteBuffer audioBuffer = ByteBuffer.allocateDirect(bufferSize);
audioBuffer.put(pcmAudioData);  // Your processed audio data

// Calculate the number of bytes
long byteCount = pcmAudioData.length;

// Submit audio to the custom source
// presentationTimeUs should be generated by and come from your audio source
int samplesProcessed = customAudioSource.appendBuffer(
   audioBuffer,
   byteCount,
   presentationTimeUs
);

if (samplesProcessed > 0) {
   Log.d(TAG, "Successfully submitted " + samplesProcessed + " samples");
} else {
   Log.w(TAG, "Failed to submit audio samples");
```

```
}

// Clear buffer for reuse
audioBuffer.clear();
```

Important considerations:

- Audio data must be in the format specified when creating the custom source.
- Timestamps should be monotonically increasing and provided by your audio source for smooth audio playback.
- Submit audio regularly to avoid gaps in the stream.
- The method returns the number of samples processed (0 indicates failure).

Publishing to a Stage

Wrap the `CustomAudioSource` in an `AudioLocalStageStream` and return it from your `StageStrategy`:

```
// Create the audio stream from custom source
AudioLocalStageStream audioStream = new AudioLocalStageStream(customAudioSource);

// Define your stage strategy
Strategy stageStrategy = new Strategy() {
   @NonNull
   @Override
   public List<LocalStageStream> stageStreamsToPublishForParticipant(
         @NonNull Stage stage,
         @NonNull ParticipantInfo participantInfo) {
      List<LocalStageStream> streams = new ArrayList<>();
      streams.add(audioStream);  // Publish custom audio
      return streams;
   }

   @Override
   public boolean shouldPublishFromParticipant(
         @NonNull Stage stage,
         @NonNull ParticipantInfo participantInfo) {
      return true;  // Control when to publish
   }

   @Override
```

```
    public Stage.SubscribeType shouldSubscribeToParticipant(
          @NonNull Stage stage,
          @NonNull ParticipantInfo participantInfo) {
       return Stage.SubscribeType.AUDIO_VIDEO;
    }
};

// Create and join the stage
Stage stage = new Stage(context, stageToken, stageStrategy);
```

Complete Example: Audio Processing Integration

Here's a complete example showing integration with an audio-processing SDK:

```
public class AudioStreamingActivity extends AppCompatActivity {
    private DeviceDiscovery deviceDiscovery;
    private CustomAudioSource customAudioSource;
    private AudioLocalStageStream audioStream;
    private Stage stage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Configure audio manager
        StageAudioManager.getInstance(this)
            .setPreset(StageAudioManager.UseCasePreset.VIDEO_CHAT);

        // Initialize IVS components
        initializeIVSStage();

        // Initialize your audio processing SDK
        initializeAudioProcessing();
    }

    private void initializeIVSStage() {
        deviceDiscovery = new DeviceDiscovery(this);

        // Create custom audio source (48kHz stereo, 16-bit)
        customAudioSource = deviceDiscovery.createAudioInputSource(
            2,  // Stereo
            BroadcastConfiguration.AudioSampleRate.RATE_48000,
            AudioDevice.Format.INT16
```

```
        );

        // Create audio stream
        audioStream = new AudioLocalStageStream(customAudioSource);

        // Create stage with strategy
        Strategy strategy = new Strategy() {
            @NonNull
            @Override
            public List<LocalStageStream> stageStreamsToPublishForParticipant(
                    @NonNull Stage stage,
                    @NonNull ParticipantInfo participantInfo) {
                return Collections.singletonList(audioStream);
            }

            @Override
            public boolean shouldPublishFromParticipant(
                    @NonNull Stage stage,
                    @NonNull ParticipantInfo participantInfo) {
                return true;
            }

            @Override
            public Stage.SubscribeType shouldSubscribeToParticipant(
                    @NonNull Stage stage,
                    @NonNull ParticipantInfo participantInfo) {
                return Stage.SubscribeType.AUDIO_VIDEO;
            }
        };

        stage = new Stage(this, getStageToken(), strategy);
    }

    private void initializeAudioProcessing() {
        // Initialize your audio processing SDK
        // Set up callback to receive processed audio
        yourAudioSDK.setAudioCallback(new AudioCallback() {
            @Override
            public void onProcessedAudio(byte[] audioData, int sampleRate,
                                         int channels, long timestamp) {
                // Submit processed audio to IVS Stage
                submitAudioToStage(audioData, timestamp);
            }
        });
```

```
    }

    // The timestamp is required to come from your audio source and you
    // should not be generating one on your own, unless your audio source
    // does not provide one. If that is the case, create your own epoch
    // timestamp and manually calculate the duration between each sample
    // using the number of frames and frame size.

    private void submitAudioToStage(byte[] audioData, long timestamp) {
        try {
            // Allocate direct buffer
            ByteBuffer buffer = ByteBuffer.allocateDirect(audioData.length);
            buffer.put(audioData);

            // Submit to custom audio source
            int samplesProcessed = customAudioSource.appendBuffer(
                buffer,
                audioData.length,
                timestamp > 0 ? timestamp : System.nanoTime() / 1000
            );

            if (samplesProcessed <= 0) {
                Log.w(TAG, "Failed to submit audio samples");
            }

            buffer.clear();
        } catch (Exception e) {
            Log.e(TAG, "Error submitting audio: " + e.getMessage(), e);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (stage != null) {
            stage.release();
        }
    }
}
```

Best Practices

Audio Format Consistency

Ensure the audio format you submit matches the format specified when creating the custom source:

```
// If you create with 48kHz stereo INT16
customAudioSource = deviceDiscovery.createAudioInputSource(
   2, RATE_48000, INT16
);

// Your audio data must be:
// - 2 channels (stereo)
// - 48000 Hz sample rate
// - 16-bit interleaved PCM format
```

Buffer Management

Use direct `ByteBuffers` and reuse them to minimize garbage collection:

```
// Allocate once
private ByteBuffer audioBuffer = ByteBuffer.allocateDirect(BUFFER_SIZE);

// Reuse in callback
public void onAudioData(byte[] data) {
   audioBuffer.clear();
   audioBuffer.put(data);
   customAudioSource.appendBuffer(audioBuffer, data.length, getTimestamp());
   audioBuffer.clear();
}
```

Timing and Synchronization

You must use timestamps provided by your audio source for smooth audio playback. If your audio source does not provide its own timestamp, create your own epoch timestamp and manually calculate the duration between each sample using the number of frames and frame size.

```
// "audioFrameTimestamp" should be generated by your audio source
// Consult your audio source's documentation for information on how to get this
long timestamp = audioFrameTimestamp;
```

Error Handling

Always check the return value of `appendBuffer()`:

```
int samplesProcessed = customAudioSource.appendBuffer(buffer, count, timestamp);

if (samplesProcessed <= 0) {
   Log.w(TAG, "Audio submission failed - buffer may be full or format mismatch");
   // Handle error: check format, reduce submission rate, etc.
}
```

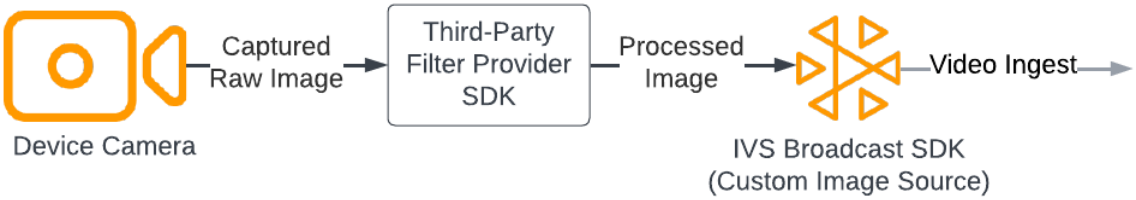
IVS Broadcast SDK: Third-Party Camera Filters | Real-Time Streaming

This guide assumes you are already familiar with custom image sources as well as integrating the IVS real-time streaming broadcast SDK into your application.

Camera filters enable live-stream creators to augment or alter their facial or background appearance. This potentially can increase viewer engagement, attract viewers, and enhance the live-streaming experience.

Integrating Third-Party Camera Filters

You can integrate third-party camera filter SDKs with the IVS broadcast SDK by feeding the filter SDK's output to a custom image input source. A custom image-input source allows an application to provide its own image input to the Broadcast SDK. A third-party filter provider's SDK may manage the camera's lifecycle to process images from the camera, apply a filter effect, and output it in a format that can be passed to a custom image source.



Consult your third-party filter provider's documentation for built-in methods to convert a camera frame, with the filter effect, applied to a format that can be passed to a custom image-input source. The process varies, depending on which version of the IVS broadcast SDK is used:

- **Web** — The filter provider must be able to render its output to a canvas element. The captureStream method can then be used to return a MediaStream of the canvas's contents. The MediaStream can then be converted to an instance of a LocalStageStream and published to a Stage.
- **Android** — The filter provider's SDK can either render a frame to an Android `Surface` provided by the IVS broadcast SDK or convert the frame to a bitmap. If using a bitmap, it can then be rendered to the underlying `Surface` provided by the custom image source, by unlocking and writing to a canvas.
- **iOS** — A third-party filter provider's SDK must provide a camera frame with a filter effect applied as a `CMSampleBuffer`. Refer to your third-party filter vendor SDK's documentation for information on how to get a `CMSampleBuffer` as the final output after a camera image is processed.

Using BytePlus with the IVS Broadcast SDK

This document explains how to use the BytePlus Effects SDK with the IVS broadcast SDK.

Android

Install and Set Up the BytePlus Effects SDK

See the BytePlus Android Access Guide for details on how to install, initialize, and set up the BytePlus Effects SDK.

Set Up the Custom Image Source

After initializing the SDK, feed processed camera frames with a filter effect applied to a custom-image input source. To do that, create an instance of a `DeviceDiscovery` object and create a custom image source. Note that when you use a custom image input source for custom control of the camera, the broadcast SDK is no longer responsible for managing the camera. Instead, the application is responsible for handling the camera's lifecycle correctly.

Java

```
var deviceDiscovery = DeviceDiscovery(applicationContext)
var customSource = deviceDiscovery.createImageInputSource( BroadcastConfiguration.Vec2(
720F, 1280F
))
var surface: Surface = customSource.inputSurface
```

```
var filterStream = ImageLocalStageStream(customSource)
```

Convert Output to a Bitmap and Feed to Custom Image Input Source

To enable camera frames with a filter effect applied from the BytePlus Effect SDK to be forwarded directly to the IVS broadcast SDK, convert the BytePlus Effects SDK's output of a texture to a bitmap. When an image is processed, the `onDrawFrame()` method is invoked by the SDK. The `onDrawFrame()` method is a public method of Android's GLSurfaceView.Renderer interface. In the Android sample app provided by BytePlus, this method is called on every camera frame; it outputs a texture. Concurrently, you can supplement the `onDrawFrame()` method with logic to convert this texture to a bitmap and feed it to a custom image input source. As shown in the following code sample, use the `transferTextureToBitmap` method provided by the BytePlus SDK to do this conversion. This method is provided by the com.bytedance.labcv.core.util.ImageUtil library from the BytePlus Effects SDK, as shown in the following code sample.You can then render to the underlying Android `Surface` of a `CustomImageSource` by writing the resulting bitmap to a Surface's canvas. Many successive invocations of `onDrawFrame()` results in a sequence of bitmaps, and when combined, creates a stream of video.

Java

```
import com.bytedance.labcv.core.util.ImageUtil;
...
protected ImageUtil imageUtility;
...


@Override
public void onDrawFrame(GL10 gl10) {
  ...
  // Convert BytePlus output to a Bitmap
  Bitmap outputBt = imageUtility.transferTextureToBitmap(output.getTexture(),ByteEffect

  Constants.TextureFormat.Texture2D,output.getWidth(), output.getHeight());

  canvas = surface.lockCanvas(null);
  canvas.drawBitmap(outputBt, 0f, 0f, null);
  surface.unlockCanvasAndPost(canvas);
```

Using DeepAR with the IVS Broadcast SDK

This document explains how to use the DeepAR SDK with the IVS broadcast SDK.

Android

See the Android Integration Guide from DeepAR for details on how to integrate the DeepAR SDK with the Android IVS broadcast SDK.

iOS

See the iOS Integration Guide from DeepAR for details on how to integrate the DeepAR SDK with the iOS IVS broadcast SDK.

Using Snap with the IVS Broadcast SDK

This document explains how to use Snap's Camera Kit SDK with the IVS broadcast SDK.

Web

This section assumes you are already familiar with publishing and subscribing to video using the Web Broadcast SDK.

To integrate Snap's Camera Kit SDK with the IVS real-time streaming Web broadcast SDK, you need to:

1. Install the Camera Kit SDK and Webpack. (Our example uses Webpack as the bundler, but you can use any bundler of your choice.)
2. Create `index.html`.
3. Add setup elements.
4. Create `index.css`.
5. Display and set up participants.
6. Display connected cameras and microphones.
7. Create a Camera Kit session.
8. Fetch lenses and populate lens selector.
9. Render the output from a Camera Kit session to a canvas.
10. Create a function to populate the Lens dropdown.
11. Provide Camera Kit with a media source for rendering and publish a `LocalStageStream`.
12. Create `package.json`.
13. Create a Webpack config file.
14. Set up an HTTPS server and test.

Each of these steps is described below.

Install the Camera Kit SDK and Webpack

In this example we use Webpack as our bundler; however, you can use any bundler.

```
npm i @snap/camera-kit webpack webpack-cli
```

Create index.html

Next, create the HTML boilerplate and import the Web broadcast SDK as a script tag. In the following code, be sure to replace <SDK version> with the broadcast SDK version that you are using.

HTML

```
<!--
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */
-->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <title>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</title>

  <!-- Fonts and Styling -->
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Roboto:300,300italic,700,700italic" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/
normalize.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/milligram/1.4.1/
milligram.css" />
  <link rel="stylesheet" href="./index.css" />

  <!-- Stages in Broadcast SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>
```

```
<body>
  <!-- Introduction -->
  <header>
    <h1>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</h1>

    <p>This sample is used to demonstrate basic HTML / JS usage. <b><a href="https://
docs.aws.amazon.com/ivs/latest/LowLatencyUserGuide/multiple-hosts.html">Use the AWS
 CLI</a></b> to create a <b>Stage</b> and a corresponding <b>ParticipantToken</b>.
 Multiple participants can load this page and put in their own tokens. You can <b><a
 href="https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#glossary"
 target="_blank">read more about stages in our public docs.</a></b></p>
  </header>
  <hr />

  <!-- Setup Controls -->

  <!-- Display Local Participants -->

  <!-- Lens Selector -->

  <!-- Display Remote Participants -->

  <!-- Load All Desired Scripts -->
```

Add Setup Elements

Create the HTML for selecting a camera, microphone, and lens and specifying a participant token:

HTML

```
<!-- Setup Controls -->
  <div class="row">
    <div class="column">
      <label for="video-devices">Select Camera</label>
      <select disabled id="video-devices">
        <option selected disabled>Choose Option</option>
      </select>
    </div>
    <div class="column">
      <label for="audio-devices">Select Microphone</label>
      <select disabled id="audio-devices">
        <option selected disabled>Choose Option</option>
```

```
      </select>
    </div>
    <div class="column">
      <label for="token">Participant Token</label>
      <input type="text" id="token" name="token" />
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
      <button class="button" style="margin: auto; width: 100%" id="join-button">Join
 Stage</button>
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
      <button class="button" style="margin: auto; width: 100%" id="leave-button">Leave
 Stage</button>
    </div>
  </div>
```

Add additional HTML beneath that to display camera feeds from local and remote participants:

HTML

```
 <!-- Local Participant -->
<div class="row local-container">
    <canvas id="canvas"></canvas>

    <div class="column" id="local-media"></div>
    <div class="static-controls hidden" id="local-controls">
      <button class="button" id="mic-control">Mute Mic</button>
      <button class="button" id="camera-control">Mute Camera</button>
    </div>
  </div>


  <hr style="margin-top: 5rem"/>

  <!-- Remote Participants -->
  <div class="row">
    <div id="remote-media"></div>
  </div>
```

Load additional logic, including helper methods for setting up the camera and the bundled JavaScript file. (Later in this section, you will create these JavaScript files and bundle them into a single file, so you can import Camera Kit as a module. The bundled JavaScript file will contain the logic for setting up Camera Kit, applying a Lens, and publishing the camera feed with a Lens

applied to a stage.) Add closing tags for the body and `html` elements to complete the creation of `index.html`.

HTML

```
<!-- Load all Desired Scripts -->
  <script src="./helpers.js"></script>
  <script src="./media-devices.js"></script>
  <!-- <script type="module" src="./stages-simple.js"></script> -->
  <script src="./dist/bundle.js"></script>
</body>
</html>
```

Create index.css

Create a CSS source file to style the page. We will not be going over this code so as to focus on the logic for managing a Stage and integrating with Snap's Camera Kit SDK.

CSS

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

html,
body {
  margin: 2rem;
  box-sizing: border-box;
  height: 100vh;
  max-height: 100vh;
  display: flex;
  flex-direction: column;
}

hr {
  margin: 1rem 0;
}

table {
  display: table;
}

canvas {
```

```
  margin-bottom: 1rem;
  background: green;
}

video {
  margin-bottom: 1rem;
  background: black;
  max-width: 100%;
  max-height: 150px;
}

.log {
  flex: none;
  height: 300px;
}

.content {
  flex: 1 0 auto;
}

.button {
  display: block;
  margin: 0 auto;
}

.local-container {
  position: relative;
}

.static-controls {
  position: absolute;
  margin-left: auto;
  margin-right: auto;
  left: 0;
  right: 0;
  bottom: -4rem;
  text-align: center;
}

.static-controls button {
  display: inline-block;
}

.hidden {
```

```
  display: none;
}

.participant-container {
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
  margin: 1rem;
}

video {
  border: 0.5rem solid #555;
  border-radius: 0.5rem;
}
.placeholder {
  background-color: #333333;
  display: flex;
  text-align: center;
  margin-bottom: 1rem;
}
.placeholder span {
  margin: auto;
  color: white;
}
#local-media {
  display: inline-block;
  width: 100vw;
}

#local-media video {
  max-height: 300px;
}

#remote-media {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
  width: 100%;
}

#lens-selector {
  width: 100%;
```

```
  margin-bottom: 1rem;
}
```

Display and Set Up Participants

Next, create `helpers.js`, which contains helper methods that you will use to display and set up participants:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

function setupParticipant({ isLocal, id }) {
  const groupId = isLocal ? 'local-media' : 'remote-media';
  const groupContainer = document.getElementById(groupId);

  const participantContainerId = isLocal ? 'local' : id;
  const participantContainer = createContainer(participantContainerId);
  const videoEl = createVideoEl(participantContainerId);

  participantContainer.appendChild(videoEl);
  groupContainer.appendChild(participantContainer);

  return videoEl;
}

function teardownParticipant({ isLocal, id }) {
  const groupId = isLocal ? 'local-media' : 'remote-media';
  const groupContainer = document.getElementById(groupId);
  const participantContainerId = isLocal ? 'local' : id;

  const participantDiv = document.getElementById(
    participantContainerId + '-container'
  );
  if (!participantDiv) {
    return;
  }
  groupContainer.removeChild(participantDiv);
}

function createVideoEl(id) {
  const videoEl = document.createElement('video');
```

```
  videoEl.id = id;
  videoEl.autoplay = true;
  videoEl.playsInline = true;
  videoEl.srcObject = new MediaStream();
  return videoEl;
}

function createContainer(id) {
  const participantContainer = document.createElement('div');
  participantContainer.classList = 'participant-container';
  participantContainer.id = id + '-container';

  return participantContainer;
}
```

Display Connected Cameras and Microphones

Next, create `media-devices.js`, which contains helper methods for displaying cameras and microphones connected to your device:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

/**
 * Returns an initial list of devices populated on the page selects
 */
async function initializeDeviceSelect() {
  const videoSelectEl = document.getElementById('video-devices');
  videoSelectEl.disabled = false;

  const { videoDevices, audioDevices } = await getDevices();
  videoDevices.forEach((device, index) => {
    videoSelectEl.options[index] = new Option(device.label, device.deviceId);
  });

  const audioSelectEl = document.getElementById('audio-devices');

  audioSelectEl.disabled = false;
  audioDevices.forEach((device, index) => {
    audioSelectEl.options[index] = new Option(device.label, device.deviceId);
  });
```

```
}

/**
 * Returns all devices available on the current device
 */
async function getDevices() {
  // Prevents issues on Safari/FF so devices are not blank
  await navigator.mediaDevices.getUserMedia({ video: true, audio: true });

  const devices = await navigator.mediaDevices.enumerateDevices();
  // Get all video devices
  const videoDevices = devices.filter((d) => d.kind === 'videoinput');
  if (!videoDevices.length) {
    console.error('No video devices found.');
  }

  // Get all audio devices
  const audioDevices = devices.filter((d) => d.kind === 'audioinput');
  if (!audioDevices.length) {
    console.error('No audio devices found.');
  }

  return { videoDevices, audioDevices };
}

async function getCamera(deviceId) {
  // Use Max Width and Height
  return navigator.mediaDevices.getUserMedia({
    video: {
      deviceId: deviceId ? { exact: deviceId } : null,
    },
    audio: false,
  });
}

async function getMic(deviceId) {
  return navigator.mediaDevices.getUserMedia({
    video: false,
    audio: {
      deviceId: deviceId ? { exact: deviceId } : null,
    },
  });
}
```

Create a Camera Kit Session

Create `stages.js`, which contains the logic for applying a Lens to the camera feed and publishing the feed to a stage. We recommend copying and pasting the following code block into `stages.js`. You can then review the code piece by piece to understand what's going on in the following sections.

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

const {
  Stage,
  LocalStageStream,
  SubscribeType,
  StageEvents,
  ConnectionState,
  StreamType,
} = IVSBroadcastClient;

import {
  bootstrapCameraKit,
  createMediaStreamSource,
  Transform2D,
} from '@snap/camera-kit';

let cameraButton = document.getElementById('camera-control');
let micButton = document.getElementById('mic-control');
let joinButton = document.getElementById('join-button');
let leaveButton = document.getElementById('leave-button');

let controls = document.getElementById('local-controls');
let videoDevicesList = document.getElementById('video-devices');
let audioDevicesList = document.getElementById('audio-devices');

let lensSelector = document.getElementById('lens-selector');
let session;
let availableLenses = [];

// Stage management
let stage;
let joining = false;
```

```
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;

const liveRenderTarget = document.getElementById('canvas');

const init = async () => {
  await initializeDeviceSelect();

  const cameraKit = await bootstrapCameraKit({
    apiToken: 'INSERT_YOUR_API_TOKEN_HERE',
  });

  session = await cameraKit.createSession({ liveRenderTarget });
  const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    'INSERT_YOUR_LENS_GROUP_ID_HERE',
  ]);

  availableLenses = lenses;
  populateLensSelector(lenses);

  const snapStream = liveRenderTarget.captureStream();

  lensSelector.addEventListener('change', handleLensChange);
  lensSelector.disabled = true;
  cameraButton.addEventListener('click', () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? 'Show Camera' : 'Hide Camera';
  });

  micButton.addEventListener('click', () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? 'Unmute Mic' : 'Mute Mic';
  });

  joinButton.addEventListener('click', () => {
    joinStage(session, snapStream);
  });

  leaveButton.addEventListener('click', () => {
```

```
    leaveStage();
  });
};

async function setCameraKitSource(session, mediaStream) {
  const source = createMediaStreamSource(mediaStream);
  await session.setSource(source);
  source.setTransform(Transform2D.MirrorX);
  session.play();
}

const populateLensSelector = (lenses) => {
  lensSelector.innerHTML = '<option selected disabled>Choose Lens</option>';

  lenses.forEach((lens, index) => {
    const option = document.createElement('option');
    option.value = index;
    option.text = lens.name || `Lens ${index + 1}`;
    lensSelector.appendChild(option);
  });
};

const handleLensChange = (event) => {
  const selectedIndex = parseInt(event.target.value);
  if (session && availableLenses[selectedIndex]) {
    session.applyLens(availableLenses[selectedIndex]);
  }
};

const joinStage = async (session, snapStream) => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById('token').value;

  if (!token) {
    window.alert('Please enter a participant token');
    joining = false;
    return;
  }

  // Retrieve the User Media currently set on the page
```

```
  localCamera = await getCamera(videoDevicesList.value);
  localMic = await getMic(audioDevicesList.value);
  await setCameraKitSource(session, localCamera);

  // Create StageStreams for Audio and Video
  cameraStageStream = new LocalStageStream(snapStream.getVideoTracks()[0]);
  micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

  const strategy = {
    stageStreamsToPublish() {
      return [cameraStageStream, micStageStream];
    },
    shouldPublishParticipant() {
      return true;
    },
    shouldSubscribeToParticipant() {
      return SubscribeType.AUDIO_VIDEO;
    },
  };

  stage = new Stage(token, strategy);

  // Other available events:
  // https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events
  stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
    connected = state === ConnectionState.CONNECTED;

    if (connected) {
      joining = false;
      controls.classList.remove('hidden');
      lensSelector.disabled = false;
    } else {
      controls.classList.add('hidden');
      lensSelector.disabled = true;
    }
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
    console.log('Participant Joined:', participant);
  });

  stage.on(
    StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED,
    (participant, streams) => {
```

```
      console.log('Participant Media Added: ', participant, streams);

      let streamsToDisplay = streams;

      if (participant.isLocal) {
        // Ensure to exclude local audio streams, otherwise echo will occur
        streamsToDisplay = streams.filter(
          (stream) => stream.streamType === StreamType.VIDEO
        );
      }

      const videoEl = setupParticipant(participant);
      streamsToDisplay.forEach((stream) =>
        videoEl.srcObject.addTrack(stream.mediaStreamTrack)
      );
    }
  );

  stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
    console.log('Participant Left: ', participant);
    teardownParticipant(participant);
  });

  try {
    await stage.join();
  } catch (err) {
    joining = false;
    connected = false;
    console.error(err.message);
  }
};

const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;

  cameraButton.innerText = 'Hide Camera';
  micButton.innerText = 'Mute Mic';
  controls.classList.add('hidden');
};
```

```
init();
```

In the first part of this file, we import the broadcast SDK and Camera Kit Web SDK and initialize the variables we will use with each SDK. We create a Camera Kit session by calling `createSession` after bootstrapping the Camera Kit Web SDK. Note that a canvas element object is passed to a session; this tells Camera Kit to render into that canvas.

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

const {
  Stage,
  LocalStageStream,
  SubscribeType,
  StageEvents,
  ConnectionState,
  StreamType,
} = IVSBroadcastClient;

import {
  bootstrapCameraKit,
  createMediaStreamSource,
  Transform2D,
} from '@snap/camera-kit';

let cameraButton = document.getElementById('camera-control');
let micButton = document.getElementById('mic-control');
let joinButton = document.getElementById('join-button');
let leaveButton = document.getElementById('leave-button');

let controls = document.getElementById('local-controls');
let videoDevicesList = document.getElementById('video-devices');
let audioDevicesList = document.getElementById('audio-devices');

let lensSelector = document.getElementById('lens-selector');
let session;
let availableLenses = [];

// Stage management
let stage;
let joining = false;
```

```
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;

const liveRenderTarget = document.getElementById('canvas');

const init = async () => {
  await initializeDeviceSelect();

  const cameraKit = await bootstrapCameraKit({
    apiToken: 'INSERT_YOUR_API_TOKEN_HERE',
  });

  session = await cameraKit.createSession({ liveRenderTarget });
```

Fetch Lenses and Populate Lens Selector

To fetch your Lenses, replace the placeholder for the Lens Group ID with your own, which can be found in the Camera Kit Developer Portal. Populate the Lens selection dropdown using the `populateLensSelector()` function which we will create later.

JavaScript

```
session = await cameraKit.createSession({ liveRenderTarget });
  const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    'INSERT_YOUR_LENS_GROUP_ID_HERE',
  ]);

  availableLenses = lenses;
  populateLensSelector(lenses);
```

Render the Output from a Camera Kit Session to a Canvas

Use the captureStream method to return a `MediaStream` of the canvas's contents. The canvas will contain a video stream of the camera feed with a Lens applied. Also, add event listeners for buttons to mute the camera and microphone as well as event listeners for joining and leaving a stage. In the event listener for joining a stage, we pass in a Camera Kit session and the `MediaStream` from the canvas so it can be published to a stage.

JavaScript

```
const snapStream = liveRenderTarget.captureStream();

  lensSelector.addEventListener('change', handleLensChange);
  lensSelector.disabled = true;
  cameraButton.addEventListener('click', () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? 'Show Camera' : 'Hide Camera';
  });

  micButton.addEventListener('click', () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? 'Unmute Mic' : 'Mute Mic';
  });

  joinButton.addEventListener('click', () => {
    joinStage(session, snapStream);
  });

  leaveButton.addEventListener('click', () => {
    leaveStage();
  });
};
```

Create a Function to Populate the Lens Dropdown

Create the following function to populate the **Lens** selector with the lenses fetched earlier. The **Lens** selector is a UI element on the page that lets you select from a list of lenses to apply to the camera feed. Also, create the `handleLensChange` callback function to apply the specified lens when it is selected from the **Lens** dropdown.

JavaScript

```
const populateLensSelector = (lenses) => {
  lensSelector.innerHTML = '<option selected disabled>Choose Lens</option>';

  lenses.forEach((lens, index) => {
    const option = document.createElement('option');
    option.value = index;
    option.text = lens.name || `Lens ${index + 1}`;
```

```
    lensSelector.appendChild(option);
  });
};

const handleLensChange = (event) => {
  const selectedIndex = parseInt(event.target.value);
  if (session && availableLenses[selectedIndex]) {
    session.applyLens(availableLenses[selectedIndex]);
  }
};
```

Provide Camera Kit with a Media Source for Rendering and Publish a LocalStageStream

To publish a video stream with a Lens applied, create a function called `setCameraKitSource` to pass in the `MediaStream` captured from the canvas earlier. The `MediaStream` from the canvas isn't doing anything at the moment because we have not incorporated our local camera feed yet. We can incorporate our local camera feed by calling the `getCamera` helper method and assigning it to `localCamera` . We can then pass in our local camera feed (via `localCamera`) and the session object to `setCameraKitSource`. The `setCameraKitSource` function converts our local camera feed to a source of media for CameraKit by calling `createMediaStreamSource`. The media source for `CameraKit` is then transformed to mirror the front-facing camera. The Lens effect is then applied to the media source and rendered to the output canvas by calling `session.play()`.

With Lens now applied to the `MediaStream` captured from the canvas, we can then proceed to publishing it to a stage. We do that by creating a `LocalStageStream` with the video tracks from the `MediaStream`. An instance of `LocalStageStream` can then be passed in to a `StageStrategy` to be published.

JavaScript

```
async function setCameraKitSource(session, mediaStream) {
  const source = createMediaStreamSource(mediaStream);
  await session.setSource(source);
  source.setTransform(Transform2D.MirrorX);
  session.play();
}

const joinStage = async (session, snapStream) => {
  if (connected || joining) {
    return;
  }
```

```
  joining = true;

  const token = document.getElementById('token').value;

  if (!token) {
    window.alert('Please enter a participant token');
    joining = false;
    return;
  }

  // Retrieve the User Media currently set on the page
  localCamera = await getCamera(videoDevicesList.value);
  localMic = await getMic(audioDevicesList.value);
  await setCameraKitSource(session, localCamera);
  // Create StageStreams for Audio and Video
  // cameraStageStream = new LocalStageStream(localCamera.getVideoTracks()[0]);
  cameraStageStream = new LocalStageStream(snapStream.getVideoTracks()[0]);
  micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

  const strategy = {
    stageStreamsToPublish() {
      return [cameraStageStream, micStageStream];
    },
    shouldPublishParticipant() {
      return true;
    },
    shouldSubscribeToParticipant() {
      return SubscribeType.AUDIO_VIDEO;
    },
  };
```

The remaining code below is for creating and managing our stage:

JavaScript

```
stage = new Stage(token, strategy);

  // Other available events:
  // https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events

  stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
    connected = state === ConnectionState.CONNECTED;

    if (connected) {
```

```
    joining = false;
    controls.classList.remove('hidden');
  } else {
    controls.classList.add('hidden');
  }
});

stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
  console.log('Participant Joined:', participant);
});

stage.on(
  StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED,
  (participant, streams) => {
    console.log('Participant Media Added: ', participant, streams);

    let streamsToDisplay = streams;

    if (participant.isLocal) {
      // Ensure to exclude local audio streams, otherwise echo will occur
      streamsToDisplay = streams.filter(
        (stream) => stream.streamType === StreamType.VIDEO
      );
    }

    const videoEl = setupParticipant(participant);
    streamsToDisplay.forEach((stream) =>
      videoEl.srcObject.addTrack(stream.mediaStreamTrack)
    );
  }
);

stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
  console.log('Participant Left: ', participant);
  teardownParticipant(participant);
});

try {
  await stage.join();
} catch (err) {
  joining = false;
  connected = false;
  console.error(err.message);
}
```

```
};

const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;

  cameraButton.innerText = 'Hide Camera';
  micButton.innerText = 'Mute Mic';
  controls.classList.add('hidden');
};

init();
```

Create package.json

Create `package.json` and add the following JSON configuration. This file defines our dependencies and includes a script command for bundling our code.

JSON Configuration

```
{
  "dependencies": {
    "@snap/camera-kit": "^0.10.0"
  },
  "name": "ivs-stages-with-snap-camerakit",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "build": "webpack"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "webpack": "^5.95.0",
    "webpack-cli": "^5.1.4"
  }
}
```

Create a Webpack Config File

Create `webpack.config.js` and add the following code. This bundles the code we created thus far so that we can use the import statement to use Camera Kit.

JavaScript

```
const path = require('path');
module.exports = {
  entry: ['./stage.js'],
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Finally, run `npm run build` to bundle your JavaScript as defined in the Webpack config file. For testing purposes, you can then serve HTML and JavaScript from your local computer. In this example, we use Python's `http.server` module.

Set Up an HTTPS Server and Test

To test our code, we need to set up an HTTPS server. Using an HTTPS server for local development and testing of your web app's integration with the Snap Camera Kit SDK will help avoid CORS (Cross-Origin Resource Sharing) issues.

Open a terminal and navigate to the directory where you created all the code up to this point. Run the following command to generate a self-signed SSL/TLS certificate and private key:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
```

This creates two files: `key.pem` (the private key) and `cert.pem` (the self-signed certificate). Create a new Python file named `https_server.py` and add the following code:

Python

```
import http.server
import ssl

# Set the directory to serve files from
DIRECTORY = '.'
```

```
# Create the HTTPS server
server_address = ('', 4443)
httpd = http.server.HTTPServer(
    server_address, http.server.SimpleHTTPRequestHandler)

# Wrap the socket with SSL/TLS
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('cert.pem', 'key.pem')
httpd.socket = context.wrap_socket(httpd.socket, server_side=True)

print(f'Starting HTTPS server on https://localhost:4443, serving {DIRECTORY}')
httpd.serve_forever()
```

Open a terminal, navigate to the directory where you created the `https_server.py` file, and run the following command:

```
python3 https_server.py
```

This starts the HTTPS server on https://localhost:4443, serving files from the current directory. Make sure that the `cert.pem` and `key.pem` files are in the same directory as the `https_server.py` file.

Open your browser and navigate to https://localhost:4443. Since this is a self-signed SSL/TLS certificate, it will not be trusted by your web browser, so you will get a warning. Since this is only for testing purposes, you can bypass the warning. You should then see the AR effect for the Snap Lens you specified earlier applied to your camera feed on screen.

Note that this setup using Python's built-in `http.server` and `ssl` modules is suitable for local development and testing purposes, but it is not recommended for a production environment. The self-signed SSL/TLS certificate used in this setup is not trusted by web browsers and other clients, which means users will encounter security warnings when accessing the server. Also, although we use Python’s built-in http.server and ssl modules in this example, you may choose to use another HTTPS server solution.

Android

To integrate Snap’s Camera Kit SDK with the IVS Android broadcast SDK, you must install the Camera Kit SDK, initialize a Camera Kit session, apply a Lens and feed the Camera Kit session’s output to the custom-image input source.

To install the Camera Kit SDK, add the following to your module's `build.gradle` file. Replace `$cameraKitVersion` with the latest Camera Kit SDK version.

Java

```
implementation "com.snap.camerakit:camerakit:$cameraKitVersion"
```

Initialize and obtain a `cameraKitSession`. Camera Kit also provides a convenient wrapper for Android's CameraX APIs, so you don't have to write complicated logic to use CameraX with Camera Kit. You can use the `CameraXImageProcessorSource` object as a Source for ImageProcessor, which allows you to start camera-preview streaming frames.

Java

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_main);

       // Camera Kit support implementation of ImageProcessor that is backed by
CameraX library:
       // https://developer.android.com/training/camerax
       CameraXImageProcessorSource imageProcessorSource = new
CameraXImageProcessorSource(
           this /*context*/, this /*lifecycleOwner*/
       );
       imageProcessorSource.startPreview(true /*cameraFacingFront*/);

       cameraKitSession = Sessions.newBuilder(this)
               .imageProcessorSource(imageProcessorSource)
               .attachTo(findViewById(R.id.camerakit_stub))
               .build();
   }
```

Fetch and Apply Lenses

You can configure Lenses and their ordering in the carousel on the Camera Kit Developer Portal:

Java

```
// Fetch lenses from repository and apply them
 // Replace LENS_GROUP_ID with Lens Group ID from https://camera-kit.snapchat.com
```

```
cameraKitSession.getLenses().getRepository().get(new Available(LENS_GROUP_ID),
 available -> {
     Log.d(TAG, "Available lenses: " + available);
     Lenses.whenHasFirst(available, lens ->
 cameraKitSession.getLenses().getProcessor().apply(lens, result -> {
          Log.d(TAG,  "Apply lens [" + lens + "] success: " + result);
      }));
});
```

To broadcast, send processed frames to the underlying `Surface` of a custom image source. Use a `DeviceDiscovery` object and create a `CustomImageSource` to return a `SurfaceSource`. You can then render the output from a `CameraKit` session to the underlying `Surface` provided by the `SurfaceSource`.

Java

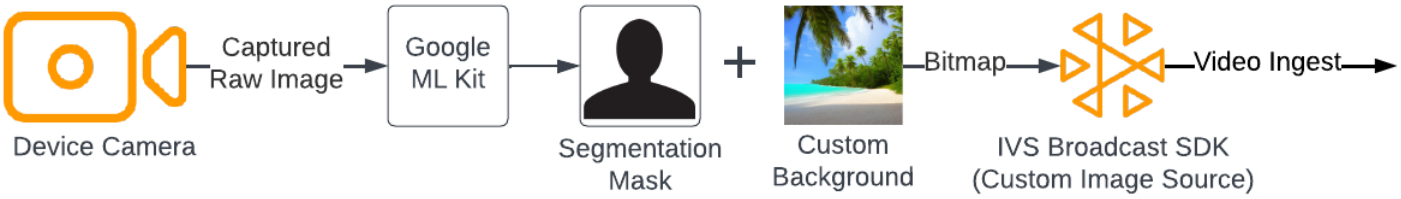
```
val publishStreams = ArrayList<LocalStageStream>()

val deviceDiscovery = DeviceDiscovery(applicationContext)
val customSource =
 deviceDiscovery.createImageInputSource(BroadcastConfiguration.Vec2(720f, 1280f))

cameraKitSession.processor.connectOutput(outputFrom(customSource.inputSurface))
val customStream = ImageLocalStageStream(customSource)

// After rendering the output from a Camera Kit session to the Surface, you can
// then return it as a LocalStageStream to be published by the Broadcast SDK
val customStream: ImageLocalStageStream = ImageLocalStageStream(surfaceSource)
publishStreams.add(customStream)

@Override
fun stageStreamsToPublishForParticipant(stage: Stage, participantInfo:
 ParticipantInfo): List<LocalStageStream> = publishStreams
```

Using Background Replacement with the IVS Broadcast SDK

Background replacement is a type of camera filter that enables live-stream creators to change their backgrounds. As shown in the following diagram, replacing your background involves:

1. Getting a camera image from the live camera feed.
2. Segmenting it into foreground and background components using Google ML Kit.

3. Combining the resulting segmentation mask with a custom background image.
4. Passing it to a Custom Image Source for broadcast.



Web

This section assumes you are already familiar with publishing and subscribing to video using the Web Broadcast SDK.

To replace the background of a live stream with a custom image, use the selfie segmentation model with MediaPipe Image Segmenter. This is a machine-learning model that identifies which pixels in the video frame are in the foreground or background. You can then use the results from the model to replace the background of a live stream, by copying foreground pixels from the video feed to a custom image representing the new background.

To integrate background replacement with the IVS real-time streaming Web broadcast SDK, you need to:

1. Install MediaPipe and Webpack. (Our example uses Webpack as the bundler, but you can use any bundler of your choice.)
2. Create `index.html`.
3. Add media elements.
4. Add a script tag.
5. Create `app.js`.
6. Load a custom background image.
7. Create an instance of `ImageSegmenter`.
8. Render the video feed to a canvas.
9. Create background replacement logic.
10. Create Webpack config File.
11. Bundle Your JavaScript file.

Install MediaPipe and Webpack

To start, install the `@mediapipe/tasks-vision` and `webpack` npm packages. The example below uses Webpack as a JavaScript bundler; you can use a different bundler if preferred.

JavaScript

```
npm i @mediapipe/tasks-vision webpack webpack-cli
```

Make sure to also update your `package.json` to specify `webpack` as your build script:

JavaScript

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
```

Create index.html

Next, create the HTML boilerplate and import the Web broadcast SDK as a script tag. In the following code, be sure to replace `<SDK version>` with the broadcast SDK version that you are using.

JavaScript

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Import the SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>

</body>
```

```
</html>
```

Add Media Elements

Next, add a video element and two canvas elements within the body tag. The video element will contain your live camera feed and will be used as input to the MediaPipe Image Segmenter. The first canvas element will be used to render a preview of the feed that will be broadcast. The second canvas element will be used to render the custom image that will be used as a background. Since the second canvas with the custom image is used only as a source to programmatically copy pixels from it to the final canvas, it is hidden from view.

JavaScript

```
<div class="row local-container">
      <video id="webcam" autoplay style="display: none"></video>
    </div>
    <div class="row local-container">
      <canvas id="canvas" width="640px" height="480px"></canvas>

      <div class="column" id="local-media"></div>
      <div class="static-controls hidden" id="local-controls">
        <button class="button" id="mic-control">Mute Mic</button>
        <button class="button" id="camera-control">Mute Camera</button>
      </div>
    </div>
    <div class="row local-container">
      <canvas id="background" width="640px" height="480px" style="display: none"></
canvas>
    </div>
```

Add a Script Tag

Add a script tag to load a bundled JavaScript file that will contain the code to do the background replacement and publish it to a stage:

```
<script src="./dist/bundle.js"></script>
```

Create app.js

Next, create a JavaScript file to get the element objects for the canvas and video elements that were created in the HTML page. Import the `ImageSegmenter` and `FilesetResolver` modules. The `ImageSegmenter` module will be used to perform the segmentation task.

JavaScript

```
const canvasElement = document.getElementById("canvas");
const background = document.getElementById("background");
const canvasCtx = canvasElement.getContext("2d");
const backgroundCtx = background.getContext("2d");
const video = document.getElementById("webcam");

import { ImageSegmenter, FilesetResolver } from "@mediapipe/tasks-vision";
```

Next, create a function called `init()` to retrieve the MediaStream from the user's camera and invoke a callback function each time a camera frame finishes loading. Add event listeners for the buttons to join and leave a stage.

Note that when joining a stage, we pass in a variable named `segmentationStream`. This is a video stream that is captured from a canvas element, containing a foreground image overlaid on the custom image representing the background. Later, this custom stream will be used to create an instance of a `LocalStageStream`, which can be published to a stage.

JavaScript

```
const init = async () => {
  await initializeDeviceSelect();

  cameraButton.addEventListener("click", () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? "Show Camera" : "Hide Camera";
  });

  micButton.addEventListener("click", () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? "Unmute Mic" : "Mute Mic";
  });

  localCamera = await getCamera(videoDevicesList.value);
  const segmentationStream = canvasElement.captureStream();

  joinButton.addEventListener("click", () => {
    joinStage(segmentationStream);
  });
```

```
  leaveButton.addEventListener("click", () => {
    leaveStage();
  });
};
```

Load a Custom Background Image

At the bottom of the `init` function, add code to call a function named `initBackgroundCanvas`, which loads a custom image from a local file and renders it onto a canvas. We will define this function in the next step. Assign the `MediaStream` retrieved from the user's camera to the video object. Later, this video object will be passed to the Image Segmenter. Also, set a function named `renderVideoToCanvas` as the callback function to invoke whenever a video frame has finished loading. We will define this function in a later step.

JavaScript

```
initBackgroundCanvas();

  video.srcObject = localCamera;
  video.addEventListener("loadeddata", renderVideoToCanvas);
```

Let's implement the `initBackgroundCanvas` function, which loads an image from a local file. In this example, we use an image of a beach as the custom background. The canvas containing the custom image will be hidden from display, as you will merge it with the foreground pixels from the canvas element containing the camera feed.

JavaScript

```
const initBackgroundCanvas = () => {
  let img = new Image();
  img.src = "beach.jpg";

  img.onload = () => {
    backgroundCtx.clearRect(0, 0, canvas.width, canvas.height);
    backgroundCtx.drawImage(img, 0, 0);
  };
};
```

Create an Instance of ImageSegmenter

Next, create an instance of `ImageSegmenter`, which will segment the image and return the result as a mask. When creating an instance of an `ImageSegmenter`, you will use the selfie segmentation model.

JavaScript

```
const createImageSegmenter = async () => {
  const audio = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/
@mediapipe/tasks-vision@0.10.2/wasm");

  imageSegmenter = await ImageSegmenter.createFromOptions(audio, {
    baseOptions: {
      modelAssetPath: "https://storage.googleapis.com/mediapipe-models/image_segmenter/
selfie_segmenter/float16/latest/selfie_segmenter.tflite",
      delegate: "GPU",
    },
    runningMode: "VIDEO",
    outputCategoryMask: true,
  });
};
```

Render the Video Feed to a Canvas

Next, create the function that renders the video feed to the other canvas element. We need to render the video feed to a canvas so we can extract the foreground pixels from it using the Canvas 2D API. While doing this, we also will pass a video frame to our instance of `ImageSegmenter`, using the segmentforVideo method to segment the foreground from the background in the video frame. When the segmentforVideo method returns, it invokes our custom callback function, `replaceBackground`, for doing the background replacement.

JavaScript

```
const renderVideoToCanvas = async () => {
  if (video.currentTime === lastWebcamTime) {
    window.requestAnimationFrame(renderVideoToCanvas);
    return;
  }
  lastWebcamTime = video.currentTime;
  canvasCtx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight);
```

```
  if (imageSegmenter === undefined) {
    return;
  }

  let startTimeMs = performance.now();

  imageSegmenter.segmentForVideo(video, startTimeMs, replaceBackground);
};
```

Create Background Replacement Logic

Create the `replaceBackground` function, which merges the custom background image with the foreground from the camera feed to replace the background. The function first retrieves the underlying pixel data of the custom background image and the video feed from the two canvas elements created earlier. It then iterates through the mask provided by `ImageSegmenter`, which indicates which pixels are in the foreground. As it iterates through the mask, it selectively copies pixels that contain the user's camera feed to the corresponding background pixel data. Once that is done, it converts the final pixel data with the foreground copied on to the background and draws it to a Canvas.

JavaScript

```
function replaceBackground(result) {
  let imageData = canvasCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  let backgroundData = backgroundCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  const mask = result.categoryMask.getAsFloat32Array();
  let j = 0;

  for (let i = 0; i < mask.length; ++i) {
    const maskVal = Math.round(mask[i] * 255.0);

    j += 4;
  // Only copy pixels on to the background image if the mask indicates they are in the
 foreground
    if (maskVal < 255) {
      backgroundData[j] = imageData[j];
      backgroundData[j + 1] = imageData[j + 1];
      backgroundData[j + 2] = imageData[j + 2];
      backgroundData[j + 3] = imageData[j + 3];
    }
```

```
  }

 // Convert the pixel data to a format suitable to be drawn to a canvas
  const uint8Array = new Uint8ClampedArray(backgroundData.buffer);
  const dataNew = new ImageData(uint8Array, video.videoWidth, video.videoHeight);
  canvasCtx.putImageData(dataNew, 0, 0);
  window.requestAnimationFrame(renderVideoToCanvas);
}
```

For reference, here is the complete `app.js` file containing all the logic above:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

// All helpers are expose on 'media-devices.js' and 'dom.js'
const { setupParticipant } = window;

const { Stage, LocalStageStream, SubscribeType, StageEvents, ConnectionState,
 StreamType } = IVSBroadcastClient;
const canvasElement = document.getElementById("canvas");
const background = document.getElementById("background");
const canvasCtx = canvasElement.getContext("2d");
const backgroundCtx = background.getContext("2d");
const video = document.getElementById("webcam");

import { ImageSegmenter, FilesetResolver } from "@mediapipe/tasks-vision";

let cameraButton = document.getElementById("camera-control");
let micButton = document.getElementById("mic-control");
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

let controls = document.getElementById("local-controls");
let audioDevicesList = document.getElementById("audio-devices");
let videoDevicesList = document.getElementById("video-devices");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
```

```
let cameraStageStream;
let micStageStream;
let imageSegmenter;
let lastWebcamTime = -1;

const init = async () => {
  await initializeDeviceSelect();

  cameraButton.addEventListener("click", () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? "Show Camera" : "Hide Camera";
  });

  micButton.addEventListener("click", () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? "Unmute Mic" : "Mute Mic";
  });

  localCamera = await getCamera(videoDevicesList.value);
  const segmentationStream = canvasElement.captureStream();

  joinButton.addEventListener("click", () => {
    joinStage(segmentationStream);
  });

  leaveButton.addEventListener("click", () => {
    leaveStage();
  });

  initBackgroundCanvas();

  video.srcObject = localCamera;
  video.addEventListener("loadeddata", renderVideoToCanvas);
};

const joinStage = async (segmentationStream) => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById("token").value;
```

```
if (!token) {
  window.alert("Please enter a participant token");
  joining = false;
  return;
}

// Retrieve the User Media currently set on the page
localMic = await getMic(audioDevicesList.value);

cameraStageStream = new LocalStageStream(segmentationStream.getVideoTracks()[0]);
micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

const strategy = {
  stageStreamsToPublish() {
    return [cameraStageStream, micStageStream];
  },
  shouldPublishParticipant() {
    return true;
  },
  shouldSubscribeToParticipant() {
    return SubscribeType.AUDIO_VIDEO;
  },
};

stage = new Stage(token, strategy);

// Other available events:
// https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
  connected = state === ConnectionState.CONNECTED;

  if (connected) {
    joining = false;
    controls.classList.remove("hidden");
  } else {
    controls.classList.add("hidden");
  }
});

stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
  console.log("Participant Joined:", participant);
});
```

```
  stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    console.log("Participant Media Added: ", participant, streams);

    let streamsToDisplay = streams;

    if (participant.isLocal) {
      // Ensure to exclude local audio streams, otherwise echo will occur
      streamsToDisplay = streams.filter((stream) => stream.streamType ===
 StreamType.VIDEO);
    }

    const videoEl = setupParticipant(participant);
    streamsToDisplay.forEach((stream) =>
 videoEl.srcObject.addTrack(stream.mediaStreamTrack));
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
    console.log("Participant Left: ", participant);
    teardownParticipant(participant);
  });

  try {
    await stage.join();
  } catch (err) {
    joining = false;
    connected = false;
    console.error(err.message);
  }
};

const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;

  cameraButton.innerText = "Hide Camera";
  micButton.innerText = "Mute Mic";
  controls.classList.add("hidden");
};

function replaceBackground(result) {
  let imageData = canvasCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
```

```
  let backgroundData = backgroundCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  const mask = result.categoryMask.getAsFloat32Array();
  let j = 0;

  for (let i = 0; i < mask.length; ++i) {
    const maskVal = Math.round(mask[i] * 255.0);

    j += 4;
    if (maskVal < 255) {
      backgroundData[j] = imageData[j];
      backgroundData[j + 1] = imageData[j + 1];
      backgroundData[j + 2] = imageData[j + 2];
      backgroundData[j + 3] = imageData[j + 3];
    }
  }
  const uint8Array = new Uint8ClampedArray(backgroundData.buffer);
  const dataNew = new ImageData(uint8Array, video.videoWidth, video.videoHeight);
  canvasCtx.putImageData(dataNew, 0, 0);
  window.requestAnimationFrame(renderVideoToCanvas);
}

const createImageSegmenter = async () => {
  const audio = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/
@mediapipe/tasks-vision@0.10.2/wasm");

  imageSegmenter = await ImageSegmenter.createFromOptions(audio, {
    baseOptions: {
      modelAssetPath: "https://storage.googleapis.com/mediapipe-models/image_segmenter/
selfie_segmenter/float16/latest/selfie_segmenter.tflite",
      delegate: "GPU",
    },
    runningMode: "VIDEO",
    outputCategoryMask: true,
  });
};

const renderVideoToCanvas = async () => {
  if (video.currentTime === lastWebcamTime) {
    window.requestAnimationFrame(renderVideoToCanvas);
    return;
  }
  lastWebcamTime = video.currentTime;
  canvasCtx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight);
```

```
  if (imageSegmenter === undefined) {
    return;
  }

  let startTimeMs = performance.now();

  imageSegmenter.segmentForVideo(video, startTimeMs, replaceBackground);
};

const initBackgroundCanvas = () => {
  let img = new Image();
  img.src = "beach.jpg";

  img.onload = () => {
    backgroundCtx.clearRect(0, 0, canvas.width, canvas.height);
    backgroundCtx.drawImage(img, 0, 0);
  };
};

createImageSegmenter();
init();
```

Create a Webpack Config File

Add this configuration to your Webpack config file to bundle `app.js`, so the import calls will work:

JavaScript

```
const path = require("path");
module.exports = {
  entry: ["./app.js"],
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist"),
  },
};
```

Bundle Your JavaScript files

```
npm run build
```

Start a simple HTTP server from the directory containing `index.html` and open `localhost:8000` to see the result:

```
python3 -m http.server -d ./
```

Android

To replace the background in your live stream, you can use the selfie segmentation API of Google ML Kit. The selfie segmentation API accepts a camera image as input and returns a mask that provides a confidence score for each pixel of the image, indicating whether it was in the foreground or the background. Based on the confidence score, you can then retrieve the corresponding pixel color from either the background image or the foreground image. This process continues until all confidence scores in the mask have been examined. The result is a new array of pixel colors containing foreground pixels combined with pixels from the background image.

To integrate background replacement with the IVS real-time streaming Android broadcast SDK, you need to:

1. Install CameraX libraries and the Google ML kit.
2. Initialize boilerplate variables.
3. Create a custom image source.
4. Manage camera frames.
5. Pass camera frames to Google ML Kit.
6. Overlay camera frame foreground onto your custom background.
7. Feed the new image to a custom image source.

Install CameraX Libraries and Google ML Kit

To extract images from the live camera feed, use Android's CameraX library. To install the CameraX library and Google ML Kit, add the following to your module's `build.gradle` file. Replace `${camerax_version}` and `${google_ml_kit_version}` with the latest version of the CameraX and Google ML Kit libraries, respectively.

Java

```
implementation "com.google.mlkit:segmentation-selfie:${google_ml_kit_version}"
implementation "androidx.camera:camera-core:${camerax_version}"
```

```
implementation "androidx.camera:camera-lifecycle:${camerax_version}"
```

Import the following libraries:

Java

```
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.ImageProxy
import androidx.camera.lifecycle.ProcessCameraProvider
import com.google.mlkit.vision.segmentation.selfie.SelfieSegmenterOptions
```

Initialize Boilerplate Variables

Initialize an instance of `ImageAnalysis` and an instance of an `ExecutorService`:

Java

```
private lateinit var binding: ActivityMainBinding
private lateinit var cameraExecutor: ExecutorService
private var analysisUseCase: ImageAnalysis? = null
```

Initialize a Segmenter instance in STREAM_MODE:

Java

```
private val options =
        SelfieSegmenterOptions.Builder()
            .setDetectorMode(SelfieSegmenterOptions.STREAM_MODE)
            .build()

private val segmenter = Segmentation.getClient(options)
```

Create a Custom Image Source

In the on`Create` method of your activity, create an instance of a `DeviceDiscovery` object and create a custom image source. The `Surface` provided by the Custom Image Source will receive the final image, with the foreground overlaid on a custom background image. You will then create an instance of a `ImageLocalStageStream` using the Custom Image Source. The instance of a `ImageLocalStageStream` (named `filterStream` in this example) can then be published to a

stage. See the IVS Android Broadcast SDK Guide for instructions on setting up a stage. Finally, also create a thread that will be used to manage the camera.

Java

```
var deviceDiscovery = DeviceDiscovery(applicationContext)
var customSource = deviceDiscovery.createImageInputSource( BroadcastConfiguration.Vec2(
720F, 1280F
))
var surface: Surface = customSource.inputSurface
var filterStream = ImageLocalStageStream(customSource)

cameraExecutor = Executors.newSingleThreadExecutor()
```

Manage Camera Frames

Next, create a function to initialize the camera. This function uses the CameraX library to extract images from the live camera feed. First, you create an instance of a `ProcessCameraProvider` called `cameraProviderFuture`. This object represents a future result of obtaining a camera provider. Then you load an image from your project as a bitmap. This example uses an image of a beach as a background, but it can be any image you want.

You then add a listener to `cameraProviderFuture`. This listener is notified when the camera becomes available or if an error occurs during the process of obtaining a camera provider.

Java

```
private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.beach
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;


        cameraProviderFuture.addListener({
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()


                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
 imageProxy.imageInfo.rotationDegrees)
```

```
                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA

            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)

            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

Within the listener, create `ImageAnalysis.Builder` to access each individual frame from the live camera feed. Set the back-pressure strategy to `STRATEGY_KEEP_ONLY_LATEST`. This guarantees that only one camera frame at a time is delivered for processing. Convert each individual camera frame to a bitmap, so you can extract its pixels to later combine it with the custom background image.

Java

```
val imageAnalyzer = ImageAnalysis.Builder()
```

```
analysisUseCase = imageAnalyzer
    .setTargetResolution(Size(360, 640))
    .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
    .build()

analysisUseCase?.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
    val mediaImage = imageProxy.image
    val tempBitmap = imageProxy.toBitmap();
    val inputBitmap = tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())
```

Pass Camera Frames to Google ML Kit

Next, create an `InputImage` and pass it to the instance of Segmenter for processing. An `InputImage` can be created from an `ImageProxy` provided by the instance of `ImageAnalysis`. Once an `InputImage` is provided to Segmenter, it returns a mask with confidence scores indicating the likelihood of a pixel being in the foreground or background. This mask also provides width and height properties, which you will use to create a new array containing the background pixels from the custom background image loaded earlier.

Java

```
if (mediaImage != null) {
        val inputImage =
            InputImage.fromMediaImag


segmenter.process(inputImage)
    .addOnSuccessListener { segmentationMask ->
        val mask = segmentationMask.buffer
        val maskWidth = segmentationMask.width
        val maskHeight = segmentationMask.height
        val backgroundPixels = IntArray(maskWidth * maskHeight)
        bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0, maskWidth, maskHeight)
```

Overlay the Camera Frame Foreground onto Your Custom Background

With the mask containing the confidence scores, the camera frame as a bitmap, and the color pixels from the custom background image, you have everything you need to overlay the foreground onto your custom background. The `overlayForeground` function is then called with the following parameters:

Java

```
resultBitmap = overlayForeground(mask, maskWidth, maskHeight, inputBitmap,
 backgroundPixels)
```

This function iterates through the mask and checks the confidence values to determine whether to get the corresponding pixel color from the background image or the camera frame. If the confidence value indicates that a pixel in the mask is most likely in the background, it will get the corresponding pixel color from the background image; otherwise, it will get the corresponding pixel color from the camera frame to build the foreground. Once the function finishes iterating through the mask, a new bitmap is created using the new array of color pixels and returned. This new bitmap contains the foreground overlaid on the custom background.

Java

```
private fun overlayForeground(
        byteBuffer: ByteBuffer,
        maskWidth: Int,
        maskHeight: Int,
        cameraBitmap: Bitmap,
        backgroundPixels: IntArray
    ): Bitmap {
        @ColorInt val colors = IntArray(maskWidth * maskHeight)
        val cameraPixels = IntArray(maskWidth * maskHeight)

        cameraBitmap.getPixels(cameraPixels, 0, maskWidth, 0, 0, maskWidth, maskHeight)

        for (i in 0 until maskWidth * maskHeight) {
            val backgroundLikelihood: Float = 1 - byteBuffer.getFloat()

            // Apply the virtual background to the color if it's not part of the
 foreground
            if (backgroundLikelihood > 0.9) {
                // Get the corresponding pixel color from the background image
                // Set the color in the mask based on the background image pixel color
                colors[i] = backgroundPixels.get(i)
            } else {
                // Get the corresponding pixel color from the camera frame
                // Set the color in the mask based on the camera image pixel color
                colors[i] = cameraPixels.get(i)
            }
        }
```

```
        return Bitmap.createBitmap(
            colors, maskWidth, maskHeight, Bitmap.Config.ARGB_8888
        )
    }
```

Feed the New Image to a Custom Image Source

You can then write the new bitmap to the `Surface` provided by a custom image source. This will broadcast it to your stage.

Java

```
resultBitmap = overlayForeground(mask, inputBitmap, mutableBitmap, bgBitmap)
canvas = surface.lockCanvas(null);
canvas.drawBitmap(resultBitmap, 0f, 0f, null)
```

Here is the complete function for getting the camera frames, passing it to Segmenter, and overlaying it on the background:

Java

```
@androidx.annotation.OptIn(androidx.camera.core.ExperimentalGetImage::class)
    private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.clouds
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;

        cameraProviderFuture.addListener({
            // Used to bind the lifecycle of cameras to the lifecycle owner
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

            val imageAnalyzer = ImageAnalysis.Builder()
            analysisUseCase = imageAnalyzer
                .setTargetResolution(Size(720, 1280))
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build()

            analysisUseCase!!.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
                val mediaImage = imageProxy.image
                val tempBitmap = imageProxy.toBitmap();
```

```
                val inputBitmap =
tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())

                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
imageProxy.imageInfo.rotationDegrees)

                    segmenter.process(inputImage)
                        .addOnSuccessListener { segmentationMask ->
                            val mask = segmentationMask.buffer
                            val maskWidth = segmentationMask.width
                            val maskHeight = segmentationMask.height
                            val backgroundPixels = IntArray(maskWidth * maskHeight)
                            bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0,
maskWidth, maskHeight)

                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA

            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)
```

```
            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

IVS Broadcast SDK: Mobile Audio Modes | Real-Time Streaming

Audio quality is an important part of any real-team media experience, and there isn't a one-size-fits-all audio configuration that works best for every use case. To ensure that your users have the best experience when listening to an IVS real-time stream, our mobile SDKs provide several preset audio configurations, as well as more powerful customizations as needed.

Introduction

The IVS mobile broadcast SDKs provide a `StageAudioManager` class. This class is designed to be the single point of contact for controlling the underlying audio modes on both platforms. On Android, this controls the AudioManager, including the audio mode, audio source, content type, usage, and communication devices. On iOS, it controls the application AVAudioSession, as well as whether voiceProcessing is enabled.

Important: Do not interact with `AVAudioSession` or `AudioManager` directly while the IVS real-time broadcast SDK is active. Doing so could result in the loss of audio, or audio being recorded from or played back on the wrong device.

Before you create your first `DeviceDiscovery` or `Stage` object, the `StageAudioManager` class must be configured.

Android (Kotlin)

```
StageAudioManager.getInstance(context).setPreset(StageAudioManager.UseCasePreset.VIDEO_CHAT)
 The default value

val deviceDiscovery = DeviceDiscovery(context)
val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
IVSStageAudioManager.sharedInstance().setPreset(.videoChat) // The default value

let deviceDiscovery = IVSDeviceDiscovery()
let stage = try? IVSStage(token: token, strategy: self)

// Other Stage implementation code
```

If nothing is set on the `StageAudioManager` before initialization of a `DeviceDiscovery` or `Stage` instance, the `VideoChat` preset is applied automatically.

Audio Use Case Presets

The real-time broadcast SDK provides three presets, each tailored to common use cases, as described below. For each preset, we cover five key categories that differentiate the presets from each other.

The **Volume Rocker** category refers to the type of volume (media volume or call volume) that is used or changed via the physical volume rockers on the device. Note that this impacts volume when switching audio modes. For example, suppose the device volume is set to the maximum value while using the Video Chat preset. Switching to the Subscribe Only preset causes a different volume level from the operating system, which could lead to a significant volume change on the device.

Video Chat

This is the default preset, designed for when the local device is going to have a real-time conversation with other participants.

Known issue on iOS: Using this preset and not attaching a microphone causes audio to play through the earpiece instead of the device speaker. Use this preset only in combination with a microphone.

Category	Android	iOS
Echo Cancellation	Enabled	Enabled

Category	Android	iOS
Noise Suppression	Enabled	Enabled
Volume Rocker	Call Volume	Call Volume
Microphone Selection	Limited based on the OS. USB microphones may not be available .	Limited based on the OS. USB and Bluetooth microphones may not be available. Bluetooth headsets that handle both input and output together should work; e.g., AirPods.
Audio Output	Any output device should work.	Limited based on the OS. Wired headsets may not be available.
Audio Quality	Medium / Low. It will sound like a phone call, not like media playback.	Medium / Low. It will sound like a phone call, not like media playback.

Subscribe Only

This preset is designed for when you plan to subscribe to other publishing participants but not publish yourself. It focuses on audio quality and supporting all available output devices.

Category	Android	iOS
Echo Cancellation	Disabled	Disabled
Noise Suppression	Disabled	Disabled
Volume Rocker	Media Volume	Media Volume
Microphone Selection	N/A, this preset is not designed for publishing.	N/A, this preset is not designed for publishing.
Audio Output	Any output device should work.	Any output device should work.

Category	Android	iOS
Audio Quality	High. Any media type should come through clearly, including music.	High. Any media type should come through clearly, including music.

Studio

This preset is designed for high quality subscribing while maintaining the ability to publish. It requires the recording and playback hardware to provide echo cancellation. A use case here would be using a USB microphone and a wired headset. The SDK will maintain the highest quality audio while relying on the physical separation of those devices from causing echo.

Category	Android	iOS
Echo Cancellation	Disabled	Disabled
Noise Supression	Disabled	Disabled
Volume Rocker	Media Volume in most cases. Call Volume when a Bluetooth microphone is connected.	Media Volume
Microphone Selection	Any microphone should work.	Any microphone should work.
Audio Output	Any output device should work.	Any output device should work.
Audio Quality	High. Both sides should be able to send music and hear it clearly on the other side. When a Bluetooth headset is connected, audio quality will drop due to Bluetooth SCO mode being enabled.	High. Both sides should be able to send music and hear it clearly on the other side. When a Bluetooth headset is connected, audio quality may drop due to Bluetooth SCO mode being enabled, depending on the headset.

Advanced Use Cases

Beyond the presets, both the iOS and Android real-time streaming broadcast SDKs allow configuring the underlying platform audio modes:

- On Android, set the AudioSource, Usage, and ContentType.
- On iOS, use AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode, and the ability to toggle if voice processing is enabled or not while publishing.

Note: When using these audio SDK methods, it is possible to incorrectly configure the underlying audio session. For example, using the `.allowBluetooth` option on iOS in combination with the `.playback` category creates an invalid audio configuration and the SDK cannot record or play back audio. These methods are designed to be used only when an application has specific audio-session requirements that have been validated.

Android (Kotlin)

```
// This would act similar to the Subscribe Only preset, but it uses a different
 ContentType.
StageAudioManager.getInstance(context)
    .setConfiguration(StageAudioManager.Source.GENERIC,
                      StageAudioManager.ContentType.MOVIE,
                      StageAudioManager.Usage.MEDIA);

val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
// This would act similar to the Subscribe Only preset, but it uses a different mode
 and options.
IVSStageAudioManager.sharedInstance()
    .setCategory(.playback,
                 options: [.duckOthers, .mixWithOthers],
                 mode: .default)

let stage = try? IVSStage(token: token, strategy: self)
```

```
// Other Stage implementation code
```

iOS Echo Cancellation

Echo cancellation on iOS can be independently controlled via `IVSStageAudioManager` as well using its `echoCancellationEnabled` method. This method controls whether voice processing is enabled on the input and output nodes of the underlying `AVAudioEngine` used by the SDK. It is important to understand the effect of changing this property manually:

- The `AVAudioEngine` property is honored only if the SDK's microphone is active; this is necessary due to the iOS requirement that voice processing be enabled on both the input and output nodes simultaneously. Normally this is done by using the microphone returned by `IVSDeviceDiscovery` to create an `IVSLocalStageStream` to publish. Alternately, the microphone can be enabled, without being used to publish, by attaching an `IVSAudioDeviceStatsCallback` to the microphone itself. This alternate approach is useful if echo cancellation is needed while using a custom audio-source-based microphone instead of the IVS SDK's microphone.
- Enabling the `AVAudioEngine` property requires a mode of `.videoChat` or `.voiceChat`. Requesting a different mode causes iOS's underlying audio framework to fight the SDK, causing audio loss.
- Enabling `AVAudioEngine` automatically enables the `.allowBluetooth` option.

Behaviors can differ depending on the device and iOS version.

iOS Custom Audio Sources

Custom audio sources can be used with the SDK by using `IVSDeviceDiscovery.createAudioSource`. When connecting to a Stage, the IVS real-time streaming broadcast SDK still manages an internal `AVAudioEngine` instance for audio playback, even if the SDK's microphone is not used. As a result, the values provided to `IVSStageAudioManager` must be compatible with the audio being provided by the custom audio source.

If the custom audio source being used to publish is recording from the microphone but managed by the host application, the echo-cancellation SDK above will not work unless the SDK-managed microphone is activated. To work around that requirement, see iOS Echo Cancellation.

Publishing with Bluetooth on Android

The SDK automatically reverts to the `VIDEO_CHAT` preset on Android when the following conditions are met:

- The assigned configuration does not use the `VOICE_COMMUNICATION` usage value.
- A Bluetooth microphone is connected to the device.
- The local participant is publishing to a Stage.

This is a limitation of the Android operating system in regard to how Bluetooth headsets are used for recording audio.

Integrating with Other SDKs

Because both iOS and Android support only one active audio mode per application, it is common to run into conflicts if your application uses multiple SDKs that require control of the audio mode. When you run into these conflicts, there are some common resolution strategies to try, explained below.

Match Audio Mode Values

Using either the IVS SDK's advanced audio-configuration options or the other SDK's functionality, have the two SDKs align on the underlying values.

Agora

iOS

On iOS, telling the Agora SDK to keep the `AVAudioSession` active will prevent it from deactivating while the IVS real-time streaming broadcast SDK is using it.

```
myRtcEngine.SetParameters("{\"che.audio.keep.audiosession\":true}");
```

Android

Avoid calling `setEnableSpeakerphone` on `RtcEngine`, and call `enableLocalAudio(false)` while publishing with the IVS real-time streaming broadcast SDK. You can call `enableLocalAudio(true)` again when the IVS SDK is not publishing.

Using Amazon EventBridge with IVS Real-Time Streaming

You can use Amazon EventBridge to monitor your Amazon Interactive Video Service (IVS) streams.

Amazon IVS sends change events about the status of your streams to Amazon EventBridge. All events that are delivered are valid. However, events are sent on a best-effort basis, which means there is no guarantee that:

- Events are delivered — A designated event can occur (e.g., a participant published) but it is possible that Amazon IVS will not send a corresponding event to EventBridge. Amazon IVS tries to deliver events for several hours before giving up.
- Events that are delivered will arrive in a specified timeframe — You may receive events up to a few hours old.
- Events are delivered in order — Events may be out of order, especially if they are sent within a short time of each other. For example, you could see Participant Unpublished before Participant Published.

While it's rare for events to be missing, late, or out of sequence, you should handle these possibilities if you write business-critical programs that depend on the order or existence of notification events.

You can create EventBridge rules for any of the following events.

Event Type	Event	Sent When ...
IVS Composition State Change	Destination Failure	An attempt to output to a Destination failed (e.g., the S3 bucket was not found, access was denied to the S3 bucket, or the stream already exists for an RTMP destination).
IVS Composition State Change	Destination Start	Output to a Destination successfully started.
IVS Composition State Change	Destination End	Output to a Destination finished.

Event Type	Event	Sent When ...
IVS Composition State Change	Destination Reconnecting	Output to a Destination was interrupted and a reconnect is being attempted.
IVS Composition State Change	Session Start	A Composition session was created. This event fires when a Composition process pipeline successfully initializes. At this point, the Composition pipeline has successfully subscribed to a Stage and is receiving media and able to compose video.
IVS Composition State Change	Session End	A Composition session completed.
IVS Composition State Change	Session Failure	A Composition pipeline failed due to a stage being deleted, one or more outputs failing, or any other internal error.
IVS Participant Recording State Change	Recording Start	A publisher has connected to the stage and is being recorded to S3.
IVS Participant Recording State Change	Recording End	A publisher has disconnected from the stage and all remaining files have been written to S3.
IVS Participant Recording State Change	Recording Start Failure	A publisher connects to the stage, but recording fails to start due to errors (e.g., if an S3 bucket is not found or cannot be accessed). This publisher's live stream is not recorded.

Event Type	Event	Sent When ...
IVS Participant Recording State Change	Recording End Failure	Recording ends with failure, due to errors encountered during recording (e.g., if an S3 bucket is not found or cannot be accessed). Some objects may still be written to the configured storage location.
IVS Stage Update	Participant Published	A participant begins publishing to a stage.
IVS Stage Update	Participant Unpublished	A participant has stopped publishing to a stage.
IVS Stage Update	Participant Publish Error	A participant's attempt to publish to a stage failed.
IVS Stage Update	Participant Replication Start	A participant replication starts.
IVS Stage Update	Participant Replication End	A participant replication ends. A replication can end due to a StopParticipantReplication API operation, if the publisher has stopped publishing, or if the publisher has stopped publishing and the reconnect window has expired.
IVS Stage Update	Token Exchanged	An existing participant token is exchanged for a new one. This exchange results in upgraded or downgraded token capabilities and/or updated token attributes.

Creating Amazon EventBridge Rules for Amazon IVS

You can create a rule that triggers on an event emitted by Amazon IVS. Follow the steps in Create a rule in Amazon EventBridge in the *Amazon EventBridge User Guide*. When selecting a service, choose **Interactive Video Service (IVS)**.

Examples: Composition State Change

Destination Failure: This event is sent when an attempt to output to a Destination failed (e.g., the S3 bucket was not found, access was denied to the S3 bucket, or the stream already exists for an RTMP destination.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Failure",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
     "error_code": "e.g., AccessDeniedException",
     "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

The following table lists `error_code` and `reason` values for Destination Failure events, along with troubleshooting guidance:

error_code	reason	Troubleshooting Guidance
ResourceNotFoundException	S3 bucket not found. Please	Verify your S3 bucket exists and is in the correct region.

error_code	reason	Troubleshooting Guidance
	verify your bucket exists.	
AccessDeniedException	Access denied to S3 bucket. Please verify your bucket policy.	Verify your S3 bucket policy grants IVS service the necessary permissio ns.
ConflictException	Stream already exists	Verify no other broadcast is active on the same RTMP destination channel.
InternalServerException	Service internal error	Retry the operation. If the issue persists, contact AWS Support.

Destination Start: This event is sent when output to a Destination successfully started.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Start",
     "stage_arn": "<stage-arn>",
     "id": "<destination-id>",
   }
}
```

Destination End: This event is sent when output to a Destination finished.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination End",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
   }
}
```

Destination Reconnecting: This event is sent when output to a Destination was interrupted and a reconnect is being attempted.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Reconnecting",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
   }
}
```

Session Start: This event is sent when a Composition session was created. This event fires when a Composition process pipeline successfully initializes. At this point, the Composition pipeline has successfully subscribed to a Stage and is receiving media and able to compose video.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Start",
     "stage_arn": "<stage-arn>"
   }
}
```

Session End: This event is sent when a Composition session completed and all resources were deleted.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session End",
     "stage_arn": "<stage-arn>"
   }
}
```

Session Failure: This event is sent when a Composition pipeline failed due to a stage being deleted, one or more outputs failing, or any other internal error.

```
{
   "version": "0",
```

```
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Failure",
     "stage_arn": "<stage-arn>",
     "error_code": "e.g., DestinationFailure",
     "reason": "e.g. One or more outputs failed"
   }
}
```

The following table lists `error_code` and `reason` values for Session Failure events, along with troubleshooting guidance:

error_code	reason	Troubleshooting Guidance
StageDeleted	Stage has been deleted	Verify the stage exists before starting a composition.
DestinationFailure	One or more outputs failed	Check individual destination errors.
InternalServerException	Service internal error	Retry the operation. If the issue persists, contact AWS Support.

Examples: Individual Participant Recording State Change

Recording Start: This event is sent when a publisher has connected to the stage and is being recorded to S3.

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
```

```
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:09:58Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
   }
}
```

Recording End: This event is sent when a publisher has disconnected from the stage and all remaining files have been written to S3.

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording End",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "recording_duration_ms": 547327
   }
}
```

Recording Start Failure: This event is sent when a publisher connects to the stage, but recording fails to start due to errors (e.g., if an S3 bucket is not found or cannot be accessed). The publisher's live stream is not recorded.

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:09:58Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start Failure",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "error_code": "e.g., AccessDeniedException",
      "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

The following table lists `error_code` and `reason` values for Recording Start Failure events, along with troubleshooting guidance:

error_code	reason	Troubleshooting Guidance
ResourceNotFoundException	S3 bucket not found. Please verify your bucket exists.	Verify your S3 bucket exists and is in the correct region.
AccessDeniedException	Access denied to S3 bucket. Please verify your bucket policy.	Verify your S3 bucket policy grants IVS service the necessary permissio ns.
ValidationException	Video codec not supported for recording	Verify the publisher is using a supported video codec.

error_code	reason	Troubleshooting Guidance
InternalServerException	Service internal error	Retry the operation. If the issue persists, contact AWS Support.

Recording End Failure: This event is sent when the recording ends with failure, due to errors encountered during recording (e.g., if an S3 bucket is not found or cannot be accessed). Some objects may still be written to the configured storage location.

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording End Failure",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "recording_duration_ms": 547327,
      "error_code": "e.g., AccessDeniedException",
      "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

The following table lists `error_code` and `reason` values for Recording End Failure events, along with troubleshooting guidance:

error_code	reason	Troubleshooting Guidance
ResourceNotFoundException	S3 bucket not found. Please	Verify your S3 bucket exists and is in the correct region.

error_code	reason	Troubleshooting Guidance
	verify your bucket exists.	
AccessDeniedException	Access denied to S3 bucket. Please verify your bucket policy.	Verify your S3 bucket policy grants IVS service the necessary permissio ns.
InternalServerException	Service internal error	Retry the operation. If the issue persists, contact AWS Support.

Note that, if individual participant recording merge is enabled, and if a stage publisher disconnects from a stage and then reconnects, IVS tries to record to the same S3 prefix as the previous session. As a consequence, in the above examples, the `session_id` component of `recording_s3_key_prefix` can have a different value than the `session_id` field in `detail`. See Merge Fragmented Individual Participant Recordings.

Examples: Stage Update

Stage update events include an event name (which classifies the event) and metadata about the event. The metadata includes the participant ID which triggered the event, the associated stage and session IDs, and the user ID.

Participant Published: This event is sent when a participant begins publishing to a stage.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
```

```
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Participant Published",
        "event_time": "2025-11-18T16:40:32Z",
        "event_time_precise": "2025-11-18T16:40:32.123456Z",
 "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```

Participant Unpublished: This event is sent when a participant has stopped publishing to a stage.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Participant Unpublished",
 "event_time": "2025-11-18T16:40:32Z",
 "event_time_precise": "2025-11-18T16:40:32.123456Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```

Participant Publish Error: This event is sent when a participant's attempt to publish to a stage failed.

```
{
```

```
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Participant Publish Error",
        "event_time": "2024-08-13T14:38:17.089061676Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "error_code": "BITRATE_EXCEEDED",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```


Participant Replication Start: This event is sent when a participant replication starts.


```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Participant Replication Start",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
```

```
        "destination_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/
XYZdef1G2hij",
        "destination_session_id":  "aBC1c2d3e4f"
    }
}
```

Participant Replication End: This event is sent when a participant replication ends. A replication can end due to a StopParticipantReplication API operation, if the publisher has stopped publishing, or if the publisher has stopped publishing and the reconnect window has expired.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Participant Replication End",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "destination_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/
XYZdef1G2hij",
        "destination_session_id":  "aBC1c2d3e4f"
    }
}
```

Token Exchanged: This event is sent when an existing participant token is exchanged for a new one, resulting in upgraded or downgraded token capabilities and/or updated token attributes.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
```

```
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-ZyXwvu1T2s",
        "event_name": "Token Exchanged",
        "event_time": "2025-11-12T20:54:53Z",
        "user_id": "UpdatedUser",
        "participant_id": "xYz1c2d3e4f",
        "previous_token": {
            "capabilities": ["SUBSCRIBE"],
            "attributes": {
                "role": "viewer"
            },
            "user_id": "InitialUser",
            "expiration_time": "2025-11-12T21:54:52Z"
        },
        "new_token": {
            "capabilities": ["SUBSCRIBE", "PUBLISH"],
            "attributes": {
                "role": "moderator"
            },
            "user_id": "UpdatedUser",
            "expiration_time": "2025-11-12T22:54:52Z"
        }
    }
}
```

IVS Server-Side Composition | Real-Time Streaming

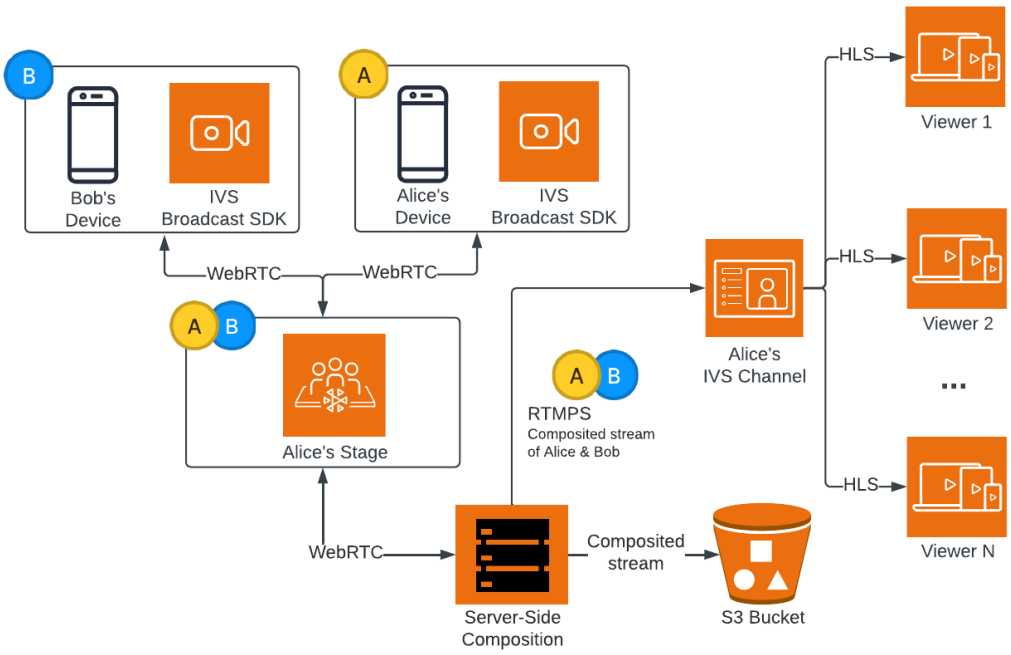
Server-side composition uses an IVS server to mix audio and video from all stage participants and then sends this mixed video to an IVS channel (e.g., to reach a larger audience) or an S3 bucket. Server-side composition is invoked through IVS control-plane operations in the stage's home region.

Broadcasting or recording a stage using server-side composition offers numerous benefits, making it an attractive choice for users seeking efficient and reliable cloud-based video workflows.

Topics

Overview of IVS Server-Side Composition

This diagram illustrates how server-side composition works:



Benefits

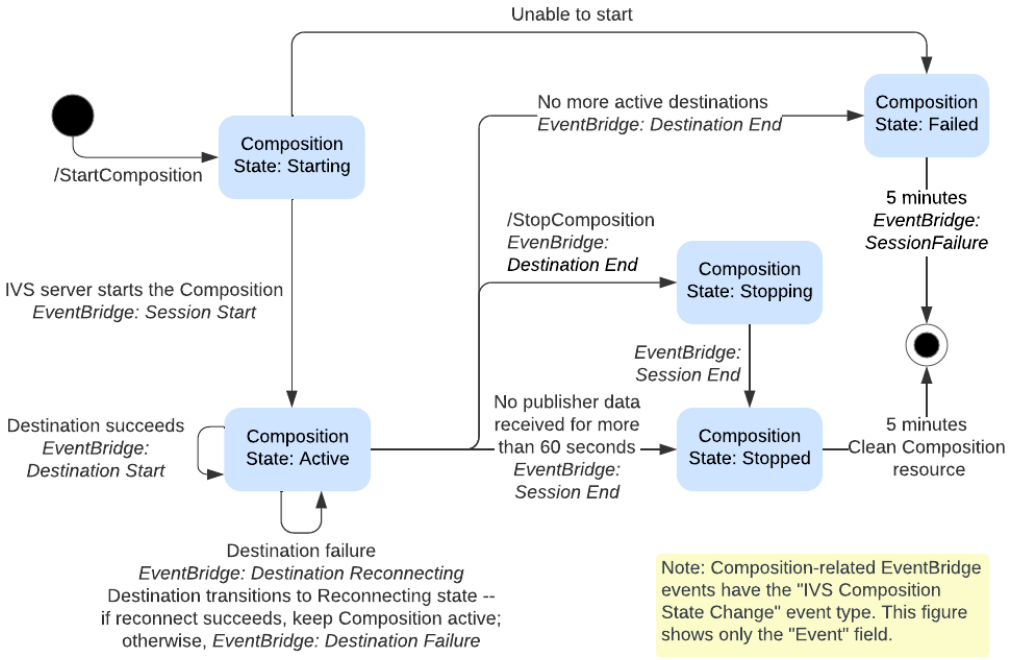
Compared to client-side composition, server-side composition has the following benefits:

- **Reduced client load** — With server-side composition, the burden of processing and combining audio and video sources is shifted from individual client devices to the server itself. Server-side composition eliminates the need for client devices to use their CPU and network resources for compositing the view and transmitting it to IVS. This means viewers can watch the broadcast without their devices having to handle resource-intensive tasks, which can lead to improved battery life and smoother viewing experiences.
- **Consistent quality** — Server-side composition allows for precise control over the quality, resolution, and bitrate of the final stream. This ensures a consistent viewing experience for all viewers, regardless of their individual devices' capabilities.
- **Resilience** — By centralizing the composition process on the server, the broadcast becomes more robust. Even if a publisher device experiences technical limitations or fluctuations, the server can adapt and provide a smoother stream to all audience members.
- **Bandwidth efficiency** — Since the server handles the composition, stage publishers do not have to spend extra bandwidth broadcasting the video to IVS.

Alternatively, to broadcast a stage to an IVS channel, you can do the composition client side; see Enabling Multiple Hosts on an IVS Stream in the *IVS Low-Latency Streaming User Guide*.

Composition Lifecycle

Use the diagram below to understand the state transitions of a composition:



At a high level, the life cycle of a Composition is as follows:

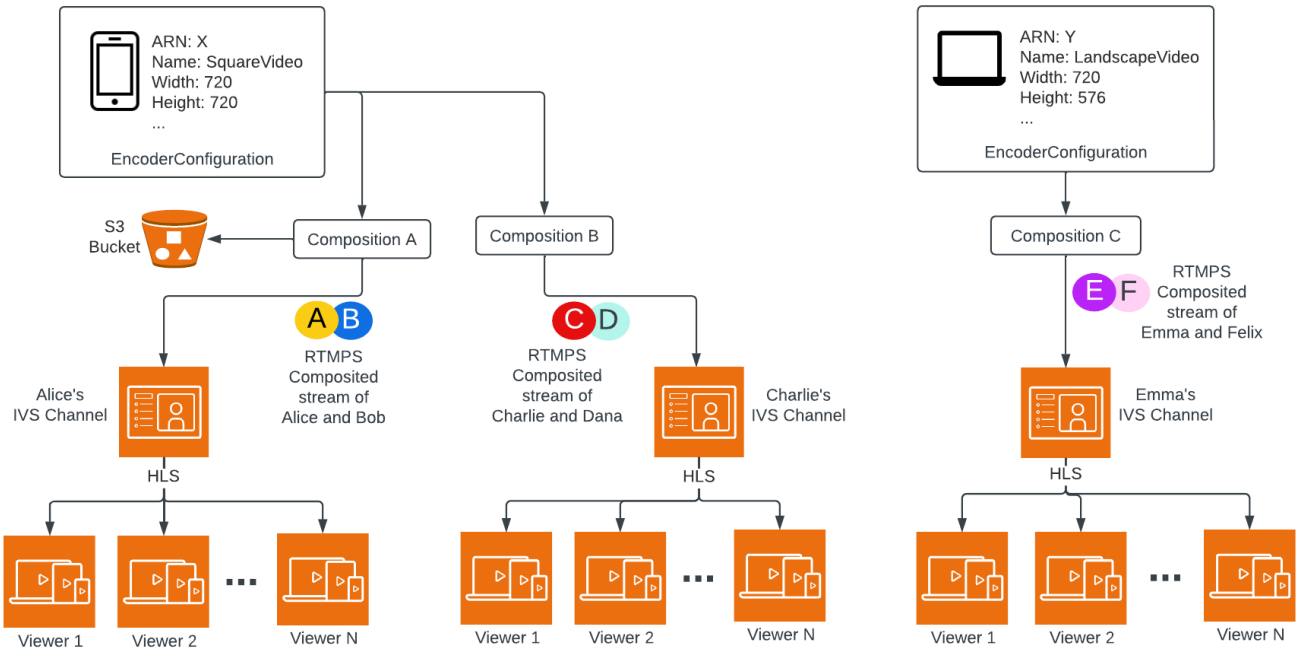
1. A Composition resource is created when the user calls the StartComposition operation.
2. Once IVS successfully starts the Composition, an “IVS Composition State Change (Session Start)” EventBridge event is sent. See Using EventBridge with IVS Real-Time Streaming for details about events.
3. Once a Composition is in an active state, the following can happen:
   - User stops the Composition — If the StopComposition operation is called, IVS initiates a graceful shutdown of the Composition, sending "Destination End" events followed by a "Session End" event.
   - Composition performs auto-shutdown — If the IVS stage is deleted or no participant is actively publishing to the IVS stage for 60 seconds, the Composition is finalized automatically and EventBridge events are sent.
   - Destination failure — If a destination unexpectedly fails (e.g., the IVS channel gets deleted), the destination transitions to the `RECONNECTING` state and a “Destination Reconnecting” event is sent. If recovery is impossible, IVS transitions the destination to the `FAILED` state and a “Destination Failure” event is sent. IVS keeps the composition alive if at least one of its destinations is active.
4. Once the composition is in the `STOPPED` or `FAILED` state, it is automatically cleaned up after five minutes. (Then it no longer is retrieved by ListCompositions or GetComposition.)

IVS API

Server-side composition uses these key API elements:

- An *EncoderConfiguration* object allows you to customize the format of the video to be generated (height, width, bitrate, and other streaming parameters). You can reuse an EncoderConfiguration every time you call the StartComposition operation.
- *Composition* operations track the video composition and output to an IVS channel.
- *StorageConfiguration* tracks the S3 bucket where compositions are recorded.

To use server-side composition, you need to create an EncoderConfiguration and attach it when calling the StartComposition operation. In this example, the SquareVideo EncoderConfiguration is used in two Compositions:



For complete information, see IVS Real-Time Streaming API Reference.

Layouts

The StartComposition operation offers two layout options: grid and PiP (Picture-in-Picture).

Server-side composition responds to token exchange events in real time. When a participant exchanges a token to update attributes such as featured status or PiP assignment, the composition layout updates automatically without requiring the participant to leave and rejoin the stage.

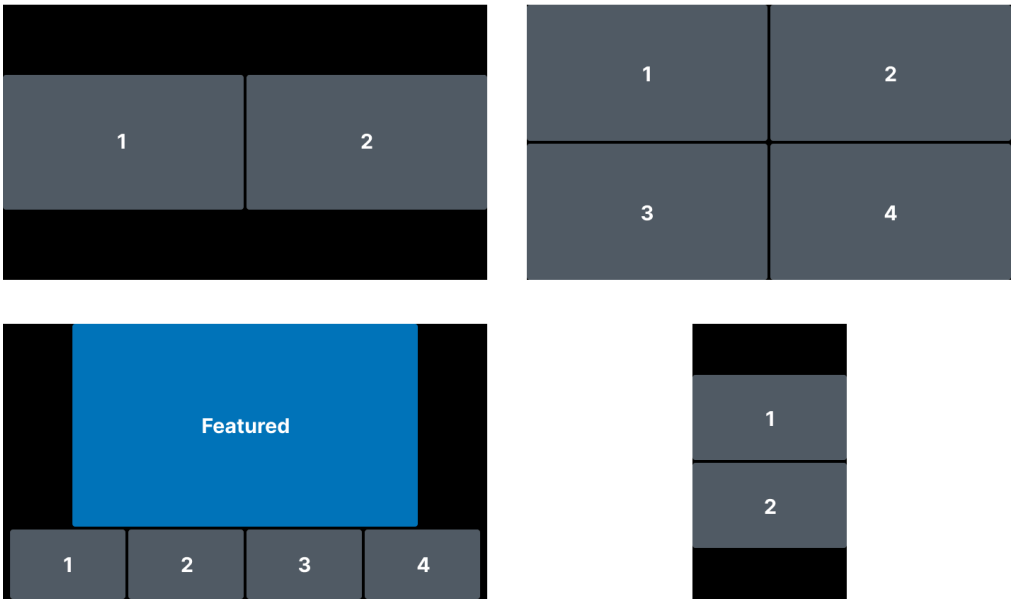
You can also use token exchange to change participant ordering during a live composition. When a participant exchanges their token with an updated order attribute value, the composition re-renders automatically to reflect the new ordering. This eliminates the need for the participant to disconnect and reconnect. For more information, see Token Exchange.

Grid Layout

The grid layout arranges stage participants in a grid of equally sized slots. It provides several customizable properties:

- `videoAspectRatio` sets the participant display mode to control the aspect ratio of video tiles.
- `videoFillMode` defines how video content fits within the participant tile.
- `gridGap` specifies the spacing between participant tiles in pixels.
- `omitStoppedVideo` allows excluding stopped video streams from the composition.
- `featuredParticipantAttribute` identifies the featured slot. When this is set, the featured participant is displayed in a larger slot on the main screen, with other participants shown below it.
- `participantOrderAttribute` enables custom participant ordering based on attribute values in participant tokens. When specified, participants are ordered numerically by their attribute values, with participants lacking the attribute falling back to arrival-time ordering. This provides optional deterministic positioning and allows for role-based layouts.

For details on grid layout (including valid values and defaults for all fields), see the GridConfiguration data type.



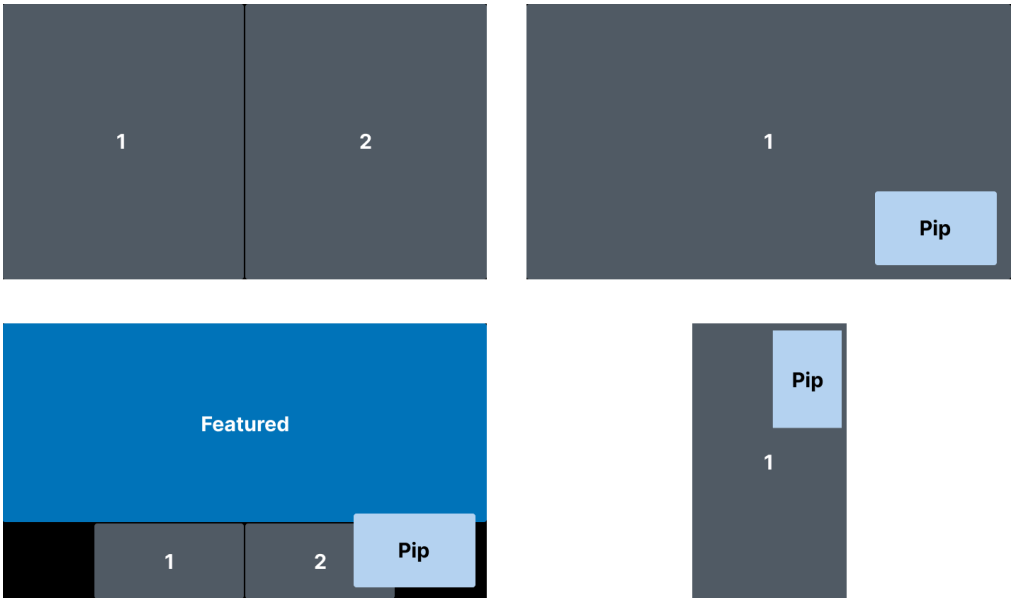
Picture-in-Picture (PiP) Layout

The PiP layout enables displaying a participant in an overlay window with configurable size, position, and behavior. Key properties include:

- `pipParticipantAttribute` specifies the participant for the PiP window.
- `pipPosition` determines the corner position of the PiP window.
- `pipWidth` and `pipHeight`configure the width and height of the PiP window.
- `pipOffset` sets the offset position of the PiP window in pixels from the closest edges.
- `pipBehavior` defines PiP behavior when all other participants have left.

Like the grid layout, the PiP layout supports `featuredParticipantAttribute`, `omitStoppedVideo`, `videoFillMode`, `gridGap`, and `participantOrderAttribute` to further customize the composition. The `participantOrderAttribute` enables custom participant ordering for both selecting the participant for the PiP window and positioning grid participants based on attribute values in participant tokens.

For details on PiP layout (including valid values and defaults for all fields), see the PipConfiguration data type.



Note: The maximum resolution supported by a stage publisher on server-side composition is 1080p. If a publisher sends video higher than 1080p, the publisher will be rendered as an audio-only participant.

Important: Ensure your application does not depend on the specific features of the current layout, such as size and position of tiles. *Visual improvements to layouts can be introduced at any time*.

Getting Started with IVS Server-Side Composition

This document takes you through the steps involved in getting started with IVS server-side composition.

Prerequisites

To use server-side composition, you must have a stage with active publishers and use an IVS channel and/or an S3 bucket as the composition destination.

To create an S3 bucket, see the S3 documentation on how to create buckets. The S3 bucket must be created in the same AWS region as the IVS stage.

Important: If you use an existing S3 bucket:

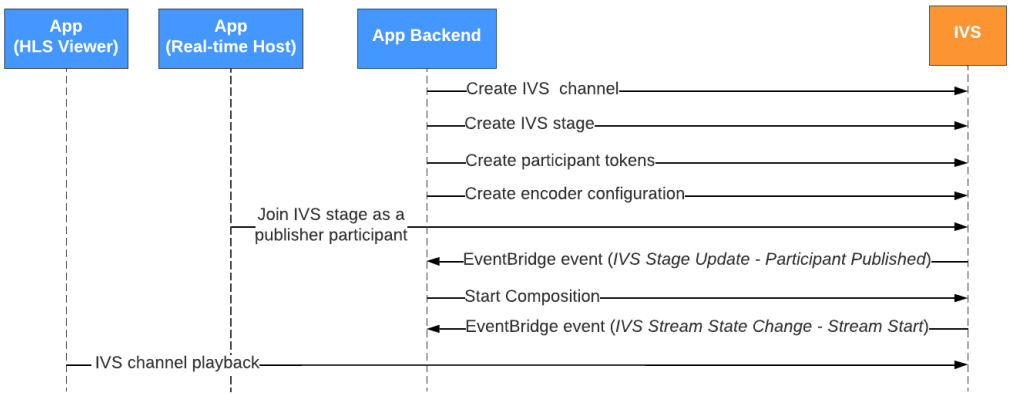
- The **Object Ownership** setting must be **Bucket owner enforced** or **Bucket owner preferred**.
- The **Default Encryption** setting must be **Server-side encryption with Amazon S3 managed keys (SSE-S3)**.

For details, see the S3 documentation on controlling ownership of objects and protecting data with encryption.

API Instructions

Below, we describe one possible workflow that uses EventBridge events to start a composition that broadcasts the stage to an IVS channel when a participant publishes. Alternatively, you can start and stop compositions based on your own app logic. See Composite Recording for another example which showcases the use of server-side composition to record a stage directly to an S3 bucket.

1. Create an IVS channel. See Getting Started with Amazon IVS Low-Latency Streaming.
2. Create an IVS stage and participant tokens for each publisher.
3. Create an EncoderConfiguration.
4. Join the stage and publish to it. (See the "Publishing and Subscribing" sections of the real-time streaming broadcast SDK guides: Web, Android, and iOS.)
5. When you receive a Participant Published EventBridge event, call StartComposition with your desired layout configuration.
6. Wait for a few seconds and see the composited view in the channel playback.



Note: A Composition performs auto-shutdown after 60 seconds of inactivity from publisher participants on the stage. At that point, the Composition is terminated and transitions to a STOPPED state. A Composition is automatically deleted after a few minutes in the STOPPED state.

CLI Instructions

Using the AWS CLI is an advanced option and requires that you first download and configure the CLI on your machine. For details, see the AWS Command Line Interface User Guide.

Now you can use the CLI to create and manage resources. The Composition operations are under the `ivs-realtime` namespace.

Create the EncoderConfiguration Resource

An EncoderConfiguration is an object that allows you to customize the format of the generated video (height, width, bitrate, and other streaming parameters). You can reuse an EncoderConfiguration every time you call the Composition operation, as explained in the next step.

The command below creates an EncoderConfiguration resource that configures server-side video composition parameters like video bitrate, frame rate and resolution:

```
aws ivs-realtime create-encoder-configuration --name "MyEncoderConfig" --video
 "bitrate=2500000,height=720,width=1280,framerate=30"
```

The response is:

```
{
"encoderConfiguration": {
  "arn": "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4",
  "name": "MyEncoderConfig",
  "tags": {},
  "video": {
  "bitrate": 2500000,
  "framerate": 30,
  "height": 720,
  "width": 1280
    }
  }
}
```

Start a Composition

Using the EncoderConfiguration ARN provided in the response above, create your Composition resource:

Grid Layout Example

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations '[{"channel":
 {"channelArn": "arn:aws:ivs:us-east-1:927810967299:channel/
```

```
DOlMW4dfMR8r", "encoderConfigurationArn": "arn:aws:ivs:us-
east-1:927810967299:encoder-configuration/9W59OBY2M8s4"}}]' --layout '{"grid":
{"participantOrderAttribute":"order","featuredParticipantAttribute":"isFeatured","videoFillMode
```

PiP Layout Example

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations '[{"channel":
 {"channelArn": "arn:aws:ivs:us-east-1:927810967299:channel/
DOlMW4dfMR8r", "encoderConfigurationArn": "arn:aws:ivs:us-
east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout '{"pip":
{"participantOrderAttribute":"priority","pipParticipantAttribute":"isPip","pipOffset":10,"pipPo
```

Note: You can use this tool to more easily generate the `--layout` configuration based on your layout choices.

The response will show that the Composition is created with a `STARTING` state. Once the Composition starts publishing the composition, the state transitions to `ACTIVE`. (You can see the state by calling the ListCompositions or GetComposition operation.)

Once a Composition is `ACTIVE`, the composite view of the IVS stage is visible on the IVS channel, using ListCompositions:

```
aws ivs-realtime list-compositions
```

The response is:

```
{
"compositions": [
  {
  "arn": "arn:aws:ivs:us-east-1:927810967299:composition/YVoaXkKdEdRP",
  "destinations": [
  {
     "id": "bD9rRoN91fHU",
     "startTime": "2023-09-21T15:38:39+00:00",
     "state": "ACTIVE"
  }
  ],
  "stageArn": "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
  "startTime": "2023-09-21T15:38:37+00:00",
  "state": "ACTIVE",
```

```
  "tags": {}
    }
  ]
}
```

Note: You need to have publisher participants actively publishing to the stage to keep the composition alive. For more information, see the "Publishing and Subscribing" sections of the real-time streaming broadcast SDK guides: Web, Android, and iOS. You must create a distinct stage token for each participant.

Custom Participant Ordering

Custom participant ordering allows you to control the positioning of participants in both grid and PiP layouts based on custom attribute values in participant tokens, including the positioning of the featured participants and selection of participants the PiP window. This provides deterministic participant positioning and enables role-based layouts.

How Custom Ordering Works

When `participantOrderAttribute` is specified in your layout configuration, participants are ordered according to the following rules:

- Participants with the specified ordering attribute in their tokens are positioned first, sorted numerically by their attribute values.
- Participants without the ordering attribute fall back to arrival-time ordering and are positioned after the ordered participants.
- When multiple participants have identical ordering values, they are sub-sorted by their arrival time into the stage.
- The ordering uses numeric sorting (not lexicographic), so "10" comes after "9" (not after "1").
- Negative values are supported. They are positioned before positive values.
- Non-numeric values (e.g., "abc", "1.5") are treated as invalid, and those participants fall back to arrival-time ordering.

Important: Participant ordering (whether based on the arrival time or custom) takes effect after the composition has started. The correct ordering of participants is not guaranteed for participants who join the stage before the composition begins.

Creating Tokens with Ordering Attributes

To use custom participant ordering, include the ordering attribute in your participant tokens when creating them:

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=1

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=2

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=3
```

You can combine the custom-participant-order attribute with the attributes for selecting participants for the featured slot and the PiP window:

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=2,isFeatured=true

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=3,isFeatured=true

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=4,isPip=true
```

Example Use Cases

Example use cases include:

- Consistent positioning – Participants maintain their positions when reconnecting with the same token.
- Role-based positioning – For instance, you could specify teachers with order=1 and students with order=2.
- Priority-based layouts – VIP participants with lower order values would appear first.
- Dynamic layouts – You can combine custom ordering with `featuredParticipantAttribute` and `pipParticipantAttribute` for complex scenarios.

- Cross-stage interactions – When using participant replication for scenarios like VS Mode competitions where streamers from different stages interact, you can override ordering attributes to control positioning in the destination stage composition.

Note: For participant-replication use cases, you can override participant attributes (including the order attribute) as needed when starting a replication to achieve the desired layout in the destination stage.

Backward Compatibility

Custom participant ordering is an optional feature and is fully backward compatible. Existing compositions without `participantOrderAttribute` continue to work unchanged, using arrival-time ordering. When `participantOrderAttribute` is set to an empty string, the system ignores custom ordering entirely and falls back to default behavior.

Enabling Screen Share in IVS Server-Side Composition

To use a fixed screen-share layout, follow the steps below.

Create the EncoderConfiguration Resource

The command below creates an EncoderConfiguration resource that configures server-side composition parameters (video bitrate, framerate, and resolution).

```
aws ivs-realtime create-encoder-configuration --name "test-ssc-with-screen-share" --
video={bitrate=2000000,framerate=30,height=720,width=1280}
```

Create a stage participant token with a `screen-share` attribute. Since we will specify `screen-share` as the name of the `featured` slot, we need to create a stage token with the `screen-share` attribute set to `true`:

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes screen-share=true
```

The response is:

```
{
   "participantToken": {
      "attributes": {
```

```
         "screen-share": "true"
      },
      "expirationTime": "2023-08-04T05:26:11+00:00",
      "participantId": "E813MFklPWLF",
      "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2OTExMjY3NzEsImlhdCI6MTY5MTA4MzU3MSwianRpIjoiRT
   }
}
```

Start the Composition

To start the composition using the screen-share feature, we use this command:

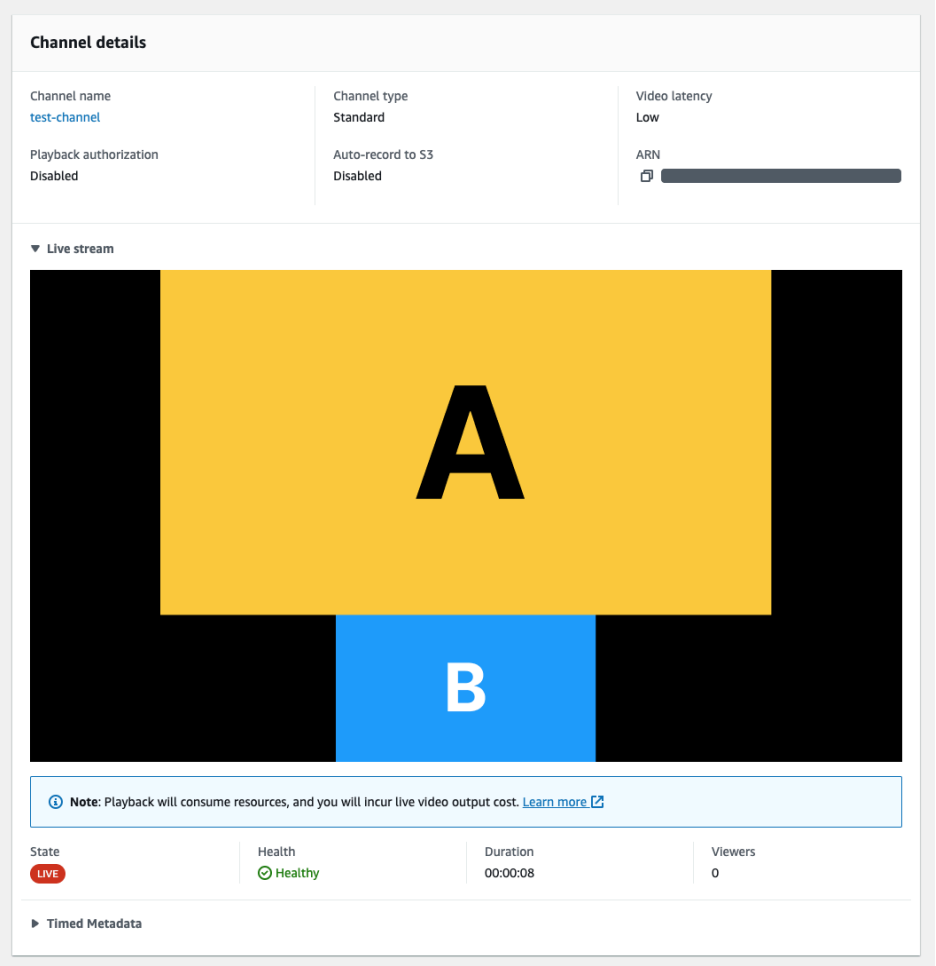
```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout
 '{"grid":{"featuredParticipantAttribute":"screen-share"}}'
```

The response is:

```
{
"composition" : {
"arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
"destinations" : [ {
 "configuration" : {
 "channel" : {
    "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r",
    "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-
configuration/DEkQHWPVaOwO"
 },
 "name" : ""
 },
 "id" : "SGmgBXTULuXv",
 "state" : "STARTING"
} ],
"layout" : {
 "grid" : {
 "featuredParticipantAttribute" : "screen-share",
 "gridGap": 2,
 "omitStoppedVideo": false,
 "videoAspectRatio": "VIDEO"
```

```
 }
},
"stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
"startTime" : "2023-09-27T21:32:38Z",
"state" : "STARTING",
"tags" : { }
}
}
```

When the stage participant `E813MFklPWLF` joins the stage, that participant's video will be displayed in the featured slot, and all other stage publishers will be rendered below the slot:



Stop the Composition

To stop a composition at any point, call the StopComposition operation:

```
aws ivs-realtime stop-composition --arn arn:aws:ivs:us-east-1:927810967299:composition/
B19tQcXRgtoz
```

Known Issues and Workarounds

This section lists known issues that you might encounter when using IVS server-side composition and suggests potential workarounds.

- Some compositions may exhibit brief audio stuttering after periods of silence.

  Workaround: None

IVS Recording | Real-Time Streaming

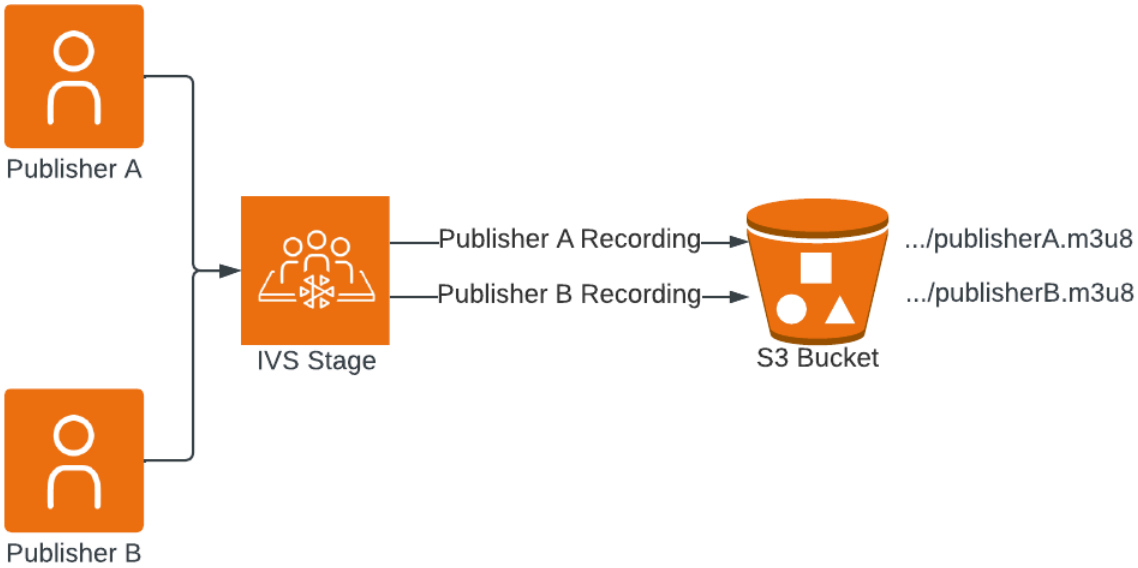
There are two recording options for IVS real-time streaming:

- With individual participant recording, each publisher's media is recorded in separate files.
- In contrast, composite recording combines media from all publishers into a single view and records it in one file.

Individual participant recording incurs no additional Amazon IVS charges, while composite recording incurs charges for the hourly rate for the video encoded. Both recording options incur standard S3 storage and request costs. For more details, see Amazon IVS pricing.

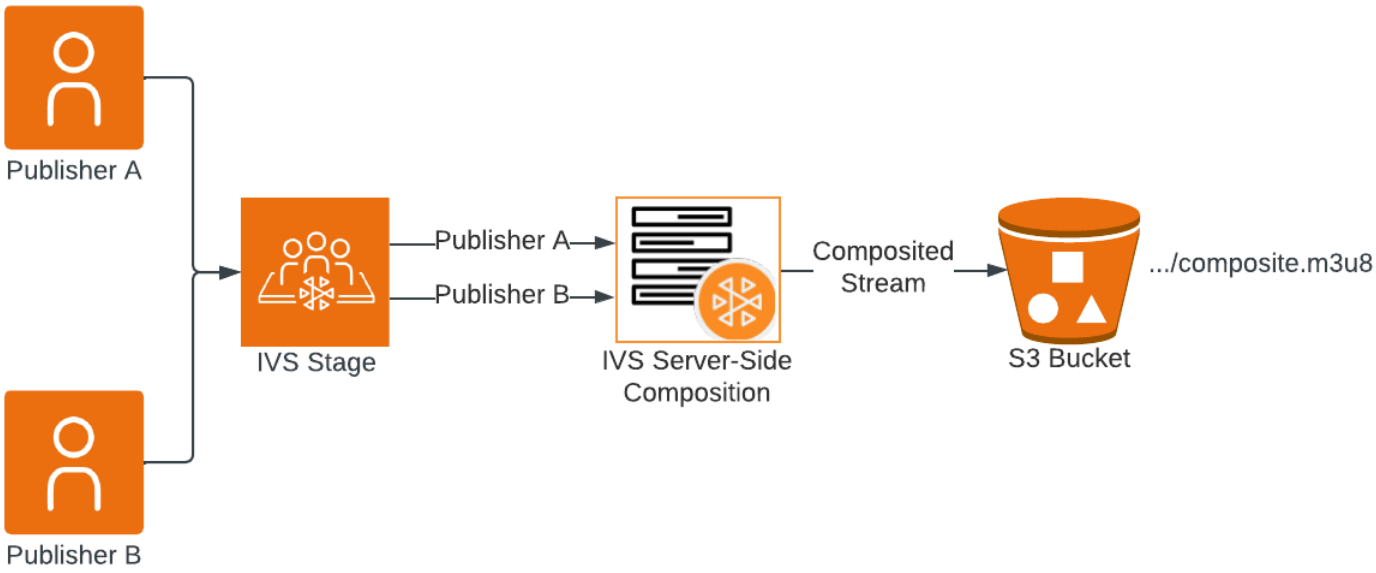
Individual Participant Recording

This option is ideal for live streams with a single publisher or when separate recordings of each publisher are needed, especially for moderation purposes. For more details, see Individual Participant Recording.



Composite Recording

This option combines media from multiple publishers into a single view and records it in one file, ideal for a video-on-demand experience. For more details, see Composite Recording.



Thumbnails

Thumbnail recording for IVS real-time streaming can be set up for both individual participant recordings and composite (multi-participant) recordings. To enable or disable thumbnail recording and adjust the interval at which thumbnails are generated:

- For individual participant recordings, use the `thumbnailConfiguration` property.
- For composite recordings, use the `thumbnailConfigurations` property.

Thumbnail intervals range from 1 to 86400 seconds (24 hours); by default, thumbnail recording is disabled. For details, see the Amazon IVS Real-Time Streaming API Reference.

A thumbnail configuration includes a `storage` field, which can be set to `SEQUENTIAL` and/or `LATEST`. The `storage` field determines the S3 storage behavior for the thumbnails:

- `SEQUENTIAL` saves all thumbnails in a serial manner. This is the default.
- `LATEST` saves only the most recent thumbnail, overwriting the previous one.

If you specify both `SEQUENTIAL` and `LATEST`, thumbnails are written to two separate S3 paths, one for the sequential archive and one for the latest thumbnail.

IVS Individual Participant Recording | Real-Time Streaming

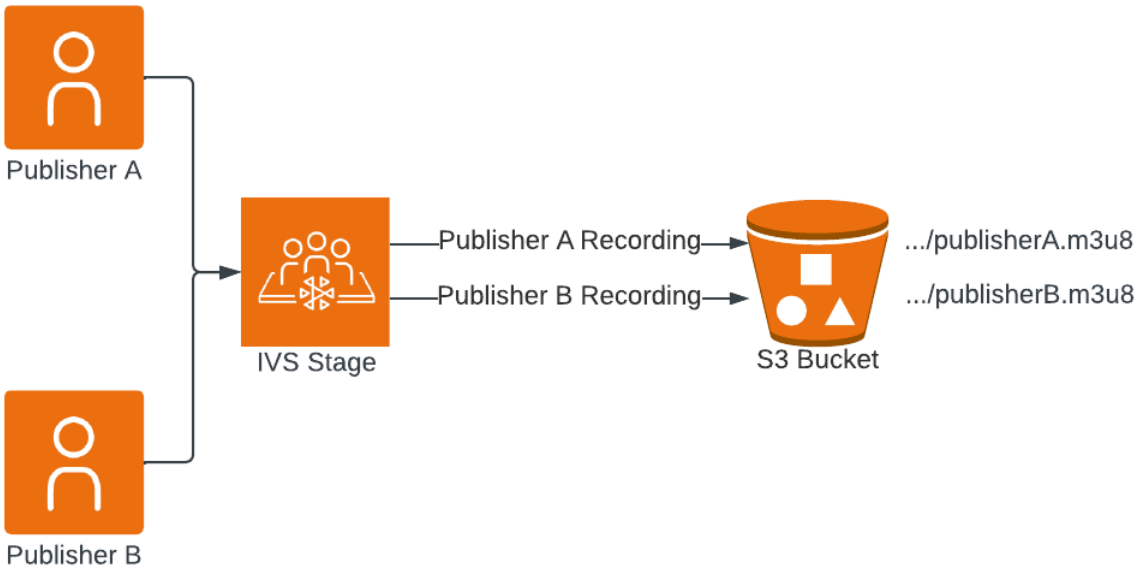
This document explains how to use individual participant recording with IVS real-time streaming.

Standard S3 storage and request costs apply. Thumbnails incur no additional IVS charges. For details, see Amazon IVS Pricing.

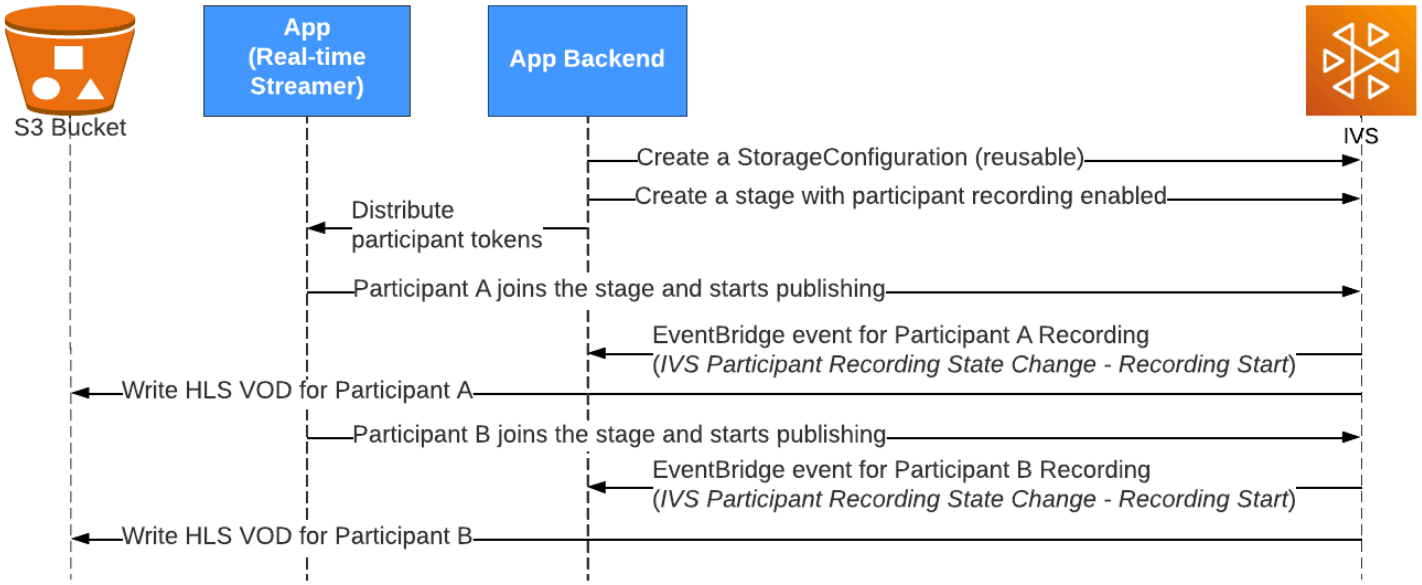
Introduction

Individual participant recording allows IVS real-time streaming customers to record IVS stage publishers individually into S3 buckets. When individual participant recording is enabled for a stage, publisher content is recorded once they start publishing to the stage.

Note: If you need to have all stage participants mixed in a single video, the composite recording feature is a better fit. See Recording for a summary of recording IVS real-time-streaming content.



Workflow



1. Create an S3 Bucket

You will need an S3 bucket to write VODs. For details, see the S3 documentation on how to create buckets. Note that for individual participant recording, the S3 buckets must be created in the same AWS region as the IVS stage.

Important: If you use an existing S3 bucket:

- The **Object Ownership** setting must be **Bucket owner enforced** or **Bucket owner preferred**.
- The **Default Encryption** setting must be **Server-side encryption with Amazon S3 managed keys (SSE-S3)**.

For details, see the S3 documentation on controlling ownership of objects and protecting data with encryption.

2. Create a StorageConfiguration Object

After creating a bucket, call the IVS real-time streaming API to create a StorageConfiguration object. Once the storage configuration is successfully created, IVS will have permission to write to the provided S3 bucket. You can re-use this StorageConfiguration object on multiple stages.

3. Create a Stage with Participant Tokens

Now you need to create an IVS stage with individual participant recording enabled (by setting the AutoParticipantRecordingConfiguration object), as well as participant tokens for each publisher.

The request below creates a stage with two participant tokens and individual participant recording enabled.

```
POST /CreateStage HTTP/1.1
Content-type: application/json

{
   "autoParticipantRecordingConfiguration": {
      "mediaTypes": ["AUDIO_VIDEO"],
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "thumbnailConfiguration": {
         "recordingMode": "INTERVAL",
         "storage": ["LATEST", "SEQUENTIAL"],
         "targetIntervalSeconds": 60
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
}
```

4. Join the Stage as an Active Publisher

Distribute the participant tokens to your publishers, and have them join the stage and start publishing to it.

When they join the stage and start publishing to it using one of IVS real-time streaming broadcast SDKs, the participant-recording process starts automatically and sends you an EventBridge event indicating that the recording started. (The event is IVS Participant Recording State Change - Recording Start.) Concurrently, the participant-recording process starts writing the VOD and metadata files to the configured S3 bucket. Note: Participants connected for extremely short durations (less than 5s) are not guaranteed to be recorded.

There are two ways to get the S3 prefix for each recording:

- Listen to the EventBridge event:

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "ivs-recordings",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
   }
}
```

- Use the GetParticipant API operation — The response includes the S3 bucket and prefix to where a participant is being recorded. Here is the request:

```
POST /GetParticipant HTTP/1.1
Content-type: application/json
{
   "participantID": "xYz1c2d3e4f",
   "sessionId": "st-ZyXwvu1T2s",
   "stageArn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
}
```

  And here is the response:

```
Content-type: application/json
{
   "participant": {
      ...
      "recordingS3BucketName": "ivs-recordings",
      "recordingS3Prefix": "<stage_id>/<session_id>/<participant_id>",
      "recordingState": "ACTIVE",
      ...
   }
}
```

5. Play Back the VOD

After the recording is finalized, you can watch it using the IVS player. See Playback of Recorded Content from Private Buckets for instructions on setting up CloudFront distributions for VOD playback.

Audio-Only Recording

When setting up individual participant recording, you can choose to have only audio HLS segments written to your S3 bucket. To use this feature, choose the `AUDIO_ONLY mediaType` when creating the stage:

```
POST /CreateStage HTTP/1.1
Content-type: application/json

{
   "autoParticipantRecordingConfiguration": {
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "mediaTypes": ["AUDIO_ONLY"],
      "thumbnailConfiguration": {
         "recordingMode": "DISABLED"
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
```

```
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
}
```

Thumbnail-Only Recording

When setting up individual participant recording, you can choose to have only thumbnails written to your S3 bucket. To use this feature, set `mediaType` to `NONE` when creating the stage. This ensures that no HLS segments are generated; thumbnails are still created and written to your S3 bucket.

```
POST /CreateStage HTTP/1.1
Content-type: application/json
{
   "autoParticipantRecordingConfiguration": {
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "mediaTypes": ["NONE"],
      "thumbnailConfiguration": {
         "recordingMode": "INTERVAL",
         "storage": ["LATEST", "SEQUENTIAL"],
         "targetIntervalSeconds": 60
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
```

```
}
```

Recording Contents

When individual participant recording is active, HLS video segments, metadata files, and thumbnails will start being written to the S3 bucket provided when the stage was created. This content is available for post-processing or playback as on-demand video.

Note that after a recording is finalized, an IVS Participant Recording State Change - Recording End event is sent through EventBridge. We recommend that you play back or process recorded streams only after this event is received. For details, see Using EventBridge with IVS Real-Time Streaming.

The following is a sample directory structure and contents of a recording of a live IVS session:

```
s3://mybucket/stageId/stageSessionId/participantId/timestamp
   events
      recording-started.json
      recording-ended.json
   media
      hls
  multivariant.m3u8
         high
            playlist.m3u8
            1.mp4
      thumbnails
         high
            1.jpg
            2.jpg
      latest_thumbnail
         high
            thumb.jpg
```

The `events` folder contains the metadata files corresponding to the recording event. JSON metadata files are generated when recording starts, ends successfully, or ends with failures:

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

A given `events` folder contains `recording-started.json` and either `recording-ended.json` or `recording-failed.json`. These contain metadata related to the recorded session and its output formats. JSON details are given below.

The `media` folder contains the supported media contents. The `hls` subfolder contains all media and the manifest files generated during the recording session and is playable with the IVS player. If configured, the `thumbnails` and `latest_thumbnail` subfolders contain JPEG thumbnail media files generated during the recording session.

Merge Fragmented Individual Participant Recordings

The `recordingReconnectWindowSeconds` property on a recording configuration allows you to specify a window of time (in seconds) during which, if a stage publisher disconnects from a stage and then reconnects, IVS tries to record to the same S3 prefix as the previous session. In other words, if a publisher disconnects and then reconnects within the specified interval, the multiple recordings are considered a single recording and merged together.

If thumbnail recording is enabled in `SEQUENTIAL` mode, then thumbnails are also merged under the same `recordingS3Prefix`. When the recordings are merged, the thumbnail counter restarts from the previous thumbnail value that was written for the previous recording.

IVS Recording State Change events in Amazon EventBridge: Recording End events and recording-ended JSON metadata files are delayed by at least `recordingReconnectWindowSeconds`, as IVS waits to ensure a new stream is not started.

For instructions on setting up the merge-streams functionality, see Step 2: Create a Stage with Optional Participant Recording in *Getting Started with Amazon IVS Real-Time Streaming*.

Eligibility

For multiple recordings to be merged using the same S3 prefix, certain conditions must be met for all the recordings:

- The value of the `recordingReconnectWindowSeconds` property of the AutoParticipantRecordingConfiguration for the stage is set greater than 0.
- The `StorageConfigurationArn` used to write the VOD artifacts is the same for each recording.
- The time difference in seconds between when the participant leaves and rejoins the stage is less than or equal to `recordingReconnectWindowSeconds`.

Note that the default value of `recordingReconnectWindowSeconds` is 0, which disables merging.

Synchronize Multiple Participant Recordings

Individual participant recordings include `EXT-X-PROGRAM-DATE-TIME` tags in HLS playlists, which provide precise UTC timestamps with millisecond accuracy for synchronizing recordings from multiple participants during post-processing.

When you record multiple participants individually and want to create a synchronized composition (such as a side-by-side or picture-in-picture layout), you can use these timestamps to align the recordings accurately, even if participants joined the stage at different times or experienced discontinuities potentially caused by network interruptions.

Each participant's HLS playlist includes `EXT-X-PROGRAM-DATE-TIME` tags that mark:

- The start of the recording (first segment).
- Any discontinuity points during the recording; e.g., when stitching occurs.

These timestamps use millisecond precision and are synchronized across all participants using the same time reference.

Example HLS Playlist

```
#EXTM3U
#EXT-X-VERSION:7
#EXT-X-TARGETDURATION:12
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MAP:URI="init-0.mp4"
#EXT-X-PROGRAM-DATE-TIME:2024-01-01T12:00:00.000Z
#EXTINF:3.30091,
0.mp4
#EXTINF:5.63794,
1.mp4
#EXTINF:2.74290,
2.mp4
#EXT-X-DISCONTINUITY
#EXT-X-MAP:URI="init-1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2024-01-01T12:00:52.772Z
#EXTINF:2.54412,
3.mp4
```

```
#EXTINF:5.63649,
4.mp4
```

The EXT-X-PROGRAM-DATE-TIME tags provide the exact UTC time for the first segment and at each discontinuity point, enabling precise synchronization with other participants' recordings.

Synchronization Workflow

To synchronize multiple participant recordings, extract the EXT-X-PROGRAM-DATE-TIME timestamps from each participant's HLS playlist and use them to calculate time offsets. These offsets can then be applied during post-processing composition using video processing tools like FFmpeg. When discontinuities are present in the recordings, timestamps at those points provide the necessary timing references to maintain accurate synchronization throughout the entire recording.

Note: For real-time synchronized output without post-processing, consider using server-side composition instead of individual participant recording.

JSON Metadata Files

This metadata is in JSON format. It comprises the following information:

Field	Type	Required	Description
stage_arn	string	Yes	ARN of the stage being used as the source of the recording.
session_id	string	Yes	String representing the stage's session_id where the participant is recorded.
participant_id	string	Yes	String representing the identifier of the recorded participant.
recording_started_at	string	Conditional	RFC 3339 UTC timestamp when the recording started. This is unavailable when recording_status is RECORDING_START_FAILED. Also, see

Field	Type	Required	Description
			the note below for `recording_ended_at`.
`recording_ended_at`	string	Conditional	RFC 3339 UTC timestamp when the recording ended. This is available only when `recording_status` is `"RECORDING_ENDED"` or `"RECORDING_ENDED_WITH_FAILURE"`. **Note:** `recording_started_at` and `recording_ended_at` are timestamps when these events are generated and may not exactly match the HLS video-segment timestamps. To accurately determine the duration of a recording, use the `duration_ms` field.
`recording_status`	string	Yes	Status of the recording. Valid values: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`.
`recording_status_message`	string	Conditional	Descriptive information on the status. This is available only when `recording_status` is `"RECORDING_ENDED"` or `"RECORDING_ENDED_WITH_FAILURE"`.

Field	Type	Required	Description
media	object	Yes	Object that contains the enumerated objects of media content available for this recording. Valid value: "hls".
hls	object	Yes	Enumerated field that describes the Apple HLS format output.
duration_ms	integer	Conditional	Duration of the recorded HLS content in milliseconds. This is available only when recording_status is "RECORDING_ENDED" or "RECORDING_ENDED_WITH_FAILURE". If a failure occurred before any recording was done, this is 0.
path	string	Yes	Relative path from the S3 prefix where HLS content is stored.
playlist	string	Yes	Name of the HLS master playlist file.
renditions	object	Yes	Array of renditions (HLS variants) of metadata objects. There always is at least one rendition.
path	string	Yes	Relative path from the S3 prefix where HLS content is stored for this rendition.
playlist	string	Yes	Name of the media playlist file for this rendition.

Field	Type	Required	Description
`thumbnails`	object	Conditional	Enumerated field that describes thumbnails output. This is available only when the thumbnail configuration's `storage` field includes `SEQUENTIAL`
`path`	string	Yes	Relative path from the S3 prefix where sequential thumbnail content is stored.
`renditions`	object	Yes	Array of renditions (thumbnail variants) of metadata objects. There always is at least one rendition.
`path`	string	Yes	Relative path from the S3 prefix where thumbnail content is stored for this rendition.
`latest_thumbnail`	object	Conditional	Enumerated field that describes thumbnails output. This is available only when the thumbnail configuration's `storage` field includes `LATEST`.
`path`	string	Yes	Relative path from the S3 prefix where `latest_thumbnail` is stored.
`renditions`	object	Yes	Array of renditions (thumbnail variants) of metadata objects. There always is at least one rendition.

Field	Type	Required	Description
path	string	Yes	Relative path from the S3 prefix where the latest thumbnail is stored for this rendition.
version	string	Yes	The version of the metadata schema.

Example: recording-started.json

```
{
   "version": "v1",
   "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T13:17:17Z",
   "recording_status": "RECORDING_STARTED",
   "media": {
      "hls": {
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
```

```
               "path": "high"
            }
         ]
      }
   }
}
```

Example: recording-ended.json

```
{
   "version": "v1",
   "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T19:44:19Z",
   "recording_ended_at": "2024-03-13T19:55:04Z",
   "recording_status": "RECORDING_ENDED",
   "media": {
      "hls": {
         "duration_ms": 645237,
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
               "path": "high"
            }
```

```
         ]
      }
   }
}
```

Example: recording-failed.json

```
{
   "version": "v1",
   "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T19:44:19Z",
   "recording_ended_at": "2024-03-13T19:55:04Z",
   "recording_status": "RECORDING_ENDED_WITH_FAILURE",
   "media": {
      "hls": {
         "duration_ms": 645237,
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
               "path": "high"
            }
         ]
      }
```

```
    }
}
```

Converting Recordings to MP4

Individual participant recordings are stored in the HLS format, consisting of playlists and fragmented MP4 (fMP4) segments. To convert an HLS recording into a single MP4 file, install FFmpeg and run the following command:

```
ffmpeg -i /path/to/playlist.m3u8 -i /path/to/playlist.m3u8 -map 0:v -map 1:a -c copy
 output.mp4
```

IVS Composite Recording | Real-Time Streaming

This document explains how to use the composite-recording feature within server-side composition. Composite recording allows you to generate HLS recordings of an IVS stage by effectively combining all stage publishers into one view using an IVS server, and then saving the resulting video to an S3 bucket.

Standard S3 storage and request costs apply. Thumbnails incur no additional IVS charges. For details, see Amazon IVS Pricing.

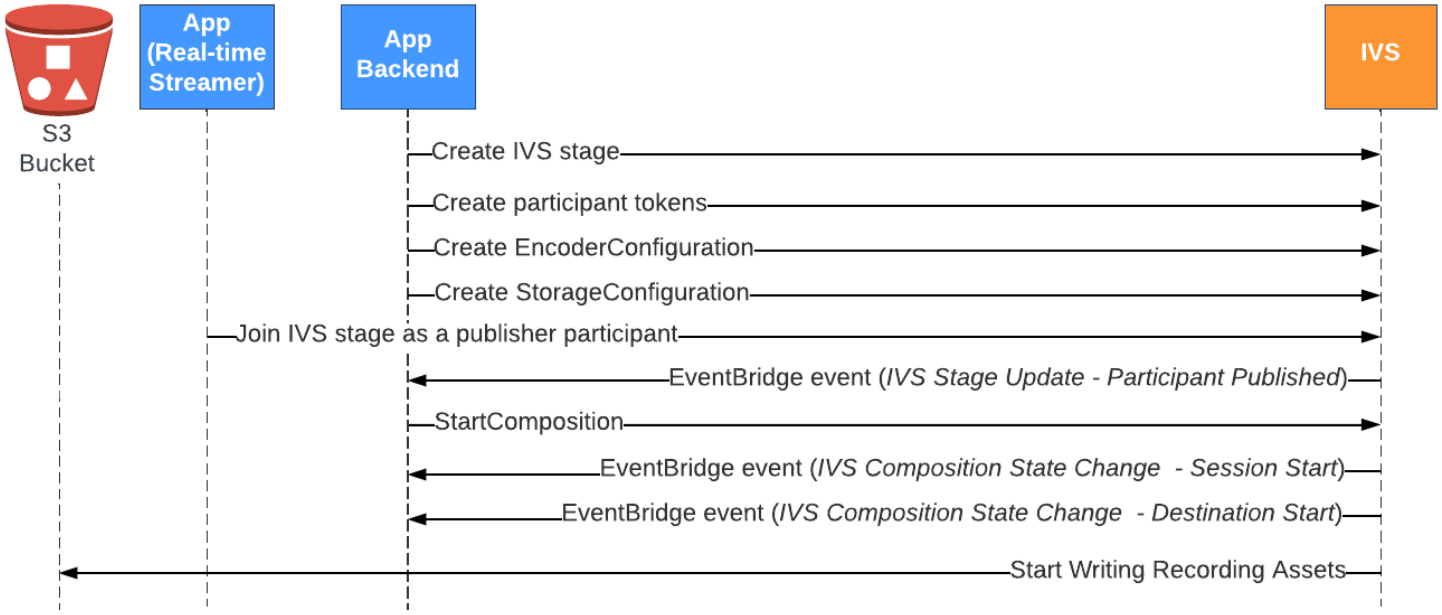
Prerequisites

To use composite recording, you must have a stage with active publishers and an S3 bucket to use as the recording destination. Below, we describe one possible workflow that uses EventBridge events to record a composition to an S3 bucket. Alternatively, you can start and stop compositions based on your own app logic.

1. Create an IVS stage and participant tokens for each publisher.
2. Create an EncoderConfiguration (an object representing how the recorded video should be rendered).
3. Create an S3 bucket and a StorageConfiguration (where the recording contents will be stored).

   Important: If you use an existing S3 bucket, the **Object Ownership** setting must be **Bucket owner enforced** or **Bucket owner preferred**. For details, see the S3 documentation on controlling ownership of objects.
4. Join the stage and publish to it.

5. When you receive a Participant Published EventBridge event, call StartComposition with an S3 DestinationConfiguration object as the destination
6. After a few seconds, you should be able to see the HLS segments being persisted to your S3 buckets.



Note: A composition performs auto-shutdown after 60 seconds of inactivity from publisher participants on the stage. At that point, the composition is terminated and transitions to a STOPPED state. A composition is automatically deleted after a few minutes in the STOPPED state. For details, see Composition Lifecycle in *Server-Side Composition*.

Composite Recording Example: StartComposition with an S3 Bucket Destination

The example below shows a typical call to the StartComposition operation, specifying S3 as the only destination for the composition. Once the composition transitions to an ACTIVE state, video segments and metadata will start to be written to the S3 bucket specified by the `storageConfiguration` object. To create compositions with different layouts, see "Layouts" in Server-Side Composition and the IVS Real-Time Streaming API Reference.

Request

```
POST /StartComposition HTTP/1.1
Content-type: application/json
```

```
{
   "destinations": [
      {
         "s3": {
            "encoderConfigurationArns": [
              "arn:aws:ivs:ap-northeast-1:927810967299:encoder-configuration/
PAAwglkRtjge"
            ],
            "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq",
     "thumbnailConfigurations": [
        {
    "storage": ["LATEST", "SEQUENTIAL"],
    "targetIntervalSeconds": 30
               }
     ]
  }
      }
   ],
   "idempotencyToken": "db1i782f1g9",
   "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr"
}
```

Response

```
{
    "composition": {
        "arn": "arn:aws:ivs:ap-northeast-1:927810967299:composition/s2AdaGUbvQgp",
        "destinations": [
            {
                "configuration": {
                    "name": "",
                    "s3": {
                        "encoderConfigurationArns": [
                            "arn:aws:ivs:ap-northeast-1:927810967299:encoder-
configuration/PAAwglkRtjge"
                        ],
                        "recordingConfiguration": {
                            "format": "HLS"
                        },
                        "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq",
```

```
                "thumbnailConfigurations": [
                    {
                "storage": ["LATEST", "SEQUENTIAL"],
                "targetIntervalSeconds": 30
                          }
                ]
                    }
                },
                "detail": {
                    "s3": {
                        "recordingPrefix": "MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/
composite"
                    }
                },
                "id": "2pBRKrNgX1ff",
                "state": "STARTING"
            }
        ],
        "layout": null,
        "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr",
        "startTime": "2023-11-01T06:25:37Z",
        "state": "STARTING",
        "tags": {}
    }
}
```

The `recordingPrefix` field, present in the StartComposition response can be used to determine where the recording contents will be stored.

Recording Contents

When the composition transitions to an `ACTIVE` state, HLS video segments, metadata files, and thumbnails (if configured) will start being written to the S3 bucket specified during the StartComposition call. This content is available for post-processing or playback as on-demand video.

Note that after a composition becomes live, an “IVS Composition State Change” event is emitted, and it may take a little time before the manifest files, video segments, and thumbnails are written. We recommend that you play back or process recorded streams only after the “IVS Composition State Change (Session End)” event is received. For details, see Using EventBridge with IVS Real-Time Streaming .

The following is a sample directory structure and contents of a recording of a live IVS session:

```
MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/composite
   events
      recording-started.json
      recording-ended.json
   media
      hls
      thumbnails
      latest_thumbnail
```

The `events` folder contains the metadata files corresponding to the recording event. JSON metadata files are generated when recording starts, ends successfully, or ends with failures:

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

A given `events` folder will contain `recording-started.json` and either `recording-ended.json` or `recording-failed.json`.

These contain metadata related to the recorded session and its output formats. JSON details are given below.

The `media` folder contains the supported media contents. The `hls` subfolder contains all media and the manifest files generated during the composition session and is playable with the IVS player. The HLS manifest is located in the `multivariant.m3u8` folder. If configured, the `thumbnails` and `latest_thumbnail` subfolders contain JPEG thumbnail media files generated during the composition session.

Bucket Policy for StorageConfiguration

When a StorageConfiguration object is created, IVS will get access to write content to the specified S3 bucket. This access is granted by making modifications to the S3 bucket's policy. *If the policy for the bucket is altered in a way that removes IVS's access, ongoing and new recordings will fail.*

The example below shows an S3 bucket policy that allows IVS to write to the S3 bucket:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CompositeWrite-y1d212y",
            "Effect": "Allow",
            "Principal": {
                "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::my-s3-bucket/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control"
                },
                "Bool": {
                    "aws:SecureTransport": "true"
                }
            }
        }
    ]
}
```

JSON Metadata Files

This metadata is in JSON format. It comprises the following information:

Field	Type	Required	Description
`stage_arn`	string	Yes	ARN of the stage being used as the source of the composition.
`media`	object	Yes	Object that contains the enumerated objects of media

Field	Type	Required	Description
			content available for this recording. Valid values: "hls".
hls	object	Yes	Enumerated field that describes the Apple HLS format output.
duration_ms	integer	Conditional	Duration of the recorded HLS content in milliseconds. This is available only when recording_status is "RECORDING_ENDED" or "RECORDING_ENDED_WITH_FAILURE". If a failure occurred before any recording was done, this is 0.
path	string	Yes	Relative path from the S3 prefix where HLS content is stored.
playlist	string	Yes	Name of the HLS master playlist file.
renditions	object	Yes	Array of renditions (HLS variant) of metadata objects. There always is at least one rendition.
path	string	Yes	Relative path from the S3 prefix where HLS content is stored for this rendition.
playlist	string	Yes	Name of the media playlist file for this rendition.
resolution_height	int	Conditional	Pixel resolution height of the encoded video. This is available only when the rendition contains a video track.

Field	Type	Required	Description
resolution_width	int	Conditional	Pixel resolution width of the encoded video. This is available only when the rendition contains a video track.
thumbnails	object	Conditional	Enumerated field that describes thumbnails output. This is available only when the thumbnail configuration's storage field includes SEQUENTIAL
path	string	Yes	Relative path from the S3 prefix where sequential thumbnail content is stored.
resolutions	object	Yes	Array of resolutions (thumbnail variants) of metadata objects. There always is at least one resolution.
path	string	Yes	Relative path from the S3 prefix where thumbnail content is stored for this resolution.
resolution_height	int	Yes	Pixel resolution height of the thumbnails.
resolution_width	int	Yes	Pixel resolution width of the thumbnails.
latest_thumbnail	object	Conditional	Enumerated field that describes thumbnails output. This is available only when the thumbnail configuration's storage field includes LATEST.

Field	Type	Required	Description
path	string	Yes	Relative path from the S3 prefix where `latest_thumbnail` is stored.
resolutions	object	Yes	Array of resolutions (thumbnail variants) of metadata objects. There always is at least one resolution.
path	string	Yes	Relative path from the S3 prefix where the latest thumbnail is stored for this resolution.
resolution_height	int	Yes	Pixel resolution height of the latest thumbnail.
resolution_width	int	Yes	Pixel resolution width of the latest thumbnail.
recording_ended_at	string	Conditional	RFC 3339 UTC timestamp when the recording ended. This is available only when `recording_status` is `"RECORDING_ENDED"` or `"RECORDING_ENDED_WITH_FAILURE"`. `recording_started_at` and `recording_ended_at` are timestamps when these events are generated and may not exactly match the HLS video-segment timestamps. To accurately determine the duration of a recording, use the `duration_ms` field.

Field	Type	Required	Description
`recording_started_at`	string	Conditional	RFC 3339 UTC timestamp when the recording started. This is unavailable when `recording_status` is `RECORDING_START_FAILED`. See the note above for `recording_ended_at`.
`recording_status`	string	Yes	Status of the recording. Valid values: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`.
`recording_status_message`	string	Conditional	Descriptive information on the status. This is available only when `recording_status` is `"RECORDING_ENDED"` or `"RECORDING_ENDED_WITH_FAILURE"`.
`version`	string	Yes	The version of the metadata schema.

Example: recording-started.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-11-01T06:01:36Z",
  "recording_status": "RECORDING_STARTED",
  "media": {
    "hls": {
      "path": "media/hls",
```

```
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

Example: recording-ended.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED",
  "media": {
    "hls": {
```

```
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

Example: recording-failed.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED_WITH_FAILURE",
```

```
  "media": {
    "hls": {
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

Playback of Recorded Content from Private Buckets

By default, the recorded content is private; hence, these objects are inaccessible for playback using the direct S3 URL. If you try to open the HLS multivariate playlist (m3u8 file) for playback using the IVS player or another player, you will get an error (e.g., “You do not have permission to access

the requested resource"). Instead, you can play back these files with the Amazon CloudFront CDN (Content Delivery Network).

CloudFront distributions can be configured to serve content from private buckets. Typically this is preferable to having openly accessible buckets where reads bypass the controls offered by CloudFront. You can set up your distribution to be served from a private bucket by creating an origin access control (OAC), which is a special CloudFront user that has read permissions on the private origin bucket. You can create the OAC after you create your distribution, through the CloudFront console or API. See Creating a new origin access control in the *Amazon CloudFront Developer Guide*.

Setting Up Playback using CloudFront with CORS Enabled

This example covers how a developer can set up a CloudFront distribution with CORS enabled, enabling playback of their recordings from any domain. This is especially useful during the development phase, but you can modify the example below to match your production needs.

Step 1: Create an S3 Bucket

Create an S3 bucket that will be used to store the recordings. Note that the bucket needs to be in the same region that you use for your IVS workflow.

Add a permissive CORS policy to the bucket:

1. In the AWS console, go to the **S3 Bucket Permissions** tab.
2. Copy the CORS policy below and paste it under **Cross-origin resource sharing (CORS)**. This will enable CORS access on the S3 bucket.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
```

```
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

Step 2: Create a CloudFront Distribution

See Creating a CloudFront distribution in the *CloudFront Developer Guide*.

Using the AWS console, enter the following information:

For this field ...	Choose this ...
Origin Domain	The S3 bucket created in the previous step
Origin Access	Origin access control settings (recommended), using default parameters
Default cache behavior: Viewer Protocol Policy	Redirect HTTP to HTTPS
Default cache behavior: Allowed HTTP methods	GET, HEAD and OPTIONS
Default cache behavior: Cache key and origin requests	CachingDisabled policy
Default cache behavior: Origin request policy	CORS-S3Origin
Default cache behavior: Response headers policy	SimpleCORS
Web Application Firewall	Enable security protections

Then save the CloudFront distribution.

Step 3: Set Up the S3 Bucket Policy

1. Delete any StorageConfiguration that you have set up for the S3 bucket. This will remove any bucket policies that were automatically added when creating the policy for that bucket.
2. Go to your CloudFront Distribution, make sure all distribution fields are in the states defined in the previous step, and **Copy the Bucket Policy** (use the **Copy policy** button).
3. Go to your S3 bucket. On the **Permissions** tab, select **Edit Bucket Policy** and paste the bucket policy that you copied in the previous step. After this step, the bucket policy should have the CloudFront policy exclusively.
4. Create a StorageConfiguration, specifying the S3 bucket.

After the StorageConfiguration is created, you will see two items in the S3 bucket policy, one allowing CloudFront to read contents and another one allowing IVS to write contents. An example of a final bucket policy, with CloudFront and IVS access, is shown in Example: S3 Bucket Policy with CloudFront and IVS Access.

Step 4: Play Back Recordings

After you successfully set up the CloudFront distribution and update the bucket policy, you should be able to play back recordings using the IVS player:

1. Successfully start a Composition and make sure you have a recording stored on the S3 bucket.
2. After following the Step 1 through Step 3 in this example, the video files should be available for consumption through the CloudFront URL. Your CloudFront URL is the **Distribution domain name** on the **Details** tab in the Amazon CloudFront console. It should be something like this:

   ```
   a1b23cdef4ghij.cloudfront.net
   ```

3. To play the recorded video through the CloudFront distribution, find the object key for your `multivariant.m3u8` file under the s3 bucket. It should be something like this:

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```

4. Append the object key to the end of your CloudFront URL. Your final URL will be something like this:

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```

5. You can now add the final URL to the source attribute of an IVS player to watch the full recording. To watch the recorded video, you can use the demo in Getting Started in the *IVS Player SDK: Web Guide*.

Example: S3 Bucket Policy with CloudFront and IVS Access

The snippet below illustrates an S3 bucket policy that allows CloudFront to read content to the private bucket and IVS to write content to the bucket. **Note: Do not copy and paste the snippet below to your own bucket. Your policy should contain the IDs that are relevant to your CloudFront distribution and StorageConfiguration.**

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CompositeWrite-7eiKaIGkC9DO",
      "Effect": "Allow",
      "Principal": {
        "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
```

```
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::844311324168:distribution/
E1NG4YMW5MN25A"
        }
      }
    }
  ]
}
```

Troubleshooting

- **The composition is not written to the S3 bucket —** Ensure that the S3 bucket and StorageConfiguration objects are created and in the same region. Also ensure that IVS has access to the bucket by checking your bucket policy; see Bucket Policy for StorageConfiguration.
- **I can't find a composition when performing *ListCompositions* —** Compositions are ephemeral resources. Once they transition to a final state, they are deleted automatically after a few minutes.
- **My composition stops automatically —** A composition will stop automatically if there is no publisher on the stage for more than 60 seconds.

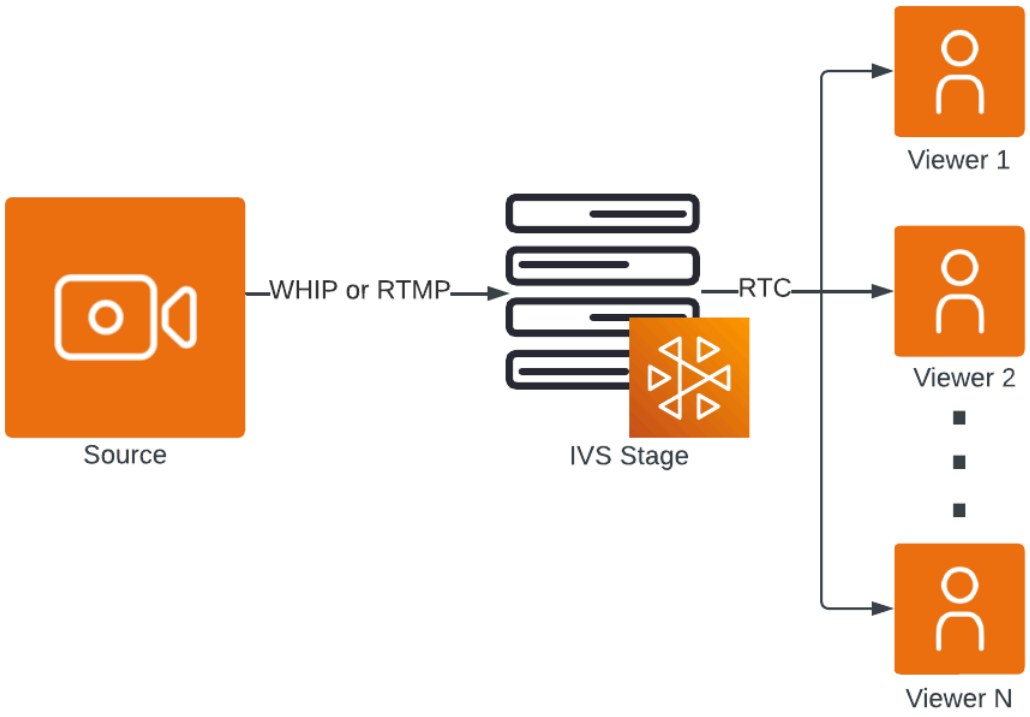
Known Issue

The media playlist written by composite recording has the tag `#EXT-X-PLAYLIST-TYPE:EVENT` while the composition is ongoing. When composition is done, the tag is updated to `#EXT-X-PLAYLIST-TYPE:VOD`. For a smooth playback experience, we recommend that you use this playlist only after the composition finalizes successfully.

IVS Stream Ingest | Real-Time Streaming

As an alternative to using the IVS broadcast SDK, you can publish video to an IVS stage from a WHIP or RTMP source. This approach offers flexibility for workflows where using the SDK is not feasible or preferred, such as when publishing video from OBS Studio or a hardware encoder. Whenever possible, we recommend using the IVS broadcast SDK, as we cannot guarantee the performance or compatibility of third-party solutions with IVS.

This diagram illustrates how publishing with WHIP and RTMP works:



Supported Protocols

IVS real-time streaming supports several ingest protocols:

- RTMP and RTMPS — RTMP (Real-Time Messaging Protocol) is the industry standard for transmitting video over a network. RTMPS is the secure version of RTMP that operates over TLS.

  IVS supports multitrack video capability of E-RTMP (Enhanced RTMP). See E-RTMP Multitrack Video in the IVS RTMP Publishing documentation.
- WHIP (WebRTC-HTTP Ingestion Protocol) — An IETF draft developed to standardize WebRTC ingestion.

For detailed guidance on using these protocols, see our RTMP and WHIP documentation.

Supported Media Specifications

- Audio input format
  - Codec: AAC-LC for RTMP and Opus for WHIP
  - Channels: 2 (Stereo) or 1 (Mono)
  - Sample rate: 44.1 kHz or 48 kHz
  - Maximum bitrate: 160 Kbps
- Video input format
  - Codec: H.264
  - H.264 profile: Baseline
  - IDR interval: 1 or 2 seconds
  - Frame rate: 10 to 60 FPS
  - B-frames: 0

    Note: The IVS broadcast SDK has B-frames enabled by default, but starting with version 1.25.0, it automatically disables B-frames when broadcasting to an IVS stage. For real-time streaming with other RTMP encoders, developers must disable B-frames. *If developers using other RTMP encoders do not disable B-frames, their streams will be disconnected*.
  - Resolution: Maximum: 720p. Minimum: 160p
  - Maximum bitrate: 8.5 Mbps

    Note: For single-track RTMP streams, this limit applies to that track. For multitrack video published using Enhanced RTMP, the limit applies to the combined bitrate of all video tracks.
  - Encoder configuration: We recommend using `veryfast` and `zerolatency` settings for an H.264 encoder. Also: the `sliced_threads` x264 option is included in the `zerolatency` presets, and we recommend that you disable it. For example, when using FFmpeg, your command should include: `-preset:v veryfast -tune zerolatency -x264-params sliced-threads=0`

IVS RTMP Publishing | Real-Time Streaming

This document outlines the process of publishing to an IVS stage using RTMP. For additional details on various ingest options, refer to the Stream Ingest documentation

Prerequisites

Create a Stage

To create a stage, use the following command:

```
aws ivs-realtime create-stage --name "test-stage"
```

See CreateStage for details, including the response.

Important: In the response, note the `endpoints` field, which lists both RTMP and RTMPS endpoints. These are required for setting up your RTMP encoder.

Create an Ingest Configuration

To publish to a stage using RTMPS, you must first create an ingest configuration and associate it with your stage. When you publish to the stage (using the stream key from the ingest configuration and the RTMP endpoint from the stage), the media will be published to the stage as a participant. You have the option to specify a `userId` and custom `attributes`, which will be associated with the participant that connects to the stage.

```
aws ivs-realtime create-ingest-configuration \
  --name 'test' \
  --stage-arn arn:aws:ivs:us-east-1:123456789012:stage/8faHz1SQp0ik \
  --user-id '123' \
  --ingest-protocol 'RTMPS'
```

See CreateIngestConfiguration for details, including the response.

When creating an ingest configuration, you can associate it with a specific stage ARN up front. Without this association, the stream key is unusable. Also, ingest configurations (including the `stageArn` field) can be updated via the UpdateIngestConfiguration operation, allowing you to reuse the same configuration for different stages.

Note: The ingest configuration `insecureIngest` field defaults to `false`, requiring the use of RTMPS. RTMP connections will be rejected. If you must use RTMP, set `insecureIngest` to `true`. We recommend using RTMPS unless you have specific and verified use cases that require RTMP.

RTMP Single-Track Video

Below we describe how to use OBS Studio; however, you can use any RTMP encoder that meets the IVS media specifications.

OBS Guide

1. Download and install the software: https://obsproject.com/download.
2. Click **Settings**. In the **Stream** section of the **Settings** panel, select **Custom** from the **Service** dropdown.
3. For the **Server**, enter the RTMP or RTMPS endpoint from the stage.
4. For the **Stream Key**, enter the `streamKey` from the ingest configuration.
5. Configure your video settings as you normally would, with a few restrictions:
   a. IVS real-time streaming supports input up to 720p at 8.5 Mbps. If you exceed either of these limits, your stream will be disconnected.
   b. We recommend setting your **Keyframe Interval** in the **Output** panel to 1s or 2s. A low keyframe interval allows video playback to start more quickly for viewers. We also recommend setting **CPU Usage Preset** to **veryfast** and **Tune** to **zerolatency**, to enable the lowest latency.
   c. Because OBS does not support simulcast, we recommend keeping your bitrate below 2.5 Mbps. This enables viewers on lower-bandwidth connections to watch.
   d. Disable B-frames, as streams with B-frames will be automatically disconnected. Do one of the following:
      - In x264 options, enter `bframes=0 sliced-threads=0`.
      - Set B-frames to 0 if it is an option (e.g., for NVENC).

   Note: RTMP streams must include both audio and video tracks, or they will be disconnected.
6. Select **Start Streaming**

Important: If your encoder's maximum bitrate is set to 8.5 Mbps, the publisher occasionally disappears from the session. This is because the maximum bitrate setting is only a target, and

encoders occasionally go over the target. To prevent this, set your encoder's maximum bitrate lower; e.g. to 6 Mbps.

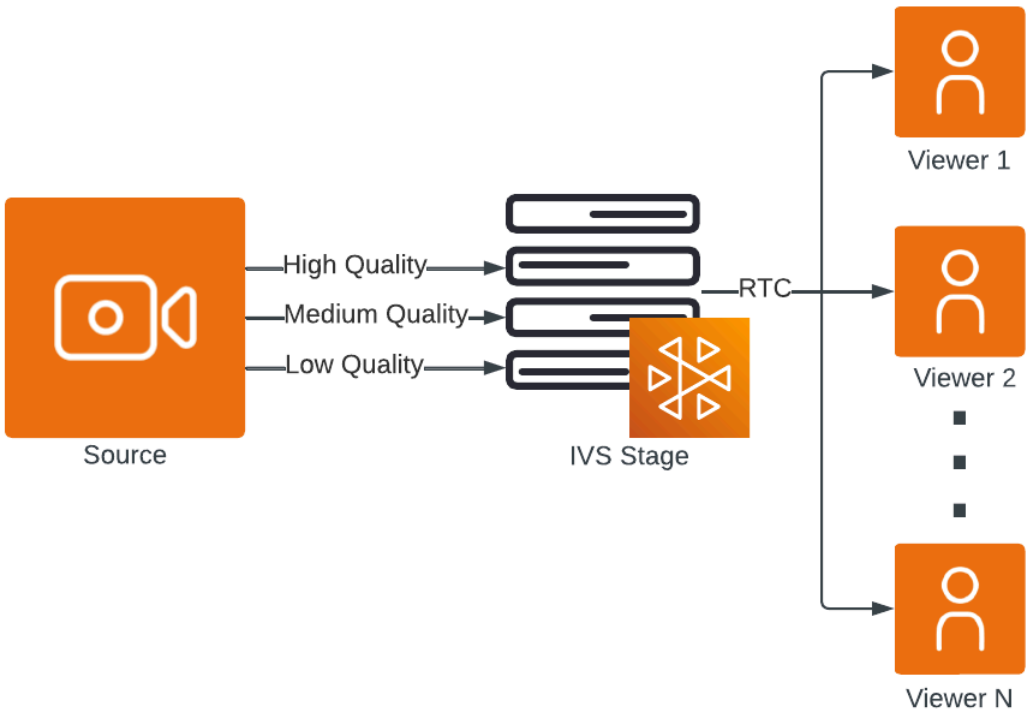
E-RTMP Multitrack Video

IVS supports the multitrack video capability of E-RTMP (Enhanced Real-Time Messaging Protocol), which allows you to publish multiple video qualities in a single RTMP stream to your IVS stage. This enables adaptive bitrate streaming, so subscribers can automatically watch in the best quality for their network connection.

Once ingested, the different video qualities are delivered to subscribers as simulcast layers. To configure which layers are received by subscribers, see the "Layered Encoding with Simulcast" sections in the real-time streaming broadcast SDK guides: Android, iOS, and Web.

For sample code, see aws-samples/sample-amazon-ivs-multitrack-video on GitHub.

This diagram illustrates how publishing with multitrack video works:



OBS Guide

1. Download and install OBS Studio:
   a. Windows: Multitrack video is supported starting in OBS Studio 30.2.

   b. macOS: Multitrack video is supported starting in OBS Studio 31.1 Beta (Apple Silicon only).
   c. Download at: https://obsproject.com/download.
2. Click **Settings**. In the **Stream** section of the **Settings** panel, select **Amazon IVS** from the **Service** dropdown.
3. For the **Server**, leave the setting as **Auto**.
4. For the **Stream Key**, enter the `streamKey` from the ingest configuration.
5. Under the **Multitrack Video** section, check **Enable Multitrack Video**.
6. In the **Video** panel, set the desired **Base (Canvas Resolution)** and **Output (Scaled) Resolution**. IVS real-time streaming supports input up to 720p. If you exceed this limit, your stream will be disconnected.

   When multitrack video is enabled, settings such as the number of video tracks, their bitrates, and the keyframe interval are automatically configured based on the device's capabilities.
7. Select **Start Streaming**.

Publishing with FFmpeg

You can use FFmpeg to publish live video and audio to IVS real-time streaming over RTMP. FFmpeg is a free, open-source project comprising a comprehensive suite of software libraries and tools for processing video, audio, and other multimedia content.

The following example command publishes a stream that includes a color pattern and a tone:

```
ffmpeg \
 -re \
 -f lavfi -i testsrc=d=300:s=1280x720:r=60,format=yuv420p \
 -f lavfi -i sine=f=440:b=4:d=300 \
 -c:v libx264 \
 -b:v 2500k \
 -g 60 -bf 0 \
 -profile:v baseline \
 -preset veryfast \
 -tune zerolatency \
 -x264opts sliced-threads=0 \
 -c:a aac \
 -ac 2 \
 -b:a 160k \
 -ar 48000 \
```

```
-f flv \
rtmps://$INGEST_ENDPOINT/app/$STREAM_KEY
```

In the example, replace `$INGEST_ENDPOINT` and `$STREAM_KEY` with your own values from the IVS console or API.

This configuration meets the Supported Media Specifications for IVS real-time streaming, including H.264 video (baseline profile, no B-frames, no sliced threads) and AAC audio.

Private Ingest to Stages

You can publish RTMP(S) and E-RTMP(S) streams to a stage from resources inside your Amazon VPC or from Direct Connect, by using an interface VPC endpoint. This enables a private connection between your VPC and IVS, keeping ingest traffic within the AWS network. To set up and configure an interface VPC endpoint for IVS, see IVS Private Ingest in the *IVS Low-Latency Streaming User Guide*.

Redundant Ingest

Redundant ingest enables streaming from two separate encoders simultaneously to a single stage, with automated failover for the same source media. This helps protect against source encoder failures and first-mile network issues. Redundant ingest is supported for RTMP(S) and E-RTMP(S) streams.

To enable redundant ingest, set `redundantIngest` to `true` when creating your ingest configuration via CreateIngestConfiguration. IVS provides two RTMP stream keys. Configure two separate encoders using the same ingest endpoint with their respective stream keys.

Each physical stream appears as a separate participant in participant APIs (e.g., ListParticipants). However, subscribers can subscribe to only one virtual participant, identified by the top-level `participantId` in the ingest configuration. IVS automatically controls which physical stream is used for the virtual participant. If you enable individual participant recording, each physical participant is recorded separately. If you enable server-side composition, only the virtual participant appears in the composition.

Redundant ingest also enables continuous 24/7 streaming. IVS limits individual publishers to 24 hours, but with redundant ingest, IVS staggers the connection timeouts between the two physical streams and automatically switches which stream is used for the virtual participant, allowing subscribers to experience uninterrupted 24/7 streaming.

Requirements

- Streams must be genlocked and must maintain matching encoding parameters (including resolution and frame rate) to ensure uninterrupted switchover.

Recommendations

- Use independent network connections with diverse network paths (e.g., different ISPs) for each encoder, to maximize protection against first-mile network issues and avoid single points of failure.
- Maintain active streams from both encoders.
- Test failover scenarios before production use.

IVS WHIP Publishing | Real-Time Streaming

This document explains how to use WHIP-compatible encoders like OBS to publish to IVS real-time streaming. WHIP (WebRTC-HTTP Ingestion Protocol) is an IETF draft developed to standardize WebRTC ingestion.

WHIP enables compatibility with software like OBS, offering an alternative (to the IVS broadcast SDK) for desktop publishing. More sophisticated streamers familiar with OBS may prefer it for its advanced production features, such as scene transitions, audio mixing, and overlay graphics. This provides developers with a versatile option: use the IVS web broadcast SDK for direct browser publishing or allow streamers to use OBS on their desktop for more powerful tools.

Also, WHIP is beneficial in situations where using the IVS broadcast SDK isn't feasible or preferred. For example, in setups involving hardware encoders, the IVS broadcast SDK might not be an option. However, if the encoder supports WHIP, you can still publish directly from the encoder to IVS.

WHIP requirements:

- Your SDP offer must include an H.264 video track, even if you are only publishing audio. If the offer does not include a video track, the connection will be rejected.
- The global WHIP endpoint (https://global.whip.live-video.net) returns a 307 Temporary Redirect. WHIP clients must handle 307 redirects correctly and persist headers in the redirected request, as required by the WHIP specification.

OBS Guide

OBS supports WHIP as of version 30. To start, download OBS v30 or newer: https://obsproject.com/.

To publish to an IVS stage using OBS via WHIP, follow these steps:

1. Generate a participant token with publish capability. In WHIP terms, a participant token is a bearer token. By default, participant tokens expire in 12 hours, but you can extend the duration up to 14 days.
2. Click **Settings**. In the **Stream** section of the **Settings** panel, select **WHIP** from the **Service** dropdown.
3. For the **Server**, enter https://global.whip.live-video.net.
4. For the **Bearer Token**, enter the participant token that you generated in step 1.
5. Configure your video settings as you normally would, with a few restrictions:
   a. IVS real-time streaming supports input up to 720p at 8.5 Mbps. If you exceed either of these limits, your stream will be disconnected.
   b. We recommend setting your **Keyframe Interval** in the **Output** panel to 1s or 2s. A low keyframe interval allows video playback to start more quickly for viewers. We also recommend setting **CPU Usage Preset** to **veryfast** and **Tune** to **zerolatency**, to enable the lowest latency.
   c. Because OBS does not support simulcast, we recommend keeping your bitrate below 2.5 Mbps. This enables viewers on lower-bandwidth connections to watch.
6. Press **Start Streaming**.

Note: We are aware of quality issues (like intermittent video freezing) that can occur with WHIP in OBS. These typically arise when the broadcaster's network is unstable. We recommend testing WHIP in OBS before using it for production live streams. Lowering your broadcast bitrate also may help reduce the occurrence of these issues.

IVS Participant Replication | Real-Time Streaming

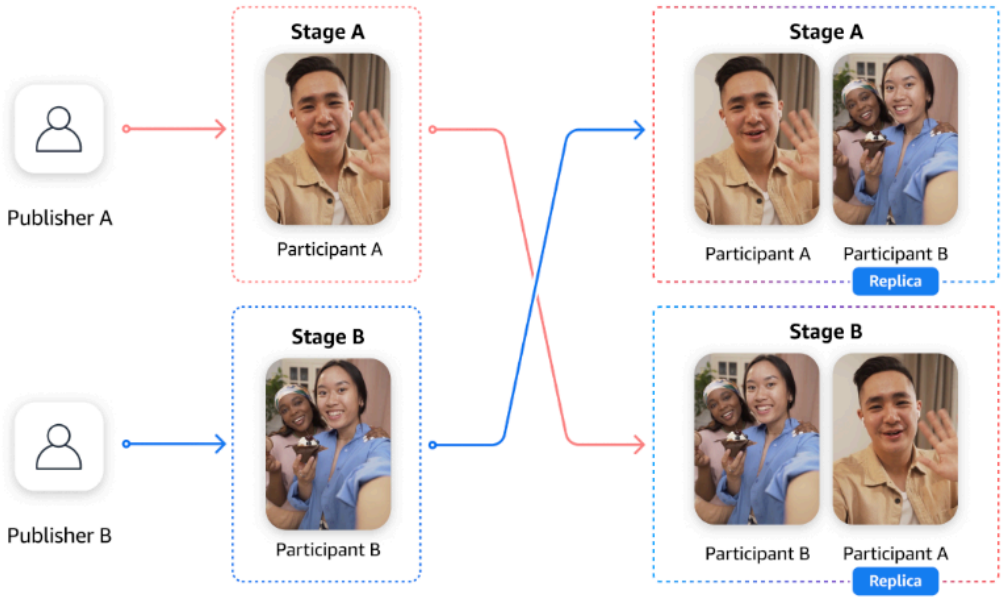
Participant replication allows you to copy a participant from one stage to another. This is useful when you want the same participant to appear in multiple stages at the same time, enabling cross-stage interactions.

A common use case in social live-streaming applications is competitions, often called VS Mode, where two streamers are temporarily matched so they can interact with each other in real time, while viewers of each stream can see both streamers.

Key Concepts:

- **Source stage** — The stage where the participant originally joined, which is used as the source for replication.
- **Destination stage** — The stage to which the participant is replicated.
- **Replicated participant** — A participant in a stage that is replicated to one or more destination stages.
- **Replica participant** — A participant in a destination stage that is replicated from another stage (the source stage).

Using Participant Replication



Prerequisites

To use participant replication, you must have at least two stages already created. For example, in the above scenario, we have two active publishers:

1. **Participant A**, connected to **stage A**
2. **Participant B**, connected to **stage B**

We will replicate participant A into stage B and participant B into stage A temporarily, to support a head-to-head competition.

Start Participant Replication

To replicate participants, use the StartParticipantReplication operation. You must call this once for each replication direction.

Replicate participant A to stage B:

```
aws ivs-realtime start-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --participant-id participant-a-id \
  --reconnect-window-seconds 10
```

Replicate participant B to stage A:

```
aws ivs-realtime start-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --participant-id participant-b-id \
  --reconnect-window-seconds 10
```

Once the replication is started, participants remain replicated until you explicitly stop it using the StopParticipantReplication operation. A replicated participant who disconnects and later reconnects within the interval specified by `reconnectWindowSeconds` will automatically appear again in both the source and destination stages. The default value for `reconnectWindowSeconds` is 0.

Stop Participant Replication

To stop replication, call the StopParticipantReplication operation.

Stop replication of participant A from stage A to stage B:

```
aws ivs-realtime stop-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --participant-id participant-a-id
```

Stop replication of participant B from stage B to stage A:

```
aws ivs-realtime stop-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --participant-id participant-b-id
```

IVS Service Quotas | Real-Time Streaming

The following are service quotas and limits for Amazon Interactive Video Service (IVS) real-time endpoints, resources, and other operations. Service quotas (also known as limits) are the maximum number of service resources or operations for your AWS account. That is, these limits are per AWS account, unless noted otherwise in the table. Also see AWS Service Quotas.

You use an endpoint to connect programmatically to an AWS service. Also see AWS Service Endpoints.

All quotas are enforced on a per-account basis in a specific AWS region.

Service Quota Increases

For quotas that are adjustable, you can request a rate increase through the AWS console. Use the console to view information about service quotas too.

API call rate quotas are not adjustable.

API Call Rate Quotas

Operation Type	Operation	Default
Composition	`GetComposition`	5 TPS
Composition	`ListCompositions`	5 TPS
Composition	`StartComposition`	5 TPS
Composition	`StopComposition`	5 TPS
IngestConfiguration	`CreateIngestConfiguration`	5 TPS
IngestConfiguration	`DeleteIngestConfiguration`	5 TPS
IngestConfiguration	`GetIngestConfiguration`	5 TPS
IngestConfiguration	`ListIngestConfigurations`	5 TPS

Operation Type	Operation	Default
IngestConfiguration	UpdateIngestConfiguration	5 TPS
MediaEncoder	CreateEncoderConfiguration	5 TPS
MediaEncoder	DeleteEncoderConfiguration	5 TPS
MediaEncoder	GetEncoderConfiguration	5 TPS
MediaEncoder	ListEncoderConfigurations	5 TPS
ParticipantReplication	ListParticipantReplicas	5 TPS
ParticipantReplication	StartParticipantReplication	5 TPS
ParticipantReplication	StopParticipantReplication	5 TPS
PublicKey	DeletePublicKey	3 TPS
PublicKey	GetPublicKey	3 TPS
PublicKey	ImportPublicKey	3 TPS
PublicKey	ListPublicKeys	3 TPS
Stage	CreateParticipantToken	50 TPS
Stage	CreateStage	5 TPS
Stage	DeleteStage	5 TPS
Stage	DisconnectParticipant	5 TPS
Stage	GetParticipant	5 TPS
Stage	GetStage	5 TPS
Stage	GetStageSession	5 TPS
Stage	ListStages	5 TPS

Operation Type	Operation	Default
Stage	UpdateStage	5 TPS
Stage	ListParticipants	5 TPS
Stage	ListParticipantEvents	5 TPS
Stage	ListStageSessions	5 TPS
StorageConfiguration	CreateStorageConfiguration	5 TPS
StorageConfiguration	DeleteStorageConfiguration	5 TPS
StorageConfiguration	GetStorageConfiguration	5 TPS
StorageConfiguration	ListStorageConfigurations	5 TPS
Tags	ListTagsForResource	10 TPS
Tags	TagResource	10 TPS
Tags	UntagResource	10 TPS

Other Quotas

Resource or Feature	Default	Adjustable	Description
Composition destinations	2	No	Maximum number of Destination objects in a Composition resource.
Composition: max duration	24	No	Maximum amount of time a composition can exist, in hours.
Compositions	20	Yes	Maximum concurrent Composition resources per account.

Resource or Feature	Default	Adjustable	Description
Compositions per stage	5	Yes	Maximum concurrent Composition resources per stage.
Concurrent participant replications	5	No	Maximum number of concurrent replications per participant across all stages in an AWS Region.
Concurrent publishers	1,000	Yes	Maximum number of participants who can be publishing across all stages in an AWS Region.
Concurrent subscriptions	20,000	Yes	Maximum number of simultaneous publisher-to-subscriber connections across all stages in an AWS Region.
EncoderConfigurations	20	Yes	Maximum number of EncoderConfiguration resources per account.
IngestConfigurations	100	Yes	Maximum number of IngestConfiguration resources per account.
Participant download bitrate	8.5 Mbps	No	Maximum aggregate download bitrate across all of a participant's subscriptions.
Participant publish bitrate	8.5 Mbps	No	Maximum bits per second that can be streamed to a stage.

Resource or Feature	Default	Adjustable	Description
Participant publish or subscribe duration	24	No	Maximum length of time a participant can publish or remain subscribed to a stage, in hours.
Participant publish resolution	720p	No	Maximum resolution of video published by participants.
PublicKeys	3	No	Maximum number of public keys, per AWS Region.
Stage participants (publishers)	12	No	Maximum number of participants who can be publishing to a stage at once.
Stage participants (subscribers)	10,000	Yes	Maximum number of participants who can be subscribing to a stage at once.
Stages	1,000	Yes	Maximum number of stages, per AWS Region.
StorageConfigurations	5	Yes	Maximum number of StorageConfiguration resources per account.

IVS Real-Time Streaming Optimizations

To ensure that your users have the best experience when streaming and viewing video using IVS real-time streaming, there are several ways you can improve or optimize for parts of the experience, using features that we offer today.

Introduction

When optimizing for a user's quality of experience, it's important to consider their desired experience, which can change depending on the content they are watching and network conditions.

Throughout this guide we focus on users who are either *publishers* of streams or *subscribers* of streams, and we consider the desired actions and experiences of those users.

The IVS SDKs allow you to configure the maximum bitrate, framerate, and resolution of the stream. When network congestion occurs for publishers, the SDK automatically adapts and lowers the video quality by lowering the bitrate, framerate, and resolution. On Android and iOS, it's possible to select the degradation preference when congestion is encountered. The same behavior is true whether you enable layered encoding with simulcast or keep the default configuration.

Adaptive Streaming: Layered Encoding with Simulcast

This feature is supported only in the following client versions:

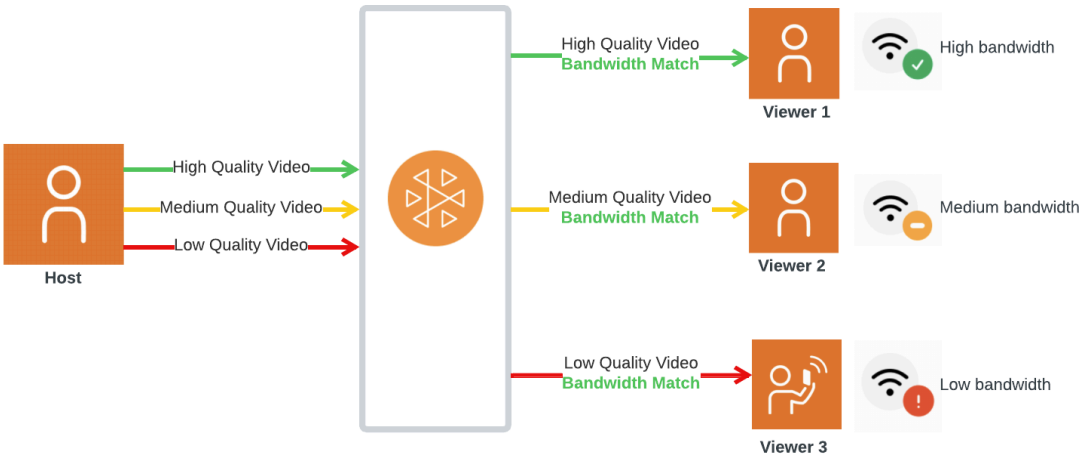
- iOS and Android 1.18.0+
- Web 1.12.0+

When using IVS real-time broadcast SDKs, publishers can encode multiple layers of video and subscribers automatically adapt or change to the quality best suited for their network. We call this *layered encoding with simulcast*.

Layered encoding with simulcast is supported on Android and iOS, and on the Chrome and Edge desktop browsers (for Windows and macOS). We do not support layered encoding on other browsers.

In the diagram below, the host is sending three video qualities (high, medium, and low). IVS forwards the highest quality video to each viewer based on available bandwidth; this provides an

optimal experience for each viewer. If Viewer 1's network connection changes from good to bad, IVS automatically starts sending Viewer 1 lower quality video, so Viewer 1 can keep watching the stream uninterrupted (with the best quality possible).



Default Layers, Qualities, and Framerates

The default qualities and layers provided for mobile and web users are as follows:

Mobile (Android, iOS)	Web (Chrome)
High layer (or custom): • Max bitrate: 900,000 bps • Framerate: 15 fps	High layer (or custom): • Max bitrate: 1,700,000 bps • Framerate: 30 fps
Mid layer: none (not needed, because the difference between the high- and low-layer bitrates on mobile is narrow)	Mid layer: • Max bitrate: 700,000 bps • Framerate: 20 fps
Low layer: • Max bitrate: 100,000 bps • Framerate: 15 fps	Low layer: • Max bitrate: 200,000 bps • Framerate: 15 fps

Resolution of Layers

The resolutions of the mid and low layers are automatically scaled down from the high layer, to maintain the same aspect ratio.

Mid and low layers are excluded if their resolutions are too close to the layer above. For example, if the configured resolution is 320x180, the SDK won't also send lower-resolution layers.

The table below shows the resolutions of layers generated for different configured resolutions. The listed values are in landscape orientation but can be applied in reverse for portrait content.

Input Resolution	Output Layer Resolutions: Mobile	Output Layer Resolutions: Web
720p (1280x720)	Hi (1280x720) Low (320x180)	Hi (1280x720) Mid (640x360) Low (320x180)
540p (960x540)	Hi (960x540) Low (320x180)	Hi (960x540) Low (320x180)
360p (640x360)	Hi (640x360) Low (360x180)	Hi (640x360) Low (360x180)
270p (480x270)	Hi (480x270)	Hi (480x270)
180p (320x180)	Hi (320x180)	Hi (320x180)

For custom input resolutions not mapped above, you can calculate them using the following tool.

Configuring Layered Encoding with Simulcast (Publisher)

To use layered encoding with simulcast, you must have enabled the feature on the client. If you enable it, you will see an increase in upload bandwidth usage by the publisher, potentially with less video freezing for viewers.

Android

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

Web

```
// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})

// Other Stage implementation code
```

For detailed information on configurating individual layers, see "Configuring Layered Encoding (Publisher)" in each broadcast SDK guide: Android, iOS, and Web.

Configuring Layered Encoding with Simulcast (Subscriber)

To configure what layers are received by subscribers, see the "Layered Encoding with Simulcast" sections in the real-time streaming SDK guides:

- Android Broadcast SDK
- iOS Broadcast SDK
- Web Broadcast SDK

With subscriber configuration, it's possible to define the `InitialLayerPreference`. This dictates what quality of video is delivered initially, as well as the `preferredLayerForStream`, which in turn determines what layer is selected during video playback. There are events and stream methods for notifying when layers change, adaption changes, or a layer selection is made.

Streaming Configurations

This section explores other configurations you can make to your video and audio streams.

Changing Video Stream Bitrate

To change the bitrate of your video stream, use the following configuration samples.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update Max Bitrate to 1.5mbps
config.setMaxBitrate(1500000);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update Max Bitrate to 1.5mbps
try! config.setMaxBitrate(1500000);

let cameraStream = IVSLocalStageStream(device: camera, configuration: config);

// Other Stage implementation code
```

Web

```
let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Bitrate to 1.5mbps or 1500kbps
   maxBitrate: 1500
})
```

```
// Other Stage implementation code
```

Changing Video Stream Framerate

To change the framerate of your video stream, use the following configuration samples.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update target framerate to 10fps
config.targetFramerate(10);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update target framerate to 10fps
try! config.targetFramerate(10);

let cameraStream = IVSLocalStageStream(device: camera, configuration: config);

// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      frameRate: {
         ideal: 10,
         max: 10,
      },
   },
});
```

```
let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Framerate to 10fps
   maxFramerate: 10
})
// Other Stage implementation code
```

Optimizing Audio Bitrate and Stereo Support

To change the bitrate and stereo settings of your audio stream, use the following configuration samples.

Web

```
// Note: Disable autoGainControl, echoCancellation, and noiseSuppression when enabling
 stereo.
const camera = await navigator.mediaDevices.getUserMedia({
   audio: {
      autoGainControl: false,
      echoCancellation: false,
      noiseSuppression: false
   },
});

let audioStream = new LocalStageStream(camera.getAudioTracks()[0], {
   // Optional: Update Max Audio Bitrate to 96Kbps. Default is 64Kbps
   maxAudioBitrateKbps: 96,

   // Signal stereo support. Note requires dual channel input source.
   stereo: true
})

// Other Stage implementation code
```

Android

```
StageAudioConfiguration config = new StageAudioConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
config.setMaxBitrate(96000);

AudioLocalStageStream microphoneStream = new AudioLocalStageStream(microphone, config);
```

```
// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
try! config.audio.setMaxBitrate(96000);

let microphoneStream = IVSLocalStageStream(device: microphone, config: config);

// Other Stage implementation code
```

Changing Subscriber Jitter Buffer MinDelay

To change the jitter buffer minimum delay for a participant who is being subscribed to, a custom `subscribeConfiguration` can be used. The jitter buffer determines how many packets are stored before playback begins. The minimum delay represents the target for the minimum amount of data that should be stored. Changing the minimum delay can help playback be more resilient when facing packet loss/connection issues.

The tradeoff when increasing the size of the jitter buffer is that it also will increase the delay before playback begins. Increasing the minimum delay provides more resilience, at the cost of impacting time to video. Note that increasing the minimum delay during playback has a similar effect: playback will pause briefly to allow the jitter buffer to fill.

If more resiliency is needed, we recommend starting with a minimum-delay preset of `MEDIUM` and setting the subscribe configuration before playback begins.

Note that the minimum delay is applied only if a participant is subscribe-only. If a participant is publishing themselves, the minimum delay is not applied. This is done to ensure that multiple publishers can speak to each other without additional delay.

The examples below use a minimum delay preset of `MEDIUM`. See the SDK reference documentation for all possible values.

Web

```
const strategy = {
   subscribeConfiguration: (participant) => {
```

```
      return {
         jitterBuffer: {
            minDelay: JitterBufferMinDelay.MEDIUM
         }
      }

   // ... other strategy functions
}
```

Android

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.jitterBuffer.setMinDelay(JitterBufferConfiguration.JitterBufferDelay.MEDIUM());

    return config;
}
```

iOS

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()
    try! config.jitterBuffer.setMinDelay(.medium())
    return config
}
```

Reducing Time to Video When Switching Between Stages

Applications might let users browse a feed of live streams. A user joins one stage, swipes to another stream, leaves the current stage, and quickly joins the next. Creating a new connection for each stage transition requires establishing a new network connection, which can increase the time until video is available.

Use a `RealTimeConnection` to keep one network connection open while the user browses streams. Create the connection once, optionally connect it before the first stage join, and reuse it

across stages. Each stage still uses its own participant token and maintains its own participants, streams, and lifecycle.

Create and retain the `RealTimeConnection` for the duration of the browsing session in an object that outlives an individual stage, such as an application coordinator, navigation controller, or view model. Do not create and disconnect it for every stage transition.

Android

The following example adds the `RealTimeConnection` lifecycle to the MainViewModel used in the Android sample. The view model implements Stage.Strategy and StageRenderer.

```
private var realTimeConnection: RealTimeConnection? = null
private var stage: Stage? = null

private fun setupConnection(connectionToken: String) {
    try {
        val connection = RealTimeConnection(getApplication(), connectionToken)

        // Establish the shared transport before the first Stage join.
        connection.connect()

        realTimeConnection = connection
    } catch (error: BroadcastException) {
        Log.e("BasicRealTime", "Unable to create the RealTimeConnection", error)
    }
}

private fun joinStage(participantToken: String) {
    val realTimeConnection = realTimeConnection ?: return

    try {
        stage?.leave()
        stage?.release()

        val stage = Stage(
            getApplication(),
            participantToken,
            realTimeConnection,
            this
        )
        stage.addRenderer(this)
        stage.join()
```

```
        this.stage = stage
    } catch (error: BroadcastException) {
        Log.e("BasicRealTime", "Unable to join the Stage", error)
    }
}

override fun onCleared() {
    stage?.leave()
    stage?.release()
    stage = null

    realTimeConnection?.disconnect()
    realTimeConnection?.release()
    realTimeConnection = null

    super.onCleared()
}
```

Calling `connect()` is optional. If the connection is not open when you join a stage, the SDK opens it automatically. Call `connect()` before the next stage join to establish the connection in advance and reduce time to video. Although the example only connects to one stage at a time, a `RealTimeConnection` can be connected to multiple stages concurrently.

Call `release()` when the application no longer needs the `RealTimeConnection`. After it is released, it cannot be reused.

iOS

The following example assumes that the application obtains a connection token when the user begins the multi-stage flow and obtains a participant token for each stage that the user joins.

```
private var realTimeConnection: IVSRealTimeConnection?
private var stage: IVSStage?

private func setupConnection(connectionToken: String) throws {
    let connection = try IVSRealTimeConnection(token: connectionToken)

    // Establish the shared transport before the first Stage join.
    try connection.connect()

    realTimeConnection = connection
}
```

```
private func joinStage(participantToken: String) throws {
    guard let realTimeConnection else {
        return
    }

    stage?.leave()

    let stage = try IVSStage(
        token: participantToken,
        on: realTimeConnection,
        strategy: self
    )
    stage.addRenderer(self)
    try stage.join()

    self.stage = stage
}

private func endStageTransitionFlow() {
    stage?.leave()
    stage = nil

    realTimeConnection?.disconnect()
    realTimeConnection = nil
}
```

Calling `connect()` is optional. If the connection is not open when you join a stage, the SDK opens it automatically. Call `connect()` before the next stage join to establish the connection in advance and reduce time to video. Although the example only connects to one stage at a time, an `IVSRealTimeConnection` can be connected to multiple stages concurrently.

Keep a strong reference to `IVSRealTimeConnection` while stages use it. Calling `disconnect()` leaves any stages using the connection.

Suggested Optimizations

Scenario	Recommendations
Streams with text, or slow moving content, like presentations or slides	Use layered encoding with simulcast or configure streams with lower framerate.

Scenario	Recommendations
Streams with action or a lot of movement	Use layered encoding with simulcast.
Streams with conversation or little movement	Use layered encoding with simulcast or choose audio-only (see "Subscribing to Participants" in the Real-Time Streaming Broadcast SDK Guides: Web, Android, and iOS).
Users streaming with limited data	Use layered encoding with simulcast or, if you want lower data usage for everyone, configure a lower framerate and lower the bitrate manually.
Users browse a feed of live streams and quickly move from one stage to the next	Reuse a `RealTimeConnection` for the browsing session. Call `connect()` before the first stage join to establish the connection in advance.

Network Requirements | Real-Time Streaming

Amazon IVS real-time streaming relies on WebRTC and WebSocket protocols for the transmission of media and data. To ensure a seamless experience, the destinations and ports listed below must be accessible. Any restrictions on inbound or outbound traffic to these destinations may disrupt the functionality of IVS real-time streaming.

You can use this network test tool to ensure that your network is properly configured and that traffic to and from the necessary destinations and ports is not being blocked.

Common

Destination	Ports
*.live-video.net	TCP:443

Media

By default, IVS real-time streaming relies on UDP for the transmission of media to ensure low-latency and high-performance streaming. If UDP is blocked for participants subscribing to media, TCP is used as a fallback. TCP fallback is not supported for publishing, so if UDP traffic is completely blocked, publishing will fail.

Destination	Ports
All subnets listed under the `IVS_REALTIME` service in ip-ranges.json must be accessible, regardless of their `region` or your chosen AWS Region. Participants may be connected to any subnet automatically. See Global Solution, Regional Control for details.	UDP:3478 UDP:443 TCP:3478 (Fallback) TCP:443

IVS Costs | Real-Time Streaming

See the IVS Pricing page for details about costs for IVS.

- **Subscribing and publishing to stages** — Subscribing and publishing consume resources, and you will incur an hourly rate for the time you are connected to the stage.
- **Recording** — Individual participant recording incurs no additional Amazon IVS charges, while composite recording incurs charges for the hourly rate for the video encoded. Both recording options incur standard S3 storage and request costs. Thumbnails incur no additional IVS charges.
- **Participant replication** — Replica participants are billed the same as regular participants.

  For example, suppose you have two stages, Stage A with Participant A and Stage B with Participant B. You are charged for two participants.

  If Participant A is replicated to Stage B, you now have three connected participants (Participant A, Participant B, and the replica of Participant A). For the duration of the replication, you are charged for three participants.

More information is on the IVS Pricing page.

Cost Allocation Tags

You can assign tags to your Amazon IVS resources (such as stages) and use them as cost allocation tags to organize and track your Amazon IVS real-time streaming costs. A tag is a key-value pair that you define—for example, by application, environment, team, or event. After you activate cost allocation tags, AWS includes them in your cost allocation report so you can categorize and track your AWS spending at a finer level of detail.

To use cost allocation tags with Amazon IVS:

1. Tag your Amazon IVS resources. You can add tags when you create a resource or add them later, using the Amazon IVS console, the AWS CLI, or the Amazon IVS API. For tag restrictions and naming requirements, see Best practices and strategies in *Tagging AWS Resources and Tag Editor*. For the Amazon IVS tagging operations (such as `TagResource`), see the Amazon IVS Real-Time Streaming API Reference.
2. Activate your tags as cost allocation tags in the AWS Billing and Cost Management console. Only tags you have activated appear in your billing reports. See Activating user-defined cost

allocation tags. After you apply tags to resources, it can take up to 24 hours for the tag keys to appear on the Cost allocation tags page, and up to another 24 hours for them to activate.

3. View your costs by tag using AWS Cost Explorer, AWS Cost and Usage Reports, or your monthly cost allocation report.

For more information, see Organizing and tracking costs using AWS cost allocation tags in the *AWS Billing User Guide*.

IVS Resources and Support | Real-Time Streaming

This document lists resources to help support your use of Amazon IVS real-time streaming.

Demos and Other Resources

https://ivs.rocks/ is a dedicated site to browse published content (demos, code samples, blog posts), estimate cost, and experience IVS through live demos. For demos and code samples, see https://ivs.rocks/examples.

The Amazon IVS page on the DEV Community site has a wealth of demos and blog posts. For example, Getting Started with Amazon Interactive Video Service is a series of articles about using IVS, for beginners. The articles give step-by-step walkthroughs of IVS APIs with interactive demos embedded in the posts. All the demos can be run directly in the posts themselves via an embedded CodePen.

The AWS Blogs site has many IVS blog postings on a variety of topics. Filter for IVS by selecting **Product or solution** > **Media Services** > **Amazon Interactive Video Service** on the right side of the page.

On GitHub, the IVS real-time streaming demo for iOS and Android shows developers how to use IVS to build a compelling real-time, social-user-generated content application:

This application features a scrollable feed of user-generated real-time streams. Users can create video streams and audio-only rooms. Video-stream guests can join in guest spot or versus (VS) mode. Instructions on how to deploy the required backend and build the application are available in the following GitHub repositories:

- iOS: https://github.com/aws-samples/amazon-ivs-real-time-for-ios-demo/
- Android: https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- Backend: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

Support

The AWS Support Center offers a range of plans that provide access to tools and expertise to support your AWS solutions. All support plans provide 24/7 access to customer service. For technical support and more resources to plan, deploy, and improve your AWS environment, choose a support plan that best aligns with your AWS use case.

AWS Premium Support is a one-on-one, fast-response support channel to help you build and run applications on AWS.

AWS re:Post is a community-based Q&A site for developers to discuss technical questions related to Amazon IVS.

Contact AWS has links for nontechnical inquiries about your billing or account. For technical questions, use the discussion forums or support links above.

IVS Glossary

Also see the AWS glossary. In the table below, LL stands for IVS low-latency streaming; RT, IVS real-time streaming.

Term	Description	LL	RT	Chat
AAC	Advanced Audio Coding. AAC is an audio coding standard for lossy digital audio compression. Designed to be the successor of the MP3 format, AAC generally achieves higher sound quality than MP3 at the same bitrate. AAC has been standardi zed by ISO and IEC as part of the MPEG-2 and MPEG-4 specifications.	✓	✓	
Adaptive bitrate streaming	Adaptive Bitrate (ABR) streaming allows the IVS player to switch to a lower bitrate when connectio n quality suffers, and to switch back to a higher bitrate when connection quality improves.	✓		
Adaptive streaming	See Layered encoding with simulcast.		✓	
Administrative user	An AWS user with administrative access to resources and services available in an AWS account. See Terminology in *AWS Setup User Guide*.	✓	✓	✓
ARN	Amazon Resource Name, a unique identifier for an AWS resource. Specific ARN formats depend on the resource type. For ARN formats used by IVS resources, see in *Service Authorization Reference*.	✓	✓	✓
Aspect ratio	Describes the ratio of frame width to frame height. For example, 16:9 is the aspect ratio that corresponds to the Full HD or 1080p resolution.	✓	✓	
Audio mode	A preset or custom audio configuration optimized for different types of mobile device users and the		✓	

Term	Description	LL	RT	Chat
	equipment that they use. See IVS Broadcast SDK: Mobile Audio Modes (Real-Time Streaming).			
AVC, H.264, MPEG-4 Part 10	Advanced Video Coding, also referred to as H.264 or MPEG-4 Part 10, a video compression standard for lossy digital video compression.	✓	✓	
Background replacement	A type of camera filter that enables live-stre am creators to change their backgrounds. See Background Replacement in *IVS Broadcast SDK: Third-Party Camera Filters (Real-Time Streaming).*		✓	
Bitrate	A streaming metric for the number of bits transmitted or received per second.	✓	✓	
Broadcast, broadcaster	Other terms for stream, streamer.	✓		
Buffering	A condition that occurs when the playback device is unable to download the content before the content is supposed to be played. Buffering can manifest in several ways: content may randomly stop and start (also known as stuttering), content may stop for long periods of time (also known as freezing), or the IVS player may pause playback.	✓	✓	
Byte-range playlist	A more granular playlist than the standard HLS playlist. The standard HLS playlist is made up of 10-second media files. With a byte-range playlist, the segment duration is the same as the keyframe interval configured for the stream. Byte-range playlist is available only for the broadcasts that were auto-recorded to an S3 bucket. It is created in addition to the HLS playlist. See Byte-Range Playlists in *Auto-Record to Amazon S3 (Low-Latency Streaming).*	✓		

Term	Description	LL	RT	Chat
CBR	Constant Bitrate, a rate-control method for encoders that maintains a consistent bitrate throughout the entire playback of a video, regardless of what is happening during the broadcast. Lulls in the action may be padded to achieve the desired bitrate, and peaks may be quantized by adjusting the quality of encoding to match the target bitrate. *We strongly recommend using CBR instead of VBR.*	✓	✓	
CDN	Content Delivery Network or Content Distribut ion Network, a geographically distributed solution that optimizes delivery of content such as streaming video by bringing it closer to where users are located.	✓		
Channel	An IVS resource that stores configuration for streaming, including an ingest server, a stream key, a playback URL, and recording options. Streamers use the stream key associated with a channel to start a broadcast. All metrics and events generated during a broadcast are associate d with a channel resource.	✓		
Channel type	Determines the allowable resolution and frame rate for the channel. See Channel Types in the *IVS Low-Latency Streaming API Reference*.	✓		
Chat logging	An advanced option that can be enabled by associating a logging configuration with a chat room.			✓

Term	Description	LL	RT	Chat
Chat room	An IVS resource that stores configuration for a chat session, including optional features such as Message Review Handler and Chat Logging. See Step 2: Create a Chat Room in *Getting Started with IVS Chat*.			✓
Client-side composition	Uses a host device to mix audio and video streams from stage participants and then sends them as a composite stream to an IVS channel. This allows more control over the look of the composition at the cost of higher utilization of client resources and a higher risk of a stage or a host issue impacting the viewers. Also see server-side composition.	✓	✓	
CloudFront	A CDN service provided by Amazon.	✓		
CloudTrail	An AWS service for collecting, monitoring, analyzing, and retaining events and account activity from AWS and external sources. See Logging IVS API Calls with AWS CloudTrail.	✓	✓	✓
CloudWatch	An AWS service for monitoring applications, responding to performance changes, optimizing resource use, and providing insights into operational health. You can use CloudWatch to monitor IVS metrics; see Monitoring IVS Real-Time Streaming and Monitoring IVS Low-Latency Streaming.	✓	✓	✓
Composition	The process of combining audio and video streams from multiple sources into a single stream.	✓	✓	
Composition pipeline	A sequence of processing steps required to combine multiple streams and encode the resulting stream.	✓	✓	

Term	Description	LL	RT	Chat
Compression	Encoding of information using fewer bits than the original representation. Any particular compression is either lossless or lossy. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by removing unnecessary or less important information.	✓	✓	
Control plane	Stores information about IVS resources such as channels, stages, or chat rooms and provides interfaces for creating and managing these resources. It is regional (based on AWS regions).	✓	✓	✓
CORS	Cross-Origin Resource Sharing, an AWS feature that allows client web applications that are loaded in one domain to interact with resources such as S3 buckets in a different domain. Access can be configured based on headers, HTTP methods, and origin domains. See Using cross-origin resource sharing (CORS) - Amazon Simple Storage Service in *Amazon Simple Storage Service User Guide*.	✓		
Custom audio source	An interface provided by the IVS Broadcast SDK that allows an application to provide its own audio input, instead of being limited to the device's built-in microphone.		✓	
Custom image source	An interface provided by the IVS Broadcast SDK that allows an application to provide its own image input instead of being limited to the preset cameras.	✓	✓	
Custom participant ordering	Enables positioning of stage participants in both grid and PiP layouts based on custom attribute values in participant tokens.		✓	

Term	Description	LL	RT	Chat
Data plane	The infrastructure that carries data from ingest to egress. It operates based on the configuration managed in the control plane and is not restricted to an AWS region.	✓	✓	✓
Encoder, encoding	The process of converting video and audio content into a digital format, suitable for streaming. Encoding can be hardware or software based.	✓	✓	
E-RTMP	Enhanced RTMP protocol. IVS supports the features of E-RTMP required for multitrack video.	✓		
Event	An automatic notification published by IVS to the AmazonEventBridge monitoring service. An event represents a state or health change of a streaming resource such as a stage or a composition pipeline. See Using Amazon EventBridge with IVS Low-Latency Streaming and Using Amazon EventBridge with IVS Real-Time Streaming.	✓	✓	✓
FFmpeg	A free and open-source software project consisting of a suite of libraries and programs for handling video and audio files and streams. FFmpeg provides a cross-platform solution to record, convert and stream audio and video.	✓		
Fragmented stream	Created when a broadcast disconnects and then reconnects within the interval specified in the channel's recording configuration. The resulting multiple streams are considered a single broadcast and merged together into a single recorded stream. See Merge Fragmented Streams in *Auto-Record to Amazon S3 (Low-Latency Streaming)*.	✓		
Frame rate	A streaming metric for the number of video frames transmitted or received per second.	✓	✓	

Term	Description	LL	RT	Chat
HLS	HTTP Live Streaming (HLS), an HTTP-based adaptive bitrate streaming communications protocol used to deliver IVS streams to viewers.	✓		
HLS playlist	A list of media segments that make up a stream. Standard HLS playlists are made up of 10-second media files. HLS also supports more granular byte-range playlists.	✓		
Host	A real-time user who creates a stage.		✓	
IAM	Identity and Access Management, an AWS service that allows users to securely manage identitie s and access to AWS services and resources, including IVS.	✓	✓	✓
Ingest	IVS process for receiving video streams from a host or broadcaster for processing or delivery to viewers or other participants.	✓	✓	
Ingest server	Receives video streams and delivers them to a transcoding system, where streams are transmuxe d or transcoded into HLS for delivery to viewers. Ingest servers are specific IVS components that receive streams for channels, along with an ingestion protocol (RTMP, RTMPS). See the information on creating a channel in Getting Started with IVS Low-Latency Streaming.	✓		
Interlaced video	Transmits and displays only odd or even lines of subsequent frames to create perceived doubling of frame rate without consuming extra bandwidth. We do not recommend using interlaced video due to the video quality concerns.	✓	✓	

Term	Description	LL	RT	Chat
JSON	JavaScript Object Notation, an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types or other serializable values.	✓	✓	✓
Keyframe, delta frame, keyframe interval	The keyframe (also referred to as intra-cod ed or i-frame) is a full frame of the image in a video. Subsequent frames, the delta frames (also referred to as predicted or p-frames), only contain the information that has changed. Keyframes will appear multiple times within a stream, depending on the keyframe interval defined in the encoder.	✓	✓	
Lambda	An AWS service for running code (referred to as Lambda functions) without provisioning any server infrastructure. Lambda functions can run in response to events and invocation requests, or based on a schedule. For example, IVS Chat uses Lambda functions to enable message review for a chat room.	✓	✓	✓
Latency, glass-to-glass latency	A delay in data transfer. IVS defines latency ranges as: • Low latency: under 3 sec • Real-time latency: under 300 ms *Glass-to-glass* latency refers to the delay from when a camera captures a live stream to when the stream appears on a viewer's screen.	✓	✓	

Term	Description	LL	RT	Chat
Layered encoding with simulcast	Enables simultaneous encoding and publishing of multiple video streams with different quality levels. See Adaptive Streaming: Layered Encoding with Simulcast in *Real-Time Streaming Optimizations*.		✓	
Message review handler	Enables IVS Chat customers to automatically review/filter user chat messages before they are delivered to the chat room. It is enabled by associating a Lambda function with a chat room. See Creating a Lambda Function in *Chat Message Review Handler*.			✓
Mixer	A feature of the IVS Mobile Broadcast SDKs that takes multiple audio and video sources and generates a single output. It supports management of on-screen video and audio elements representing sources such as cameras, microphones, screen captures, and audio and video generated by the application. The output can then be streamed to IVS. See Configuring a Broadcast Session for Mixing in *IVS Broadcast SDK: Mixer Guide (Low-Latency Streaming)*.	✓		
Multi-host streaming	Combines streams from multiple hosts into a single stream. This can be accomplished using either client-side or server-side composition. Multi-host streaming enables scenarios such as inviting viewers onto a stage for Q&A, competitions between hosts, video chat, and hosts conversing with each other in front of a large audience.		✓	

Term	Description	LL	RT	Chat
Multitrack video	Allows broadcaster software tools to encode and stream multiple video qualities directly from a GPU-powered computer. See Amazon IVS Multitrack Video.	✓		
Multivariant playlist	An index of all the variant streams available for a broadcast.	✓		
OAC	Origin Access Control, a mechanism for restricting access to an S3 bucket, so that content such as a recorded stream can be served only through CloudFront CDN.	✓		
OBS	Open Broadcaster Software, free and open source software for video recording and live streaming. OBS offers an alternative (to the IVS broadcast SDK) for desktop publishing. More sophisticated streamers familiar with OBS may prefer it for its advanced production features, such as scene transitions, audio mixing, and overlay graphics.	✓	✓	
Participant	A real-time user connected to a stage as a publisher or subscriber.		✓	
Participant ordering	The order in which stage participants are positioned in grid and PiP layouts.		✓	
Participant token	Authenticates a real-time event participant when they join a stage. A participant token also controls whether a participant can send video to the stage.		✓	

Term	Description	LL	RT	Chat
Playback token, playback key pair	An authorization mechanism that allows customers to restrict video playback on private channels. Playback tokens are generated from a playback key pair. A playback key pair is the public-private pair of keys used to sign and validate the viewer authorization token for playback. See Create or Import an IVS Playback Key in *Setting up IVS Private Channels* and see the Playback Key Pair operations in the IVS Low-Latency API Reference.	✓		
Playback URL	Identifies the address a viewer uses to start playback for a specific channel. This address can be used globally. IVS automatically selects the best location on the IVS global content delivery network for delivering the video to each viewer. See the information on creating a channel in Getting Started with IVS Low-Latency Streaming.	✓		
Private channel	Allows customers to restrict access to their streams using an authorization mechanism based on playback tokens. See Workflow for IVS Private Channels in *Setting up IVS Private Channels*.	✓		
Private ingest	Enables secure private connection between your Amazon VPC and IVS using interface VPC endpoints powered by AWS PrivateLink. See IVS Private Ingest.	✓		
Progressive video	Transmits and displays all lines of each frame in sequence. We recommend using progressive video during all stages of a broadcast.	✓	✓	

Term	Description	LL	RT	Chat
Publisher	A real-time event participant who publishes video and/or audio to a stage. See What is IVS Real-Time Streaming.		✓	
Quotas	The maximum numbers of IVS service resources or operations for your AWS account. That is, these limits are per AWS account, unless noted otherwise. All quotas are enforced per region. See Amazon Interactive Video Service endpoints and quotas in *AWS General Reference Guide*.	✓	✓	✓
Regions	Provide access to AWS services that physicall y reside in a specific geographic area. Regions provide fault tolerance, stability, and resilience, and can also reduce latency. With Regions, you can create redundant resources that remain available and unaffected by a regional outage. Most AWS service requests are associated with a particular geographic region. The resources that you create in one region do not exist in any other region unless you explicitly use a replication feature offered by an AWS service. For example, Amazon S3 supports cross-region replication. Some services, such as IAM, do not have cross-reg ional resources.	✓	✓	✓
Resolution	Describes the number of pixels in a single video frame, for example, Full HD or 1080p defines a frame with 1920x1080 pixels.	✓	✓	
Root user	The owner of an AWS account. The root user has complete access to all AWS services and resources in the AWS account.	✓	✓	✓

Term	Description	LL	RT	Chat
RTMP, RTMPS	Real-Time Messaging Protocol, an industry standard for transmitting audio, video, and data over a network. RTMPS is the secure version of RTMP, running over a Transport Layer Security (TLS/SSL) connection.	✓	✓	
S3 bucket	A collection of objects stored in Amazon S3. Many policies, including access and replication, are defined at the bucket level and apply to all objects in the bucket. For example, an IVS broadcast is stored as multiple objects in an S3 bucket.	✓		
SDK	Software Development Kit, a collection of libraries for the developers building applications with IVS. The IVS *Player SDK* is for playback of IVS streams. It leverages the IVS architecture and is optimized for IVS low-latency playback. There are IVS player SDKs for web, Android, and iOS. The IVS *Broadcast SDK* is for developers who are building applications with IVS. This SDK leverages the IVS architecture and is continually improved, alongside IVS. As a native broadcast SDK, it is designed to minimize the performance impact on your applications and on the devices with which users access applications. There are IVS broadcast SDKs for web, Android, and iOS, for low-latency and real-time streaming.	✓	✓	✓

Term	Description	LL	RT	Chat
Selfie segmentation	Enables replacing the background in a live stream, using a client-specific solution that accepts a camera image as input and returns a mask that provides a confidence score for each pixel of the image, indicating whether it is in the foreground or the background. See Background Replacement in *IVS Broadcast SDK: Third-Party Camera Filters (Real-Time Streaming)*.		✓	
Semantic versioning	A version format in the form of Major.Minor.Patch. Bug fixes not affecting the API increment the patch version, backward compatible API additions /changes increment the minor version, and backward incompatible API changes increment the major version.	✓	✓	✓
Server-side composition	Uses an IVS server to mix audio and video from stage participants and then sends this mixed video to an IVS channel to reach a larger audience or to store it in an S3 bucket. Server-side composition reduces client load, improves resilience of the broadcast, and enables more efficient use of bandwidth. Also see client-side composition.		✓	
Service quotas	An AWS service that helps you manage your quotas for many AWS services from one location. Along with looking up the quota values, you can also request a quota increase from the Service Quotas console.	✓	✓	✓

Term	Description	LL	RT	Chat
Service-linked role	A unique type of IAM role that is linked directly to an AWS service. Service-linked roles are automatically created by IVS and include all the permissions that the service requires to call other AWS services on your behalf, for example, to access an S3 bucket. See Using Service-Linked Roles for IVS in *IVS Security*.	✓		
Stage	An IVS resource that represents a virtual space where real-time event participants can exchange video in real time. See Create a Stage with Optional Participant Recording in *Getting Started with IVS Real-Time Streaming*.		✓	
Stage session	Begins when the first participant joins a stage and ends a few minutes after the last participant stops publishing to the stage. A long-lived stage may have multiple sessions over its lifetime.		✓	
Stream	Data representing video or audio content being sent continuously from a source to a destination.	✓	✓	
Stream key	An identifier assigned by IVS when you create a channel; it is used to authorize streaming to the channel. **Treat the stream key like a secret, since anyone with it can stream to the channel.** See Getting Started with IVS Low-Latency Streaming.	✓		

Term	Description	LL	RT	Chat
Stream starvation	A delay or halt in stream delivery to IVS. It occurs when IVS does not receive the expected amount of bits that the encoding device advertise d it would send over a certain timeframe. An occurrence of stream starvation results in a stream starvation event. From a viewer's perspective, stream starvatio n may appear as video that lags, buffers, or freezes. Stream starvation can be brief (less than 5 seconds) or long (several minutes), depending on the specific situation that resulted in stream starvation. See What is Stream Starvation in *Troubleshooting FAQ*.	✓	✓	
Streamer	A person or a device sending a video or audio stream to IVS.	✓	✓	
Subscriber	A real-time event participant who receives video and/or audio of stage publishers. See What is IVS Real-Time Streaming.		✓	
Tag	A metadata label that you assign to an AWS resource. Tags can help you identify and organize your AWS resources. On the IVS documentation landing page, see “Tagging” in any of the IVS API documentation (for real-time streaming, low-laten cy streaming, or chat).	✓	✓	✓
Third-party camera filters	Software components that can be integrated with the IVS Broadcast SDK to allow an application to process images before providing them to the Broadcast SDK as a custom image source. A third-party camera filter may process images from the camera, apply a filter effect, etc.	✓	✓	

Term	Description	LL	RT	Chat
Thumbnail	A reduced-size image taken from a stream. By default, thumbnails are generated every 60 seconds, but a shorter interval can be configure d. Thumbnail resolution depends on the channel type. See Recording Contents in *Auto-Record to Amazon S3 (Low-Latency Streaming)*.	✓		
Timed metadata	Metadata tied to specific timestamps within a stream. It can be added programmatically using the IVS API and becomes associated with specific frames. This ensures that all viewers receive the metadata at the same point relative to the stream. Timed metadata can be used to trigger actions on the client such as updating team statistics during a sporting event. See Embedding Metadata within a Video Stream.	✓		
Token exchange	An interface provided by the IVS Broadcast SDK that allows upgrading or downgrading participant-token capabilities and updating token attributes, without requiring participants to reconnect. This enables scenarios like co-hosting, where participants may start with subscribe-only capabilities and later need publish capabilities.		✓	
Transcoding	Converts video and audio from one format to another. An incoming stream may be transcode d to a different format at multiple bitrates and resolutions, to support a range of playback devices and network conditions.	✓	✓	

Term	Description	LL	RT	Chat
Transmuxing	A simple repackaging of an ingested stream to IVS, with no re-encoding of the video stream. “Transmux” is short for transcode multiplexing, a process that changes the format of an audio and/or video file while keeping some or all of the original streams. Transmuxing converts to a different container format without changing the file contents. Distinguished from transcoding.	✓	✓	
Variant streams	A set of encodings of the same broadcast in several distinct quality levels. Each variant stream is encoded as a separate HLS playlist. An index of the available variant streams is referred to as a multivariant playlist. After the IVS player receives a multivariant playlist from IVS, it can then choose between the variant streams during playback, changing back and forth seamlessly as network conditions change.	✓		
VBR	Variable Bitrate, a rate-control method for encoders that uses a dynamic bitrate that changes throughout playback, depending on the level of detail needed. We strongly recommend against using VBR due to video-quality concerns; use CBR instead.	✓	✓	

Term	Description	LL	RT	Chat
View	A unique viewing session which is actively downloading or playing media files. Views are the basis for the concurrent views quota. A view starts when a viewing session begins video playback. A view ends when a viewing session stops video playback. Playback is the sole indicator of viewership; engagement heuristic s such as audio levels, browser tab focus, and video quality are not considered. When counting views, IVS does not consider the legitimacy of individual viewers or try to deduplicate localized viewership, such as multiple video players on a single machine. See Other Quotas in *Service Quotas (Low-Latency Streaming)*.	✓		
Viewer	A person receiving a stream from IVS.	✓		
WebRTC	Web Real-Time Communication, an open-sour ce project providing web browsers and mobile applications with real-time communication. It allows audio and video communication to work inside web pages by allowing direct peer-to-peer communication, eliminating the need to install plugins or download native apps. The technologies behind WebRTC are implement ed as an open web standard and are available as regular JavaScript APIs in all major browsers or as libraries for native clients, like Android and iOS.	✓	✓	

Term	Description	LL	RT	Chat
WHIP	WebRTC-HTTP Ingestion Protocol, an HTTP based protocol that allows WebRTC based ingestion of content into streaming services and/or CDNs. WHIP is an IETF draft developed to standardize WebRTC ingestion. WHIP enables compatibility with software like OBS, offering an alternative (to the IVS broadcast SDK) for desktop publishing. More sophisticated streamers familiar with OBS may prefer it for its advanced production features, such as scene transitions, audio mixing, and overlay graphics WHIP is also beneficial in situations where using the IVS broadcast SDK isn't feasible or preferred . For example, in setups involving hardware encoders, the IVS broadcast SDK might not be an option. However, if the encoder supports WHIP, you can still publish directly from the encoder to IVS. See IVS WHIP Support (Real-Time Streaming).		✓	
WSS	WebSocket Secure, a protocol for establishing WebSockets over an encrypted TLS connection. It is being used for connecting to IVS Chat endpoints . See Step 4: Send and Receive Your First Message in *Getting Started with IVS Chat*.			✓

IVS Document History | Real-Time Streaming

The following tables describe the important changes to the documentation for Amazon IVS Real-Time Streaming. We update the documentation frequently, for new releases and to address the feedback that you send us.

Real-Time Streaming User Guide Changes

Change	Description	Date
IVS Broadcast SDK Guide	Removed known issues and workarounds from SDK documentation.	August 31, 2026
Cost Allocation Tags	Added information about using cost allocation tags with Amazon IVS. See Cost Allocation Tags.	August 27, 2026
Broadcast SDK: Web 1.39.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	August 27, 2026
Broadcast SDK: Android 1.46.0, iOS 1.46.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	August 27, 2026
Connection Reuse Across Stages	Updated Getting Started - Step 3: Distribute Tokens and added a new section in Streaming Optimizations on reducing time to video when switching between stages for	August 19, 2026

	`RealTimeConnection` . Also see the Release Notes.	
Broadcast SDK: Android 1.43.2	Added SDK patch 1.43.2 to the Release Notes.	August 19, 2026
Broadcast SDK: Web 1.38.1	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	August 12, 2026
Broadcast SDK: Android 1.43.1	Added SDK patch 1.43.1 to the Release Notes.	August 11, 2026
Broadcast SDK: Web 1.38.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	July 30, 2026
Broadcast SDK: Android 1.45.0, iOS 1.45.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	July 30, 2026
Broadcast SDK: iOS 1.44.1	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: iOS. Also see the Release Notes.	July 7, 2026
Broadcast SDK: Web 1.37.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	July 2, 2026

Broadcast SDK: Android 1.44.0, iOS 1.44.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	July 2, 2026
EventBridge	For Participant Published and Participant Unpublished events, added `event_time_precise` field with millisecond precision.	June 29, 2026
Broadcast SDK: Web 1.36.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	June 4, 2026
Broadcast SDK: Android 1.43.0, iOS 1.43.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	June 4, 2026
Broadcast SDK: Web 1.35.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	May 7, 2026
Broadcast SDK: Android 1.42.0, iOS 1.42.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	May 7, 2026

RT token exchange: Web broadcast SDK & SSC changes	Updated Broadcast SDK: Token Exchange and Server-Side Composition to reflect token-exchange support across all SDKs (not just mobile) and server-side composition.	April 16, 2026
Broadcast SDK: Web 1.34.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	April 9, 2026
Broadcast SDK: Android 1.41.0, iOS 1.41.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	April 9, 2026
Redundant ingest 24/7 streaming	Added Redundant Ingest to *IVS RTMP Publishing*. API changes are described in the API Reference table.	April 8, 2026
Broadcast SDK: Web 1.33.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	March 12, 2026
Broadcast SDK: Android 1.40.0, iOS 1.40.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	March 12, 2026

EventBridge	For four events (Destination Failure, Session Failure, Recording Start Failure, Recording End Failure), we updated descriptions and added the `error_code` field and a table of error codes and reasons.	February 27, 2026
Broadcast SDK: Android 1.39.0, iOS 1.39.0	Updated version number and artifact links in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. For iOS integration, CocoaPods is no longer used. Related documentation changes were made in: • Getting Started with IVS Real-Time Streaming – "Step 4: Integrate the IVS Broadcast SDK" > "iOS" • iOS Broadcast SDK Guide – "Install the Library"	February 13, 2026
Broadcast SDK: Web 1.32.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	February 12, 2026
Service Quotas	Added TPS values for the three ParticipantReplication operations (List/Start/Stop).	January 30, 2026

Broadcast SDK: Android 1.38.0, iOS 1.38.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	January 13, 2026
Broadcast SDK: Android 1.37.1	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android. Also see the Release Notes.	December 11, 2025
Broadcast SDK – Custom Audio Sources	Added this new page.	December 10, 2025
Participant token exchange	We added a new Token Exchange page under *IVS Broadcast SDK*. In Using Amazon EventBridge with IVS, we added an IVS Stage Update event, Token Exchanged. We also added a Token Exchanged example in Examples: Stage Update. API changes are described in the API Reference table.	December 9, 2025
Broadcast SDK: Web 1.31.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	December 5, 2025

Broadcast SDK: Android 1.37.0, iOS 1.37.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	December 5, 2025
CloudWatch metrics	In Monitoring Real-Time Streaming > CloudWatch Metrics, we added many new DroppedFrames, PublishBi trate, and SubscribeBitrate metrics. We also updated some metrics descriptions.	November 18, 2025
IPR synchronization	In *Individual Participa nt Recording*, we added Synchronize Multiple Participa nt Recordings.	November 7, 2025
Server-Side Composition	Added Known Issues and Workarounds.	October 30, 2025
Broadcast SDK: Web 1.30.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	October 30, 2025
Broadcast SDK: Android 1.36.0, iOS 1.36.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. We also added new sections on "Embed Messages": Android and iOS.	October 30, 2025

Server-Side composition	Updated Composition Lifecycle (when the compositi on performs auto-shutdown).	October 27, 2025
RTMP	Added Publishing with FFmpeg.	October 27, 2025
Update quota for Compositi ons	In Service Quotas > Other Quotas, we updated the quota for "maximum concurrent Composition resources per account" from 5 to 20.	October 14, 2025
Web Broadcast SDK update	Rewrote the section on Publishing & Subscribing with the IVS Web Broadcast SDK > Get WebRTC Statistics.	October 3, 2025
CloudWatch Metrics	For the `ConcurrentPublishers` and `ConcurrentSubscriptions` metrics, updated the description and deleted the dimension.	October 2, 2025
Broadcast SDK: Web 1.29.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	October 2, 2025
Broadcast SDK: Android 1.35.0, iOS 1.35.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	October 2, 2025

CloudWatch Metrics	Added several more DownloadPacketLoss metrics.	September 23, 2025
SSC custom participant ordering	Made various changes to Server-Side Composition, including adding `participantOrderAttribute` and "Custom Participant Ordering. " API changes are described in the API Reference table.	September 16, 2025
Broadcast SDK: Android 1.34.0, iOS 1.34.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	September 11, 2025
Private Ingest to Stages	New section for the release of interface VPC endpoints.	September 10, 2025
Broadcast SDK: Web 1.28.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	September 4, 2025
Broadcast SDK: iOS	Added a 1.32.1 iOS broadcast SDK fix to the most recent Release Notes for iOS broadcast SDK (1.33.0).	August 21, 2025
Broadcast SDK: Web	Updates in Getting Started > Imports, both Using a Script Tag and Using npm.	August 8, 2025

Broadcast SDK: Web 1.27.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	August 7, 2025
Broadcast SDK: Android 1.33.0, iOS 1.33.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. In the iOS Broadcast Guide, we added "Recommended: Integrate the Player SDK (Swift Package Manager)" and updated the existing information on CocoaPods integration.	August 7, 2025
Broadcast SDK: Mixed Devices	This is a new document. (*Mixed Devices* is identical in both the IVS Low-Latency Streaming User Guide and the IVS Real-Time Streaming User Guide.)	July 28, 2025
Broadcast SDK: Android 1.32.2	Updated version number and artifact links in the real-time -streaming broadcast SDK guide: Android. Also see the Release Notes.	July 25, 2025
Add limit for concurrent participant replications	In Service Quotas > Other Quotas, we added a quota for "concurrent participant replications."	July 15, 2025

Broadcast SDK: Android 1.32.1, iOS 1.32.1	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	July 10, 2025
Broadcast SDK: Web 1.26.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. Also: • In Handling Network Issues, we updated the example. • In StageErrors, we added information about the FAILED error and we added STAGE_DISCONNECTED and PARTICIPANT_DISCON NECTED errors.	July 7, 2025

Add limits for concurrent publishers and subscriptions	In Service Quotas > Other Quotas, we added quotas for "concurrent publishers" and "concurrent subscriptions." In Monitoring Real-Time Streaming > CloudWatch Metrics, we added metrics for ConcurrentPublishers and ConcurrentSubscriptions. In the *Broadcast Web SDK Guide*, we updated the STAGE_AT_CAPACITY table entry in Error Handling > Stage Errors.	June 23, 2025

E-RTMP multitrack video ingest support	In Stream Ingest > Supported Protocols, we added support for E-RTMP (Enhanced RTMP) multitrack video. In IVS RTMP Publishing, we added information about streaming E-RTMP multitrack video using the real-time-streaming broadcast SDKs and OBS Studio. In Monitoring Real-Time Streaming > CloudWatch Metrics, we added a PublishFramerate metric with the Stage, Participant, Simulcast Layer, and MediaType dimensions. We also updated SimulcastLayer dimension values ("no-rid" and "disabled" are replaced with "none").	June 20, 2025
SSE-S3 encryption	We added new information to: • Individual Participant Recording – In 1. Create an S3 Bucket • Server-Side Composition – In Getting Started with IVS Server-Side Composition > Prerequisites	June 19, 2025

Broadcast SDK: Web 1.25.1	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	June 16, 2025
Broadcast SDK: Web 1.25.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	June 12, 2025
Broadcast SDK: Android 1.31.0, iOS 1.31.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	June 12, 2025
B-frames in stream ingest	In Stream Ingest > Supported Media Specifications, we updated the information on B-frames.	May 30, 2025

Participant replication	Initial release of this new functionality. See these documentation changes: • Getting Started with Amazon IVS – In Step 2: Create a Stage with Optional Participant Recording, updated console instructions and screenshots and added `recordParticipantReplicas` to the CLI response for creating a stage with individual participant recording. • Monitoring Real-Time Streaming – In CloudWatch Metrics: IVS Real-Time Streaming, added a note about participant replication. • EventBridge – In Examples: Stage Update, added two events, Participant Replication Start and Participant Replication End. Also updated Participant Published, Participant Unpublished, and Participant Publish Error. • Participant Replication – This new document includes an overview and CLI instructions.	May 29, 2025

	• Costs – This new document includes costs associate d with IVS real-time streaming. API changes are described in the API Reference table.	
Broadcast SDK: Android 1.30.1	Updated version number and artifact links in the real-time -streaming broadcast SDK guide: Android. Also see the Release Notes.	May 26, 2025
Broadcast SDK: Web 1.24.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	May 15, 2025
Broadcast SDK: Android 1.30.0, iOS 1.30.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	May 15, 2025
Broadcast SDK: Web 1.23.1	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	May 2, 2025

Broadcast SDK: Web 1.23.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. We also added additional information and examples to "Configuring Layered Encoding (Publisher)" in the Web Broadcast SDK Guide.	April 17, 2025
Broadcast SDK: Android 1.29.0, iOS 1.29.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. We also added additional information and examples to "Configuring Layered Encoding (Publisher)" in the Android and iOS Broadcast SDK Guides.	April 17, 2025
Service Quotas	Added a quota for "Compositions per stage."	April 2, 2025
Streaming Optimizations	In the introduction, added a paragraph on configuring maximum bitrate, framerate, resolution, and (on mobile) degradation preference.	March 21, 2025

Broadcast SDK: Web 1.22.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. Also, in the introduction, we added another Sample Code example (Simple Playback). In "Supplemental Enhancement Information (SEI) > Inserting SEI Payloads," we added a note about memory usage. And in "Known Issues & Workarounds," we added an item about `stage.leave()` .	March 20, 2025
Broadcast SDK: Android 1.28.1, iOS 1.28.1	Updated version number and artifact links in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	March 20, 2025
Broadcast SDK: Android 1.27.2, iOS 1.27.2	Updated version number and artifact links in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	March 19, 2025

Target Segment Duration	Updated some screensho ts in "Getting Started with IVS Real-Time Streaming > Step 2: Create a Stage with Optional Participant Recording" to show the new “Target segment duration” field.	March 13, 2025
Individual participant recording	Added Converting Recordings to MP4.	March 7, 2025
Individual participant recording stitching	Initial release of this new functionality. See these documentation changes: • Getting Started with Amazon IVS – Updated console and CLI instructions in Step 2: Create a Stage with Optional Participant Recording. • Individual Participant Recording – Added Merge Fragmented Individual Participant Recordings. • EventBridge – In Examples: Individual Participant Recording State Change, added information about S3 prefix construction when merging is enabled for Individual Participant Recording.	March 6, 2025

WHIP requirements	In the introductory text, added a requirement for WHIP clients to handle 307 redirects.	March 5, 2025
Broadcast SDK: iOS 1.27.1	Updated version number and artifact links in the real-time-streaming broadcast SDK guide: iOS. Also see the Release Notes.	March 3, 2025
Broadcast SDK: Web 1.21.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	February 20, 2025
Broadcast SDK: Android 1.27.0, iOS 1.27.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	February 20, 2025
Broadcast SDK: Android 1.26.0, iOS 1.26.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	January 30, 2025
Broadcast SDK: Web 1.20.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. We also updated "Suppleme ntal Enhanced Information."	January 23, 2025

RTMP Publishing	In Publish Using an RTMP Encoder, we noted that streams must include both audio and video tracks or they will be disconnected.	January 21, 2025
Network Requirements	Added this new top-level page.	January 21, 2025
Streaming Optimizations	Renamed "Configuring Layered Encoding with Simulcast" to "Configuring Layered Encoding with Simulcast (Publisher)" and added "Configuring Layered Encoding with Simulcast (Subscriber)."	December 12, 2024
Broadcast SDK: Web 1.19.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. We also added "Layered Encoding with Simulcast" and added three simulcast items in "Events."	December 12, 2024

Broadcast SDK: Android 1.25.0, iOS 1.25.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. In each guide, we also: • Added "Get Supplemental Enhancement Information." • Added "Layered Encoding with Simulcast." • Added three simulcast items in "Renderer." • Deleted "Enable/Disable Layered Encoding with Simulcast."	December 12, 2024
Real-time thumbnail configuration	In Individual Participant Recording and Composite Recording, updated examples and JSON metadata informati on, and added pricing information. In Individual Participant Recording, added "Thumbnail-Only Recordings."	December 10, 2024
Broadcast SDK: Android 1.24.0, iOS 1.24.0	Updated version number and artifact links in the real-time -streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	November 13, 2024
Third-Party Camera Filters	Many changes in Using Snap with the IVS Broadcast SDK > Web.	November 12, 2024

Broadcast SDK: Web 1.18.0	Updated version number and artifact links in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. We added a new section, Get Supplemental Enhancement Information (SEI), to the SDK Guide.	November 12, 2024
RTMP	In "Create an Ingest Configuration," updated example (added `--ingest-protocol`).	November 7, 2024
Broadcast SDK: Web 1.17.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	October 10, 2024
Broadcast SDK: Android 1.23.0, iOS 1.23.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. For Android, we added Using the SDK with Debug Symbols.	October 10, 2024
Service Quotas	We added a quota for "Participant publish bitrate."	September 25, 2024

Monitoring IVS Real-Time Streaming	Added the `PublishFramerate` CloudWatch metric.	September 13, 2024
Broadcast SDK: Web 1.16.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	September 11, 2024
Broadcast SDK: Android 1.22.0, iOS 1.22.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. We also updated a section, "Getting Started > Install the Library" in the Android broadcast SDK guide.	September 11, 2024

RTMP ingest	We added an IVS Stream Ingest page. Under this are two pages, RTMP (new) and WHIP. In Using EventBridge with IVS Real-Time Streaming, we added an IVS Stage Update event, Participant Publish Error. In Service Quotas, we added TPS values for the five new API operations and 1 new IngestConfiguration quota (in "Other Quotas"). API changes are described in the API Reference table.	September 9, 2024
Real-Time Streaming Optimizations	Made various simulcast-related updates and added "Resolution of Layers."	August 22, 2024
In-console publish/subscribe	In *Getting Started with IVS Real-Time Streaming*, we added in-console publishin g and subscribing to Publish and Subscribe to Video.	August 19, 2024

Broadcast SDK: Web 1.15.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. We also added a new section, Configuration for Subscribing to Participants, to the *Web Broadcast SDK Guide*. In *Streaming Optimizations*, we added a new section, Changing Subscriber Jitter Buffer MinDelay. This includes information about the Web, Android, and iOS Broadcast SDKs.	August 15, 2024
Broadcast SDK: Android 1.21.0, iOS 1.21.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes. We also added a new section, "Configuration for Subscribing to Participants," to the Android and iOS *Broadcast SDK Guides*.	August 15, 2024

Recording clarification	Added a note about using an existing S3 bucket, in *Individual Participant Recording* (in 1: Create an S3 Bucket) and *Composite Recording* (in Prerequisites, Step 3). The **Object Ownership** setting must be **Bucket owner enforced** or **Bucket owner preferred**.	August 13, 2024
Broadcast SDK: Web 1.14.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes.	July 18, 2024
Broadcast SDK: Android 1.20.0, iOS 1.20.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	July 18, 2024
Getting Started with Real-Time Streaming	Added information on attributes to "Distribute Participant Tokens," in both "Token Schema: Payload" and "Creating Tokens with the IVS Real-Time Streaming API."	July 12, 2024
Service Quotas	Increased the maximum number of stage subscribers from 10,000 to 25,000.	June 27, 2024

Generate Participant Tokens with a Key Pair	In *Getting Started with IVS Real-Time Streaming*, we updated Distribute Participant Tokens to explain the two ways to generate tokens (API and key pair) and we added "Creating Tokens with a Key Pair."	June 26, 2024
Individual participant recording	Added a new documentation section on Recording, with sub-documents on Individual Participant Recording (new) and Composite Recording (pre-existing). We also added Participant Recording State Change events and examples to Using EventBridge with IVS. API changes are described in the API Reference table.	June 20, 2024
Service Quotas	Increased the Stages quota from 100 to 1,000.	June 14, 2024

Broadcast SDK: Web 1.13.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. In the guide, we updated the information in Error Handling for the new ERROR stage event.	June 13, 2024
Broadcast SDK: Android 1.19.0, iOS 1.19.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	June 13, 2024
Broadcast SDK: Web 1.12.0	Updated version number and artifact links on the IVS documentation landing page and in the low-latency-streaming broadcast SDK guide: Web. Also see the Release Notes. In the guide, we updated the information in Handling Network Issues about the stage-connection ERRORED state.	May 20, 2024

Real-Time Streaming Optimizations	In Default Layers, Qualities, and Framerates, changed the max bitrate for Mobile, Low Layer, from 150,000 to 100,000 bps.	May 16, 2024
Broadcast SDK: Android 1.18.0, iOS 1.18.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	May 16, 2024
Broadcast SDK: Web 1.11.0	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	May 6, 2024
Broadcast SDK: Web 1.10.1	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guide: Web. Also see the Release Notes.	April 30, 2024
Broadcast SDK: Android 1.15.2, iOS 1.15.2	Updated version number and artifact links on the IVS documentation landing page and in the real-time-streaming broadcast SDK guides: Android and iOS. Also see the Release Notes.	April 30, 2024

Broadcast SDK: iOS Guide	In Publish a Media Stream, we updated the code example.	April 26, 2024
Broadcast SDK: Android 1.17.0, iOS 1.17.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	April 22, 2024
Server-side composition	In SSC, made various changes, especially in "Layout," to explain PiP and grid layouts. In the Web Broadcast SDK Guide, added Server-Side Rendering Support.	March 26, 2024
OBS and WHIP Support	Added a note about quality issues (like intermittent video freezing) that can occur with WHIP in OBS.	March 22, 2024

Broadcast SDK: Android 1.16.0, iOS 1.16.0, Web 1.10.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android, iOS, and Web. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	March 21, 2024
Broadcast SDK: Android 1.15.1, iOS 1.15.1	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	March 13, 2024

Broadcast SDK: Mobile Audio Modes	In "Audio Mode Presets," added information on the Volume Rocker preset category and an iOS known issue with the Video Chat preset. In "Advanced Use Cases," added a note on avoiding incorrect configurations, and added sections on "iOS Echo Cancellation" and "iOS Custom Audio Sources."	March 1, 2024
Broadcast SDK: Android 1.15.0, iOS 1.15.0, Web 1.9.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android, iOS, and Web. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	February 22, 2024
OBS and WHIP Support	Added a new page. This document explains how to use WHIP-compatible encoders like OBS to publish to IVS real-time streaming . WHIP (WebRTC-HTTP Ingestion Protocol) is an IETF draft developed to standardize WebRTC ingestion.	February 6, 2024

Broadcast SDK: Android 1.14.1, iOS 1.14.1, Web 1.8.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android, iOS, and Web. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release. For the Android Guide, we added a new Known Issue (video size less than 176x176). For the Web Guide, we added a new Known Issue. The workaround is constraining video resolution to 720p when invoking `getUserMedia` or `getDisplayMedia`. In *Real-Time Streaming Optimizations* we updated Configuring Layered Encoding with Simulcast; now this is disabled by default.	February 1, 2024

Broadcast SDK: Android 1.13.4, iOS 1.13.4, Web 1.7.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android, iOS, and Web. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	January 3, 2024
IVS Glossary	Extended the glossary, covering IVS real-time, low-latency, and chat terms.	December 20, 2023
Stage Health: New CloudWatch Metrics	Renamed the PacketLoss (Stage) metric to be DownloadPacketLoss (Stage) and released additional CloudWatch metrics for IVS real-time streaming: • DownloadPacketLoss (Stage,Participant) • DroppedFrames (Stage,Participant) • SubscribeBitrate (Stage,Participant,MediaType) See Monitoring IVS Real-Time Streaming.	December 7, 2023

IAM managed policies	Added two managed policies, IVSReadOnlyAccess and IVSFullAccess. See: • The new section on Managed Policies for Amazon IVS on the *Security* page. • Changes to Step 3: Set Up IAM Permissions in *Getting Started with IVS Low-Latency Streaming*.	December 5, 2023
Broadcast SDK: Android 1.13.2, iOS 1.13.2	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	December 4, 2023

Broadcast SDK: Android 1.13.1	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guide: Android. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	November 21, 2023
Service Quotas	Changed "Participant publish resolution" from 1080p to 720p.	November 18, 2023

Broadcast SDK: Android 1.13.0, iOS 1.13.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Android and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release. We also made various updates to Streaming Optimizations. Among other things, the "Adaptive Streaming: Layered Encoding with Simulcast" feature now requires explicit opt-in and is supported only in recent versions of the SDK.	November 17, 2023

Server-side composition (SSC)	IVS server-side composition enables clients to offload the composition and broadcasting of an IVS stage to an IVS-managed service. SSC and RTMP broadcast to a channel are invoked through IVS control-plane endpoints in the stage's home region. See: • Getting Started – We added SSC endpoints to the policy in "Set Up IAM Permissions." • Using Amazon EventBridge with IVS – We added new metrics. • Server-Side Composition – This new document includes an overview and setup instructions. • Service Quotas – We added new call-rate limits and other quotas. Also see: • Changes listed below in IVS Real-Time Streaming API Reference Changes. • Changes listed in Document History (Low-Latency Streaming).	November 16, 2023

Composite Recording	Made the following changes: • Added a Composite Recording page for this new feature. • Updated Getting Started with IVS Real-Time Streaming with S3 endpoints in the policy in "Set Up IAM Permissions." • Updated Service Quotas with call-rate quotas for the new endpoints.	November 16, 2023
IVS broadcast SDK	In the Broadcast SDK overview, we updated Platform Requirements > Native Platforms to clarify which SDK versions are supported and we added "Mobile Browsers (iOS and Android)." In the Broadcast Web Guide, we added "Mobile Web Limitations."	November 9, 2023
IVS broadcast SDK	We added a new page on Third-Party Camera Filters.	November 9, 2023
Getting Started with IVS Real-Time Streaming	We updated procedures in Set Up IAM Permissions.	October 20, 2023
Monitoring Real-Time Streaming	In CloudWatch Metrics: IVS Real-Time Streaming, we added sample values for dimensions.	October 17, 2023

Broadcast SDK: Web Guide	We made several changes to Monitor Remote Participant Media Mute State.	October 17, 2023
Broadcast SDK: Web 1.6.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guide: Web. The Amazon IVS documentation landing page points to the current version of Broadcast SDK References. Also see the Amazon IVS Release Notes for this release. In the Web Guide, in "Retrieve a MediaStream from a Device," we also deleted the two `max` lines; best practice is to specify only `ideal`. In Real-Time Streaming Optimizations, we added a new section, Optimizing Audio Bitrate and Stereo Support.	October 16, 2023
Stage Health: New CloudWatch Metrics	Released CloudWatch metrics for IVS real-time streaming. See Monitoring IVS Real-Time Streaming.	October 12, 2023

Broadcast SDK: Android 1.12.1	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guide: Android. Also added a new section, Using Bluetooth Microphones. The Amazon IVS documentation landing page points to the current version of Broadcast SDK References. Also see the Amazon IVS Release Notes for this release.	October 12, 2023
Broadcast SDK: Web 1.5.2	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guide: Web. The Amazon IVS documentation landing page points to the current version of Broadcast SDK References. Also see the Amazon IVS Release Notes for this release.	September 14, 2023
Getting Started with IVS Real-Time Streaming	In Android > Install the Broadcast SDK, added data binding.	September 12, 2023
Broadcast SDK error handling	Added "Error Handling" sections to the Broadcast SDK Guides: Web, Android, and iOS.	September 12, 2023

Getting Started with IVS Real-Time Streaming	In Distribute Participant Tokens, added an **Important** note about not building functionality based on current token format.	September 1, 2023
Getting Started with IVS Real-Time Streaming	In Set Up IAM Permissions, updated the set of permissions.	August 31, 2023
Broadcast SDK: Web 1.5.1, Android 1.12.0, and iOS 1.12.0	Updated version number and artifact links for the new release, in the real-time-streaming broadcast SDK guides: Web, Android, and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	August 23, 2023

Real-time streaming launch	Major documentation changes accompany this release. We renamed the previous documentation to be IVS Low-Latency Streaming and published new IVS Real-Time Streaming documentation. The IVS documentation landing page now has separate sections for real-time streaming and low-latency streaming. Each section has its own User Guide and API Reference. For other documentation changes, see Document History (Low-Latency Streaming).	August 7, 2023
Broadcast SDK: Web 1.5.0, Android 1.11.0, and iOS 1.11.0	Updated version number and artifact links for the new release, in the broadcast SDK guides: Web, Android, and iOS. On the Amazon IVS documentation landing page, updated the broadcast SDK Reference links to point to the new version. Also see the Amazon IVS Release Notes for this release.	August 7, 2023

IVS Real-Time Streaming API Reference Changes

API Change	Description	Date
Redundant ingest 24/7 streaming	Modified the IngestConfiguration object (added the `redundantIngest` and `redundantIngestCredentials` fields). This affects CreateIngestConfiguration, GetIngestConfiguration, and UpdateIngestConfiguration requests and responses. Modified the IngestConfigurationSummary object (added the `redundantIngest` field). This affects the ListIngestConfigurations response. Modified the Participant object (added the `redundantIngest` and `ingestConfigurationArn` fields). This affects the GetParticipant response. User Guide changes are described in the User Guide table.	April 8, 2026
Update StartParticipantReplication operation	In StartParticipantReplication, changed the maximum value for the `reconnectWindowSeconds` field from 60 to 300.	March 16, 2026
Participant token exchange	Added the ExchangedParticipantToken object. Added `newToken` and `previousToken` fields to the Event object. Added `TOKEN_EXCHANGED` as a valid value for the `name` field in the Event object. User Guide changes are described in the User Guide table.	December 9, 2025
SSC custom participant ordering	Modified the GridConfiguration and PipConfiguration data types (added the `participa`	September 16, 2025

API Change	Description	Date
	`ntOrderAttribute` field), which in turn affects the Composition object. This affects the StartComposition request and response and the GetComposition response. User Guide changes are described in the User Guide table.	

API Change	Description	Date
Participant replication	Initial release of this new functionality. See these documentation changes: • Added terminology specific to participant replicati on in “Key Concepts.” • Added three new operations: ListParticipantRep licas, StartParticipantReplication, StopParti cipantReplication. • Added one new object, ParticipantReplica. • Added the `recordParticipantReplicas` field to the AutoParticipantRecordingConfiguratio n object. This affects the CreateStage request and response, GetStage response, and UpdateStage request and response. • Added three fields to the Event object: `destinationStageArn` , `destinati onSessionId` , `replica`. This affects the ListParticipantEvents response. • Added four fields to Participant and Participa ntSummary objects: `replicationState` , `replicationType` , `sourceStageArn` , `sourceSessionId` . This affects the GetPartic ipant and ListParticipants responses. • In Event, added two valid values to the `name` field: `REPLICATION_STARTED` , `REPLICATI ON_STOPPED` .	May 29, 2025

API Change	Description	Date
	User Guide changes are described in the User Guide table.	
Target Segment Duration	Modified the AutoParticipantRecordingConfiguration object (added the `hlsConfiguration` field); this in turn affects the Stage object. This affects the CreateStage request and response, GetStage response, and UpdateStage request and response. Modified the RecordingConfiguration object (added `hlsConfiguration` field). This affects the GetComposition response and StartComposition request and response. Added two objects: CompositionRecordingHlsConfiguration and ParticipantRecordingHlsConfiguration.	March 13, 2025
Individual participant recording stitching	Modified the AutoParticipantRecordingConfiguration object (added the `recordingReconnectWindowSeconds` field); this in turn affects the Stage object. This affects the CreateStage request and response, GetStage response, and UpdateStage request and response. Updated the description of `Participant.recordingS3Prefix`.	March 6, 2025

API Change	Description	Date
Real-time thumbnail configuration	Modified the S3DestinationConfiguration object: added `thumbnailConfigurations` . This affects the GetComposition response and StartComposition request and response. Modified the AutoParticipantRecordingConfiguration object: added `thumbnailConfiguration` and added `NONE` as a valid value for `mediaTypes` . This affects the CreateStage request and response, GetStage response, and UpdateStage request and response. Added two objects: CompositionThumbnailConfiguration and ParticipantThumbnailConfiguration.	December 10, 2024
Update Event and Video objects	In the Event object, we added more valid values for `errorCode` . In the Video object, we clarified that `height` and `width` must be even numbers.	October 2, 2024

API Change	Description	Date
RTMP ingest	We added two objects: IngestConfiguration and IngestConfigurationSummary. We added five IngestConfiguration endpoints (Create, Delete, Get, List, and Update). We updated DeleteStage (description of the operation) and DisconnectParticipant (descriptions of the operation and `participantId`). We modified the Participant object (added the `protocol` field); that affects the GetParticipant response. We modified the StageEndpoints object (added the `rtmp` and `rtmps` fields); that affects the CreateStage, GetStage, and UpdateStage responses. We also updated this object's description (added a caching recommendation).	September 9, 2024
Generate Participant Tokens with a Key Pair	We added three objects (PublicKey, PublicKeySummary, StageEndpoints) and four endpoints: (DeletePublicKey, GetPublicKey, ImportPublicKey, ListPublicKeys). We modified the Stage object (added the `endpoints` field); that affects the CreateStage, GetStage, and UpdateStage responses.	June 26, 2024
Individual participant recording	We added one object (AutoParticipantRecordingConfiguration) and modified three objects (Participant, ParticipantSummary, Stage). This affects five endpoints: CreateStage request and response, GetParticipant response, GetStage response, ListParticipants request and response, and UpdateStage request and response.	June 20, 2024

API Change	Description	Date
Remove `svs` from ARN patterns	ARN patterns which specified `[is]vs` were updated to specify `ivs`. This affects all three Tag endpoints and the `ChannelDestinationConfiguration$channelArn` field.	April 25, 2024
Server-side composition updates	We added one object: PipConfiguration. We modified two objects (LayoutConfiguration, GridConfiguration). This affects the GetComposition response and the StartComposition request and response.	March 13, 2024
Composite recording	We added 4 StorageConfiguration endpoints and 7 objects (DestinationDetail, RecordingConfiguration, S3DestinationConfiguration, S3Detail, S3StorageConfiguration, StorageConfiguration, StorageConfigurationSummary). We modified 3 objects (Composition, Destination, DestinationConfiguration). This affects the GetComposition response and the StartComposition request and response.	November 16, 2023
Server-side composition	We added 8 Composition and EncoderConfiguration endpoints and 11 objects (ChannelDestinationConfiguration, Composition, CompositionSummary, Destination, DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration, LayoutConfiguration, and Video).	November 16, 2023
Stage Health: New Participant Data	Added six fields to the Participant object: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion`, and `sdkVersion`. This affects the GetParticipant response.	October 12, 2023

API Change	Description	Date
Participant Token	Added an **Important** note about not building functionality based on current token format.	September 1, 2023
IVS Real-Time Streaming launch	Major documentation changes accompany this release. We renamed the previous documentation to be IVS Low-Latency Streaming and published new IVS Real-Time Streaming documentation. The IVS documentation landing page now has separate sections for real-time streaming and low-latency streaming. Each section has its own User Guide and API Reference. IVS Real-Time Streaming API Reference is part of IVS real-time streaming documentation. Previously it was titled IVS Stage API Reference. Its prior history is described in Document History (Low-Latency Streaming).	August 7, 2023

IVS Release Notes | Real-Time Streaming

This document contains all Amazon IVS Real-Time Streaming release notes, latest first, organized by date of release.

August 27, 2026

Amazon IVS Broadcast SDK: Android 1.46.0, iOS 1.46.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.46.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.46.0/android/ • Fixed a rare crash that could occur when calling `leave()` on a stage while it is handling a fatal error. • Introduced `RealTimeConnection` APIs that create a persistent connection across multiple stages, reducing time to video when users transition between stages. • Fixed a rare crash that could occur if OpenSL ES audio playback failed to start correctly. • Fixed rare playback and recording issues on some device models when using STUDIO or SUBSCRIBE_ONLY audio use case presets.
iOS Broadcast SDK 1.46.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.46.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.46.0/ios/

Platform	Downloads and Changes
	• Fixed a rare crash that could occur when calling `leave()` on a stage while it is handling a fatal error. • Introduced `IVSRealTimeConnection` APIs that create a persistent connection across multiple stages, reducing time to video when users transition between stages.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.089 MB	14.784 MB
armeabi-v7a	5.277 MB	10.235 MB
x86_64	6.206 MB	15.378 MB
x86	6.481 MB	16.010 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	4.085 MB	8.297 MB

August 27, 2026

IVS Broadcast SDK: Web 1.39.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.39.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed an issue where Firefox users could not reconnect to a stage after being offline for an extended period. • Fixed a RangeError that causes publishing to fail on Chrome 152+ when simulcast was enabled.

August 19, 2026

Connection Reuse Across Stages

You can now reuse a single connection across multiple stages, reducing time to video when users transition between stages. Use the new `RealTimeConnection` feature to establish the network connection once and share it across stage joins, eliminating repeated connection setup. This is useful for applications where users browse a feed of live streams and quickly move from one to the next. See Reducing Time to Video When Switching Between Stages.

August 19, 2026

Amazon IVS Broadcast SDK: Android 1.43.2 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.43.2	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.43.2/android/

Platform	Downloads and Changes
	• Fixed an issue that could cause audio to fail to play on some Pixel devices.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.041 MB	14.669 MB
armeabi-v7a	5.234 MB	10.164 MB
x86_64	6.151 MB	15.250 MB
x86	6.427 MB	15.875 MB

August 12, 2026

IVS Broadcast SDK: Web 1.38.1 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.38.1	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed `PUBLISH_ERROR` (code 1015) caused by a Simulcast regression in Chrome 152+ for publishers with simulcast enabled and two or fewer layers configured.

August 11, 2026

Amazon IVS Broadcast SDK: Android 1.43.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.43.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.43.1/android/ • Fixed a rare crash that could occur if OpenSL ES audio playback failed to start correctly. • Fixed an issue that could cause audio to play at reduced volume on some Samsung devices.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.041 MB	14.669 MB
armeabi-v7a	5.234 MB	10.164 MB
x86_64	6.151 MB	15.250 MB
x86	6.427 MB	15.875 MB

July 30, 2026

Amazon IVS Broadcast SDK: Android 1.45.0, iOS 1.45.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.45.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.45.0/android/ • Fixed an issue where subscribers could see a frozen video stream after recovering from network degradation. • Fixed a rare crash related to `CustomImageSource` usage in stages. • Bug fixes and stability improvements.
iOS Broadcast SDK 1.45.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.45.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.45.0/ios/ • Fixed an issue where subscribers could see a frozen video stream after recovering from network degradation. • Bug fixes and stability improvements. • Support for iOS 14 will be deprecated as of IVS Broadcast SDK 1.48.0.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.071 MB	14.716 MB
armeabi-v7a	5.260 MB	10.188 MB
x86_64	6.186 MB	15.307 MB
x86	6.459 MB	15.940 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	4.077 MB	8.281 MB

July 30, 2026

IVS Broadcast SDK: Web 1.38.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.38.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed edge case where audio would be dropped in stage composition due to temporary publisher network failure.

July 7, 2026

Amazon IVS Broadcast SDK: iOS 1.44.1 (Real-Time Streaming)

Platform	Downloads and Changes
iOS Broadcast SDK 1.44.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.44.1/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.44.1/ios/ • Fixed a rare deadlock when leaving and deallocating an `IVSStage`.

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	4.073 MB	8.281 MB

July 2, 2026

Amazon IVS Broadcast SDK: Android 1.44.0, iOS 1.44.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.44.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.44.0/android/ • Improved network utilization during publish and subscribe operations.

Platform	Downloads and Changes
	• Improved recovery performance for subscribers after network degradation.
iOS Broadcast SDK 1.44.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.44.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.44.0/ios/ • Improved network utilization during publish and subscribe operations. • Improved recovery performance for subscribers after network degradation.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.061	14.699
armeabi-v7a	5.250	10.175
x86_64	6.176	15.289
x86	6.452	15.922

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	4.072	8.264

July 2, 2026

IVS Broadcast SDK: Web 1.37.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.37.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and stability improvements.

June 4, 2026

Amazon IVS Broadcast SDK: Android 1.43.0, iOS 1.43.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.43.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.43.0/android/ • The callback provided to the Stage.set AudioCallback API added in 1.42.0 will now be persisted across leave/join cycles, where previously it was cleared on leave. • Fixed a rare crash when tearing down the last Stage that was publishing with the device's microphone.
iOS Broadcast SDK 1.43.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.43.0/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.43.0/ios/ • The callback provided to the IVSStage.setAudioCallback API added in 1.42.0 will now be persisted across leave/join cycles, where previously it was cleared on leave. • Fixed a bug where the camera torch may turn back on unintentionally after changing cameras and rotating the device.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	6.040 MB	14.665 MB
armeabi-v7a	5.232 MB	10.161 MB
x86_64	6.149 MB	15.246 MB
x86	6.425 MB	15.870 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	4.078 MB	8.281 MB

June 4, 2026

IVS Broadcast SDK: Web 1.36.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.36.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and performance improvements.

May 7, 2026

Amazon IVS Broadcast SDK: Android 1.42.0, iOS 1.42.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.42.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.42.0/android/ • Added `Stage.setAudioCallback` for receiving mixed PCM audio data from all remote participants. Must be called before `join()`; automatically cleared on `leave()`. • Bug fixes and performance improvements.
iOS Broadcast SDK 1.42.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.42.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.42.0/ios/

Platform	Downloads and Changes
	• Added `IVSSStage.setAudioCallback` for receiving mixed PCM audio data from all remote participants. Must be called before `join`; automatically cleared on `leave`. • Bug fixes and performance improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.853 MB	14.194 MB
armeabi-v7a	5.078 MB	9.840 MB
x86_64	5.967 MB	14.758 MB
x86	6.220 MB	15.318 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.888 MB	7.935 MB

May 7, 2026

IVS Broadcast SDK: Web 1.35.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.35.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference

Platform	Downloads and Changes
	• Bug fixes and performance improvements.

April 16, 2026

Web Broadcast SDK Token Exchange

Server-side composition layouts now update dynamically after token exchange. If your layout uses attributes such as `featuredParticipantAttribute` or `participantOrderAttribute`, changes made as part of a token exchange will immediately update the active composition without requiring the participant to reconnect.

In addition, token exchange is now supported in the web broadcast SDK, implemented through the `exchangeToken` method added in Web Broadcast SDK 1.33.0.

April 9, 2026

IVS Broadcast SDK: Web 1.34.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.34.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed a bug where subscribe or publish attempts immediately following `exchangeToken` invocations could fail. • Added additional information in the details property for `TOKEN_EXCHANGE_FAILED` errors.

April 9, 2026

Amazon IVS Broadcast SDK: Android 1.41.0, iOS 1.41.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.41.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.41.0/android/ • Added ability to choose between platform-native echo cancellation, AECM, or AEC3 algorithms with automatic fallback support. • Added ability to choose between platform-native or software-based noise suppression with automatic fallback support. • **Important:** If you use `StageAudioConfiguration.enableNoiseSuppression`, you must now call `StageAudioManager.enableNoiseSuppression` instead. Noise suppression is now managed globally rather than per-stream. • Software noise suppression is now disabled by default for `STUDIO` and `SUBSCRIBE_ONLY` audio mode presets to align with iOS.
iOS Broadcast SDK 1.41.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.41.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.41.0/ios/

Platform	Downloads and Changes
	• Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.834 MB	14.157 MB
armeabi-v7a	5.056 MB	9.812 MB
x86_64	5.945 MB	14.721 MB
x86	6.200 MB	15.285 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.927 MB	7.980 MB

April 8, 2026

Redundant Ingest 24/7 Streaming

Redundant ingest is now available for RTMP(S) and E-RTMP(S) streams. This feature enables streaming from two encoders simultaneously to a single stage, with automated failover for the same source media. Redundant ingest is ideal for live events, 24/7 live streams, or any scenario where uninterrupted delivery is essential. By streaming from two encoders, you can protect against unexpected disruptions while also enabling continuous 24/7 streaming. For more information, see:

- User Guide — We added Redundant Ingest in *IVS RTMP Publishing*.
- Real-Time Streaming API Reference — Changes are described in the “API Reference” table of the Document History.

March 12, 2026

IVS Broadcast SDK: Web 1.33.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.33.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Implemented the `exchangeToken` method for real-time token exchange. • Implemented the STAGE_PARTICIPANT_METADATA_CHANGED event, which fires when `attributes` and/or `userId` change(s) after token exchanges. • Added the `encoderImplementation` field on the request local stage stream `requestQualityStats()` method.

March 12, 2026

Amazon IVS Broadcast SDK: Android 1.40.0, iOS 1.40.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.40.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/android/ • Improved error messages around TLS certificate validation failures and expanded error enum codes.

Platform	Downloads and Changes
iOS Broadcast SDK 1.40.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.40.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/ios/ • Improved error messages around TLS certificate validation failures and expanded error enum codes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.823 MB	14.139 MB
armeabi-v7a	5.046 MB	9.798 MB
x86_64	5.935 MB	14.702 MB
x86	6.190 MB	15.265 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.939 MB	8.013 MB

February 13, 2026

Amazon IVS Broadcast SDK: Android 1.39.0, iOS 1.39.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.39.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.39.0/android/ • Minor bug fixes for Bluetooth headset reconnects using the STUDIO audio mode. • Updated core Android build tools and NDK version. • Fixed rare deadlock when stopping a `MixedImageDevice` .
iOS Broadcast SDK 1.39.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.39.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.39.0/ios/ • Updated Xcode to version 26.2. • Effective with this release, the IVS SDKs are no longer distributed via CocoaPods. CocoaPods announced its deprecation in 2024 and will enter read-only state later this year. Swift Package Manager (SPM) replaces CocoaPods as Apple's supported dependency-management solution and is the standard way to integrate SDKs in modern Xcode projects.

Platform	Downloads and Changes
	We recommend that you migrate to SPM or integrate the IVS SDK frameworks directly into your project. IVS SDKs are fully supported via both approaches. Related documentation changes were made in: • Getting Started with IVS Real-Time Streaming – "Step 4: Integrate the IVS Broadcast SDK" > "iOS" • iOS Broadcast SDK Guide – "Install the Library"

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.788 MB	14.059 MB
armeabi-v7a	5.016 MB	9.740 MB
x86_64	5.898 MB	14.615 MB
x86	6.154 MB	15.184 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.932 MB	7.996 MB

February 12, 2026

IVS Broadcast SDK: Web 1.32.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.32.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and stability improvements.

January 13, 2026

Amazon IVS Broadcast SDK: Android 1.38.0, iOS 1.38.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.38.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.38.0/android/ • Layer prioritization function — Added `Priority` enum to `StageVideoConfiguration.Layer` with values: `VERY_LOW`, `LOW`, `MEDIUM`, `HIGH`. This will determine which layer is dropped first under network bandwidth constraints. • Faster stage reconnection after network connectivity is restored. • Changed the codes associated with some errors. See Mobile Broadcast SDK Error Migration Guide below.

Platform	Downloads and Changes
iOS Broadcast SDK 1.38.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.38.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.38.0/ios/ • Layer prioritization function — Added `IVSLocalStageStreamLayerPriority` enum with values: `VeryLow`, `Low`, `Medium`, `High`. This will determine which layer is dropped first under network bandwidth constraints. • Faster stage reconnection after network connectivity is restored. • Changed the codes associated with some errors. See Mobile Broadcast SDK Error Migration Guide below.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.795 MB	14.070 MB
armeabi-v7a	5.021 MB	9.746 MB
x86_64	5.904 MB	14.630 MB
x86	6.161 MB	15.198 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.609 MB	8.078 MB

Mobile Broadcast SDK Error Migration Guide

In version 1.38.0 of the iOS and Android broadcast SDKs, the codes associated with some errors have changed. Previously, there was no single property that could be used to uniquely identify any error emitted from the SDKs. Instead, to understand what an error meant, a combination of the following properties needed to be examined:

Android	iOS
`BroadcastException.getCode()`	`NSError.code`
`BroadcastException.getUid()`	`NSError.userInfo[IVSBroadcastUidDescriptionErrorKey]`
`BroadcastException.getError()`	`NSError.userInfo[IVSBroadcastResultDescriptionErrorKey]`
`BroadcastException.getSource()`	`NSError.userInfo[IVSBroadcastSourceDescriptionErrorKey]`
`BroadcastException.getDetail()`	`NSError.userInfo[NSLocalizedDescriptionKey]`

With version 1.38.0 and greater, `BroadcastException.getCode()` (Android) and `NSError.code` (iOS) return a unique ID that can be looked up in the public `BroadcastErrorCode` (Android) and `IVSBroadcastErrorCode` (iOS) enums.

In addition to making `code` the unique ID for all errors, an additional field was added: `BroadcastException.getPlatformCode()` (Android) and `NSError.userInfo[IVSBroadcastPlatformCodeDescriptionErrorKey]` (iOS). If an error is caused by the underlying platform (such as a network error or a video encode or decode

error), this field is non-zero and can be used to collect additional information from the platform's documentation.

Migrating from SDK 1.37.0 and Earlier

To make every error conform to the new strategy, some existing errors had to change their values. Below is a guide to map existing logic to the new logic:

- Any error where `code` was non-zero will keep the same value for code; however, referencing the code through the new enum constants may improve clarity. For example, comparing an error to `BroadcastErrorCode.Broadcast.LatencyThresholdReached` is clearer than comparing it to `20401`.
- Any error where `UID` had a value (i.e. was not `-1` on Android or `"-1"` on iOS) will now have the `code` field set to what the existing `UID` value was. If you have conditionals comparing the `UID` field, you can keep the constants but compare them against the `code` field going forward.
- Some legacy errors did not contain a `code` or a `UID` value. These were commonly matched based on the `message` (Android) or `description` (iOS) of the error, which is not a reliable way to identify errors because of the dynamic nature of error messages. Because these errors didn't have uniquely identifying characteristics, one-to-one mappings can't be provided. However, most errors kept the same description, so it is possible to continue using the same matching logic while also gathering and reporting the new `code` value for future app releases.

As a concrete example, the error checking in the following table should be migrated as follows:

Before	After
`error.code == 20401`	`error.code == BroadcastErrorCode .Broadcast.LatencyThreshold Reached` No change, but prefer comparison to the enum value.
`error.uid == 207`	`error.code == BroadcastErrorCode .Net.SocketRemoteHangup` Compare to `code` instead of `uid`.

Before	After
`error.message.contains("Ice ConnectionFailed")`	`error.code == BroadcastErrorCode .RealTime.PeerConnectionIce ConnectionFailed` Don't compare to `message` (or `source`, or `result/detail`). Instead, find the appropriate enum code to compare to.

The most important part of an error is still `BroadcastException.getPlatformCode()` (Android) and `NSError.userInfo[IVSBroadcastPlatformCodeDescriptionErrorKey]` (iOS), but in version 1.38.0 and beyond, the `code` field uniquely identifies errors and allows immediate lookup of the error name and description in the `BroadcastErrorCode` (Android) and `IVSBroadcastErrorCode` (iOS) enums. As a result, other fields like `UID`, `source`, and `detail` should not be used in lookup logic; they exist only as supplemental information.

December 11, 2025

Amazon IVS Broadcast SDK: Android 1.37.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.37.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.37.1/android/ • Fixed issues related to participant preview teardown.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.754 MB	13.965 MB

Architecture	Compressed Size	Uncompressed Size
armeabi-v7a	4.991 MB	9.683 MB
x86_64	5.858 MB	14.529 MB
x86	6.128 MB	15.120 MB

December 9, 2025

Participant Token Exchange

New support for participant token exchange enables you to upgrade or downgrade token capabilities and update token attributes within the IVS client SDK without forcing clients to disconnect and reconnect. This is useful for scenarios like co-hosting, where participants may start with subscribe-only capabilities and later need publish capabilities.

See the new page on Token Exchange.

December 5, 2025

IVS Broadcast SDK: Web 1.31.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.31.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and stability improvements.

December 5, 2025

Amazon IVS Broadcast SDK: Android 1.37.0, iOS 1.37.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.37.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.37.0/android/ • Bug fixes and stability improvements. • Added support for participant token exchange.
iOS Broadcast SDK 1.37.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.37.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.37.0/ios/ • Bug fixes and stability improvements. • Added support for participant token exchange.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.753 MB	13.961 MB
armeabi-v7a	4.990 MB	9.680 MB
x86_64	5.857 MB	14.525 MB
x86	6.127 MB	15.116 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.588 MB	8.028 MB

November 7, 2025

Individual Participant Recording Synchronization

New support for `EXT-X-PROGRAM-DATE-TIME` tags in individual participant recording HLS playlists enables precise synchronization of multiple participant recordings during post-processing. This feature provides millisecond-accurate UTC timestamps at recording start and discontinuity points, allowing you to create synchronized compositions (such as side-by-side or picture-in-picture layouts) even when participants experience network interruptions or join at different times. For details, see Synchronize Multiple Participant Recordings in *Individual Participant Recording*.

October 30, 2025

IVS Broadcast SDK: Web 1.30.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.30.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and stability improvements.

October 30, 2025

Amazon IVS Broadcast SDK: Android 1.36.0, iOS 1.36.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.36.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.36.0/android/ • Improved camera recovery when returning to the foreground after being in the background for a prolonged period of time. • Added an `embedMessage` method on `ImageDevice` to enable the insertion of metadata payloads into a publishing video stream. See Embed Messages in the *Android Broadcast SDK Guide*.
iOS Broadcast SDK 1.36.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.36.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.36.0/ios/ • Added an `embedMessage` method on `IVSImageDevice` to enable the insertion of metadata payloads into a publishing video stream. See Embed Messages in the *iOS Broadcast SDK Guide*.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.736 MB	13.898 MB
armeabi-v7a	4.974 MB	9.638 MB
x86_64	5.839 MB	14.456 MB
x86	6.109 MB	15.047 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.569 MB	7.962 MB

October 14, 2025

Updated Real-Time Limit: Compositions

We updated the quota for "maximum concurrent Composition resources per account" from 5 to 20. It is documented in Service Quotas > Other Quotas.

October 2, 2025

IVS Broadcast SDK: Web 1.29.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.29.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Bug fixes and stability improvements.

October 2, 2025

Amazon IVS Broadcast SDK: Android 1.35.0, iOS 1.35.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.35.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.35.0/android/ • Bug fixes and stability improvements.
iOS Broadcast SDK 1.35.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.35.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.35.0/ios/ • `IVSImageDevice.setOnFrameCallback` can now be customized with a `DispatchQueue`, and it can optionally include the `CVPixelBuffer` associated with the frame.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.730 MB	13.900 MB
armeabi-v7a	4.971 MB	9.639 MB
x86_64	5.835 MB	14.455 MB
x86	6.104 MB	15.041 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.569 MB	7.963 MB

September 16, 2025

Server-Side Composition Custom Participant Ordering

New support for custom participant ordering for SSC provides granular control over participant positioning in both grid and Picture-in-Picture (PiP) layouts. See Server-Side Composition (various changes, including adding `participantOrderAttribute` and "Custom Participant Ordering") and the IVS Real-Time Streaming API Reference (added `participantOrderAttribute` to the Composition object).

September 11, 2025

Amazon IVS Broadcast SDK: Android 1.34.0, iOS 1.34.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.34.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.34.0/android/ • CPU improvements for publish and subscribe media transport. • Added `packetsLost` to `LocalVideoStats` and `LocalAudioStats`.
iOS Broadcast SDK 1.34.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.34.0/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.34.0/ios/ • CPU improvements for publish and subscribe media transport. • Added `packetsLost` to `IVSLocalVideoStats` and `IVSLocalAudioStats`. • Fixed a bug where devices did not detach after leaving a stage, which could result in privacy indicators unexpectedly being lit.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.796 MB	14.089 MB
armeabi-v7a	5.036 MB	9.788 MB
x86_64	5.906 MB	14.653 MB
x86	6.174 MB	15.240 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.594 MB	8.046 MB

September 10, 2025

Interface VPC Endpoints

New support for interface VPC (Virtual Private Cloud) endpoints enables you to establish a secure private connection between your Amazon VPC and IVS, for workloads that require secure, live video ingestion. This keeps your IVS ingest traffic within the AWS network and off the public internet. Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services, using an elastic network interface with private IPs in your Amazon VPC. See Private Ingest in the *IVS Low-Latency Streaming User Guide* and Private Ingest to Stages in the *IVS Real-Time Streaming User Guide*.

September 4, 2025

IVS Broadcast SDK: Web 1.28.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.28.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Joining a stage that was deleted, or with a participant token that was disconnected, now reports `STAGE_DELETED` or `STAGE_DISCONNECTED` errors instead of `TIMEOUT`. • Optimized internal polling requests related to simulcast.

August 7, 2025

IVS Broadcast SDK: Web 1.27.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.27.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added `requestQualityStats` to `RemoteStageStream`, which then exposes a simplified object of video and audio stats sourced from `requestRTCStats`. • Updates to ensure that the `RemoteStageStream` muted state and its `mediaStreamTrack` enabled state are always in sync.

August 7, 2025

Amazon IVS Broadcast SDK: Android 1.33.0, iOS 1.33.0 (Real-Time Streaming)

Platform	Downloads and Changes

Android Broadcast SDK 1.33.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.33.0/android/
	• New methods to control device torch:
	  • `CameraSource.Capabilities` implements `isTorchSupported`.
	  • `CameraSource.Options.Builder` implements `setEnableTorch`.

Platform	Downloads and Changes
	• The Android broadcast SDK meets Google Play's 16 KB page-size compatibility requirement. (Note: This was implemented as of version 1.23.0 of the SDK.)
iOS Broadcast SDK 1.33.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.33.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.33.0/ios/ • New method to control device torch: `IVSImageDevice` implements two properties, `isTorchSupported` and `torchEnabled`. Check if the device supports torch with `isTorchSupported`, and then toggle it by setting `torchEnabled`. • Resolved an issue on iOS 18.5+ with certain VPNs that could result in peer connection timeouts. (Note: This was implemented as of version 1.32.1 of the SDK.)

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.689 MB	13.829 MB
armeabi-v7a	4.962 MB	9.649 MB
x86_64	5.806 MB	14.413 MB
x86	6.066 MB	14.983 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.505 MB	7.828 MB

July 25, 2025

Amazon IVS Broadcast SDK: Android 1.32.2 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.32.2	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.32.2/android/ • Disabled IPv6 for `Stage` connections.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.693 MB	13.838 MB
armeabi-v7a	4.964 MB	9.653 MB
x86_64	5.810 MB	14.422 MB
x86	6.067 MB	14.988 MB

July 23, 2025

Enforcement of New Real-Time Metrics and Limits: Concurrent Publishers and Subscriptions

On June 23, we introduced two new adjustable service quotas, for the maximum number of concurrent publishers and concurrent subscriptions across all stages in an AWS Region. Today we start enforcing these new quotas.

July 15, 2025

New Real-Time Limit: Concurrent Participant Replications

We've introduced a new non-adjustable service quota, for the maximum number of concurrent replications per participant across all stages in an AWS Region. It is documented in Service Quotas > Other Quotas.

July 10, 2025

Amazon IVS Broadcast SDK: Android 1.32.1, iOS 1.32.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.32.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.32.1/android/ • Removed `StageAudioConfiguration.enableEchoCancellation()`. Instead, use `StageAudioManager` to enable or disable echo cancellation. • Modified the `STUDIO` and `SUBSCRIBE_ONLY` presets in `StageAudioManager` to turn off echo cancellation. If you want to

Platform	Downloads and Changes
	use `STUDIO` with echo cancellation, first set the preset, then enable echo cancellation to override `STUDIO`'s default preference for no echo cancellation. • Added a `MixedDevice` API suite for compositing multiple image and audio sources into a single output `Device`, which can be used for publishing more complex audio and visuals to a stage.
iOS Broadcast SDK 1.32.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.32.1/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.32.1/ios/ • Added an `IVSMixedDevice` API suite for compositing multiple image and audio sources into a single output `IVSDevice`, which can be used for publishing more complex audio and visuals to a stage.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.692 MB	13.840 MB
armeabi-v7a	4.965 MB	9.655 MB
x86_64	5.810 MB	14.424 MB
x86	6.068 MB	14.990 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.508 MB	7.900 MB

July 7, 2025

IVS Broadcast SDK: Web 1.26.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.26.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added `requestQualityStats` to LocalStageStream, which exposes a simplified object of video and audio stats sourced from requestRTCStats. • Fixed websocket leaks that could occur during setup, causing subsequent join failures. • Fixed an issue where 1302 errors would incorrectly surface when retrying a failed subscribe or publish operation. • Improved retry stability for subscribe and publish connections when the join connection is in an ERRORED or CONNECTING state.

June 23, 2025

New Real-Time Metrics and Limits: Concurrent Publishers and Subscriptions

We've introduced two new adjustable service quotas, for the maximum number of concurrent publishers and concurrent subscriptions across all stages in an AWS Region. They are documented in Service Quotas > Other Quotas. These quotas give you more control over total usage across your account. Previously, IVS enforced limits only on the number of publishers and subscribers *per stage*. This made it hard to set safeguards at the account level and could result in higher usage and associated costs than expected, especially for customers creating many stages.

Note: We will start enforcing these new quotas on July 23, to allow 30 days for you to review your usage and request service-quota increases if needed.

We also added two new CloudWatch metrics, `ConcurrentPublishers` and `ConcurrentSubscriptions`. These metrics help you monitor usage across all stages and assess whether you are approaching the default limits. They are documented in Monitoring Real-Time Streaming > CloudWatch Metrics. We recommend setting up CloudWatch alarms to alert you when your usage is close to a quota limit.

June 20, 2025

E-RTMP Multitrack Video Ingest Support

You can use E-RTMP (Enhanced Real-Time Messaging Protocol) multitrack video to send multiple video qualities to your IVS stages. This feature enables adaptive bitrate streaming, allowing viewers to watch in the best quality for their network connection. See E-RTMP Multitrack Video in the IVS RTMP Publishing documentation.

June 16, 2025

IVS Broadcast SDK: Web 1.25.1 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.25.1	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Removed the NPM unintentional engine enforcement of v22. All LTS node versions are supported as the package is transpiled.

June 12, 2025

Amazon IVS Broadcast SDK: Android 1.31.0, iOS 1.31.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.31.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.31.0/android/ • Bug fixes and stability improvements.
iOS Broadcast SDK 1.31.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.31.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.31.0/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.579 MB	13.594 MB
armeabi-v7a	4.864 MB	9.473 MB
x86_64	5.697 MB	14.173 MB
x86	5.951 MB	14.724 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.431 MB	7.732 MB

June 12, 2025

IVS Broadcast SDK: Web 1.25.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.25.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed a bug where SEI messages might fail to send after a remote participant encountered an ERROR state. • Fixed a bug where multiple remote stage streams might be returned when the `STAGE_STREAM_MUTE_CHANGED` stage event was invoked.

Platform	Downloads and Changes
	• Fixed a bug where `STAGE_PARTICIPANT_STREAMS_REMOVED` was not invoked for streams that had errored.

May 29, 2025

Participant Replication

Participant replication allows you to copy a participant from one stage to another. This is useful when you want the same participant to appear in multiple stages at the same time, enabling cross-stage interactions. For documentation changes, see the Document History (both User Guide and API Reference tables).

May 26, 2025

Amazon IVS Broadcast SDK: Android 1.30.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.30.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.30.1/android/ • Fixed a low-microphone-volume bug on some Android devices when using SDK-managed microphones from `DeviceDiscovery` with the `STUDIO` audio preset.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.579 MB	13.592 MB

Architecture	Compressed Size	Uncompressed Size
armeabi-v7a	4.863 MB	9.472 MB
x86_64	5.696 MB	14.171 MB
x86	5.950 MB	14.722 MB

May 15, 2025

IVS Broadcast SDK: Web 1.24.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.24.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed memory leaks when leaving and rejoining a stage.

May 15, 2025

Amazon IVS Broadcast SDK: Android 1.30.0, iOS 1.30.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.30.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.30.0/android/ • Bug fixes and stability improvements.

Platform	Downloads and Changes
iOS Broadcast SDK 1.30.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.30.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.30.0/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.571 MB	13.577 MB
armeabi-v7a	4.857 MB	9.462 MB
x86_64	5.691 MB	14.156 MB
x86	5.944 MB	14.708 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.430 MB	7.732 MB

May 2, 2025

IVS Broadcast SDK: Web 1.23.1 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.23.1	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed an issue where participant join events always occurred before `join()` resolved. • Fixed an issue where local participants were erroneously reported as remote participants when leaving and rejoining in quick succession.

April 17, 2025

Amazon IVS Broadcast SDK: Android 1.29.0, iOS 1.29.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.29.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.29.0/android/ • Added a simulcast publisher controls feature. See "Configuring Layered Encoding (Publisher)" in the Android Broadcast SDK Guide. • Bug fixes and stability improvements.

Platform	Downloads and Changes
iOS Broadcast SDK 1.29.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.29.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.29.0/ios/ • Added a simulcast publisher controls feature. See "Configuring Layered Encoding (Publisher)" in the iOS Broadcast SDK Guide. • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.566 MB	13.546 MB
armeabi-v7a	4.853 MB	9.444 MB
x86_64	5.681 MB	14.119 MB
x86	5.939 MB	14.674 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.429 MB	7.715 MB

April 17, 2025

IVS Broadcast SDK: Web 1.23.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.23.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added a simulcast publisher controls feature. See "Configuring Layered Encoding (Publisher)" in the Web Broadcast SDK Guide. • Improved time to publish latency. This impacts the timing of the `PUBLISHED` event. • Fixed a bug where the SDK fired join category errors via the ERROR callback when connection to the stage was lost but potentially recoverable (specifically, `FAILED` and `TIMEOUT` errors for the `JOIN_ERROR` category). • Fixed a bug with the `insertSeiMessage` operation where a strategy refresh could result in subsequent invocations of `insertSeiMessage` failing to send the SEI message.

April 2, 2025

New Quota: Compositions Per Stage

We added a new quota, for the maximum concurrent compositions allowed per stage. This is documented in Service Quotas > Other Quotas.

March 20, 2025

Amazon IVS Broadcast SDK: Android 1.28.1, iOS 1.28.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.28.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.28.1/android/ • Bug fixes and stability improvements.
iOS Broadcast SDK 1.28.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.28.1/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.28.1/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.613 MB	13.760 MB
armeabi-v7a	4.885 MB	9.558 MB
x86_64	5.728 MB	14.342 MB
x86	5.987 MB	14.923 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.417 MB	7.698 MB

March 20, 2025

IVS Broadcast SDK: Web 1.22.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.22.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added `null` as a valid return type to the preferredLayerForStream strategy method. • Fixed a bug where `preferredLayerForStream` was not called again if new layers became available after the stream started. • Fixed a bug where `stream.getHighestQualityLayer` did not pick the highest quality layer after the stream started.

March 19, 2025

Amazon IVS Broadcast SDK: Android 1.27.2, iOS 1.27.2 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.27.2	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.27.2/android/ • Fixed a resource-leak regression that impacted some devices when creating 50 or more stages. • Fixed a regression that could cause an increased rate of video freezes when using third-party publishing software.
iOS Broadcast SDK 1.27.2	**Download for real-time streaming:** https://broadcast.live-video.net/1.27.2/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.27.2/ios/ • Fixed a regression that could cause an increased rate of video freezes when using third-party publishing software.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.700 MB	14.197 MB
armeabi-v7a	4.945 MB	9.879 MB

Architecture	Compressed Size	Uncompressed Size
x86_64	5.810 MB	14.802 MB
x86	6.073 MB	15.412 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.622 MB	8.584 MB

March 13, 2025

Target Segment Duration

This release adds to the IVS real-time streaming API, to allow you to define the target duration for recorded segments generated when either using composite recording or recording a stage participant. For specific API changes, see the Document History (both User Guide and API Reference tables).

March 6, 2025

Individual Participant Recording Stitching

This is the first release of new functionality. If your stage is configured for individual participant recording, you can now specify a window of time during which, if a stage publisher disconnects from a stage and then reconnects, IVS tries to record to the same S3 prefix as the previous session. In other words, if a publisher disconnects and then reconnects within the specified interval, the multiple recordings are considered a single recording and merged. For documentation changes, see the Document History (both the User Guide and API Reference tables).

March 3, 2025

Amazon IVS Broadcast SDK: iOS 1.27.1 (Real-Time Streaming)

Platform	Downloads and Changes
iOS Broadcast SDK 1.27.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.27.1/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.27.1/ios/ • Improved focus performance for objects held close to the camera while using the ultra-wide lens on Pro devices.

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.625 MB	8.601 MB

February 20, 2025

Amazon IVS Broadcast SDK: Android 1.27.0, iOS 1.27.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.27.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.27.0/android/ • Bug fixes and stability improvements.

Platform	Downloads and Changes
iOS Broadcast SDK 1.27.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.27.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.27.0/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.700 MB	14.197 MB
armeabi-v7a	4.944 MB	9.879 MB
x86_64	5.809 MB	14.802 MB
x86	6.073 MB	15.412 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.625 MB	8.601 MB

February 20, 2025

IVS Broadcast SDK: Web 1.21.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.21.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Updated `preferredLayerForStream` strategy types to include `null`, which is a valid return. • Fixed TypeScript compile errors when TSconfig `skipLibCheck` is set to false. Note: As part of this release, types have been consolidated into a single rollup. If an application imports nested types based on path, errors may occur. If errors do occur, change the import to simply `'amazon-ivs-broadcast'`.

January 30, 2025

Amazon IVS Broadcast SDK: Android 1.26.0, iOS 1.26.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.26.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.26.0/android/ • Bug fixes and stability improvements.

Platform	Downloads and Changes
iOS Broadcast SDK 1.26.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.26.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.26.0/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.695 MB	14.186 MB
armeabi-v7a	4.939 MB	9.872 MB
x86_64	5.804 MB	14.790 MB
x86	6.065 MB	15.398 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.624 MB	8.601 MB

January 23, 2025

IVS Broadcast SDK: Web 1.20.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.20.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added the `insertSeiMessage` method on LocalStageStream to enable the insertion of Supplemental Enhancement Information (SEI) payloads into a publishing video stream. See Supplemental Enhanced Information in the *IVS Broadcast SDK: Web Guide*.

December 12, 2024

Amazon IVS Broadcast SDK: Android 1.25.0, iOS 1.25.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.25.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.25.0/android/ • Added a simulcast controls feature. See Configuring Layered Encoding with Simulcast (Subscriber) in *Streaming Optimizations*. • Made SEI (Supplemental Enhanced Information) payloads available to subscribe

Platform	Downloads and Changes
	rs with a new field on ImageDeviceFrame objects. See Get Supplemental Enhancement Information (SEI) in the *IVS Broadcast SDK: Android Guide*. • Added the `SubscribeConfiguration::setInitialGain` method to allow the configuration of the initial gain value for incoming audio streams. • Bug fixes and stability improvements.
iOS Broadcast SDK 1.25.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.25.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.25.0/ios/ • Added a simulcast controls feature. See Configuring Layered Encoding with Simulcast (Subscriber) in *Streaming Optimizations*. • Made SEI (Supplemental Enhanced Information) payloads available to subscribers with a new field on IVSImageDeviceFrame objects. See Get Supplemental Enhancement Information (SEI) in the *IVS Broadcast SDK: iOS Guide*. • Added the `IVSSubscribeConfiguration.initialGain` method to allow the configuration of the initial gain value for incoming audio streams. • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.677 MB	14.103 MB
armeabi-v7a	4.905 MB	9.791 MB
x86_64	5.786 MB	14.725 MB
x86	6.030 MB	15.302 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.625 MB	8.585 MB

December 12, 2024

IVS Broadcast SDK: Web 1.19.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.19.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added a simulcast controls feature. See Configuring Layered Encoding with Simulcast (Subscriber) in *Streaming Optimizations*. • Bug fixes and stability improvements.

December 10, 2024

Real-Time Streaming Thumbnail Configuration

This release allows you to enable/disable the recording of thumbnails for a live session and modify the interval at which thumbnails are generated for the live session. This is the first release of this new functionality. See:

- Individual Participant Recording — We updated examples and JSON metadata information, and we added pricing information and "Thumbnail-Only Recordings."
- Composite Recording — We updated examples and JSON metadata information, and we added pricing information.
- API Reference RT — We made several changes:
  - Modified the S3DestinationConfiguration object: added `thumbnailConfigurations`. This affects the GetComposition response and StartComposition request and response.
  - Modified the AutoParticipantRecordingConfiguration object: added `thumbnailConfiguration` and added `NONE` as a valid value for `mediaTypes`. This affects the CreateStage request and response, GetStage response, and UpdateStage request and response.
  - Added two objects: CompositionThumbnailConfiguration and ParticipantThumbnailConfiguration.

November 13, 2024

Amazon IVS Broadcast SDK: Android 1.24.0, iOS 1.24.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.24.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.24.0/android/ • Bug fixes and stability improvements.

Platform	Downloads and Changes
iOS Broadcast SDK 1.24.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.24.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.24.0/ios/ • Bug fixes and stability improvements.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.521 MB	13.791 MB
armeabi-v7a	4.789 MB	9.623 MB
x86_64	5.718 MB	14.709 MB
x86	5.933 MB	15.163 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.589 MB	8.466 MB

November 12, 2024

IVS Broadcast SDK: Web 1.18.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.18.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added a new event to make SEI (Supplemental Enhanced Information) payloads available to subscribers. • Fixed an exception that would occur during unpublish and unsubscribe requests. • Fixed a race condition where joining and leaving rapidly would cause an error for other participants.

October 10, 2024

IVS Broadcast SDK: Web 1.17.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.17.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Minor bug fixes.

October 10, 2024

Amazon IVS Broadcast SDK: Android 1.23.0, iOS 1.23.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.23.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/android/ • With this release we also began publishing a version of the Android broadcast SDK which includes debug symbols. See Using the SDK with Debug Symbols. • Minor bug fixes.
iOS Broadcast SDK 1.23.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.23.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/ios/ • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.432 MB	13.560 MB
armeabi-v7a	4.707 MB	9.451 MB
x86_64	5.626 MB	14.459 MB
x86	5.838 MB	14.908 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.542 MB	8.316 MB

September 11, 2024

Amazon IVS Broadcast SDK: Android 1.22.0, iOS 1.22.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.22.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/android/ • Fixed a bug where certain Android devices show a black frame in the preview after switching camera inputs. • Minor bug fixes.
iOS Broadcast SDK 1.22.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.22.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/ios/ • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.359 MB	13.392 MB
armeabi-v7a	4.636 MB	9.325 MB
x86_64	5.548 MB	14.268 MB
x86	5.754 MB	14.710 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.488 MB	8.199 MB

September 11, 2024

IVS Broadcast SDK: Web 1.16.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.16.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Minor bug fixes.

September 9, 2024

RTMP Ingest

As an alternative to using the IVS broadcast SDK, you can now publish video to an IVS stage from an RTMP source (in addition to WHIP, which already was supported). For documentation changes, see the Document History (both the User Guide and API Reference tables).

August 19, 2024

In-Console Publish/Subscribe

You can now publish and subscribe from the IVS console. In *Getting Started with IVS Real-Time Streaming*, see Publish and Subscribe to Video.

August 15, 2024

IVS Broadcast SDK: Web 1.15.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.15.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed a race condition that impacts publisher media quality when `join()` is called repeatedly. Calling `join()` in succession no longer re-triggers the `STAGE_PARTICIPANT_JOINED` event, along with accompanying publish and stream state changes. • Fixed a bug that causes issues parsing participant tokens when non-text characters are used in the token `attributes` field.

Platform	Downloads and Changes
	• Added a method to configure a participant's subscribers. Initially, you can configure only the jitter-buffer minimum delay. See the SDK reference documentation, Configuration for Subscribing to Participants in the *Web Broadcast SDK Guide*, and Changing Subscriber Jitter Buffer MinDelay in *Streaming Optimizations*.

August 15, 2024

Amazon IVS Broadcast SDK: Android 1.21.0, iOS 1.21.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.21.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/android/ • Fixed a bug impacting devices with MT6765 chipsets, where the subscriber preview renders black frames under some circumstances. • Added a method to configure a participant's subscribers. Initially, you can configure only the jitter-buffer minimum delay. See the SDK reference documentation, Configuration for Subscribing to Participants in the *Android Broadcast SDK Guide*, and Changing Subscriber Jitter Buffer MinDelay in *Streaming Optimizations*. • Minor bug fixes.

Platform	Downloads and Changes
iOS Broadcast SDK 1.21.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.21.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/ios/ • Added a method to configure a participant's subscribers. Initially, you can configure only the jitter-buffer minimum delay. See the SDK reference documentation, Configuration for Subscribing to Participants in the *iOS Broadcast SDK Guide*, and Changing Subscriber Jitter Buffer MinDelay in *Streaming Optimizations*. • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.350 MB	13.378 MB
armeabi-v7a	4.628 MB	9.312 MB
x86_64	5.538 MB	14.253 MB
x86	5.744 MB	14.694 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.485 MB	8.199 MB

July 18, 2024

IVS Broadcast SDK: Web 1.14.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.14.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • API documentation improvements. • Fixed video and audio stats outliers reported during connection resets. • Minor dependency updates.

July 18, 2024

Amazon IVS Broadcast SDK: Android 1.20.0, iOS 1.20.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.20.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/android • Fixed a bug that prevented the Broadcast SDK from running on Chromebooks with Intel processors. • Minor bug fixes.
iOS Broadcast SDK 1.20.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.20.0/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/ios • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.318 MB	13.299 MB
armeabi-v7a	4.605 MB	9.254 MB
x86_64	5.507 MB	14.168 MB
x86	5.715 MB	14.608 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.465 MB	8.164 MB

June 26, 2024

Generate Participant Tokens with a Key Pair

You can now generate participant tokens on your own server application by using a key pair. This enables you to avoid calling CreateParticipantToken every time you need a participant token. For documentation changes, see the Document History (both the User Guide and API Reference tables).

June 20, 2024

Individual Participant Recording

Individual participant recording allows IVS real-time streaming customers to record IVS stage publishers individually into S3 buckets. See Recording, Individual Participant Recording, and changes in the Real-Time Streaming API Reference. (For specific documentation changes, see the Document History.)

June 13, 2024

Amazon IVS Broadcast SDK: Android 1.19.0, iOS 1.19.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.19.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.19.0/android • Recent Android versions require an icon in the notification that is displayed when capturing the screen. If desired, you can now customize the icon by calling `setSmallIcon` on the `Notification.Builder` returned by `Session # createServiceNotificationBuilder` . • Improved connection recovery time on devices transitioning from wifi to cellular connections. This change requires the `CHANGE_NETWORK_STATE` permission.
iOS Broadcast SDK 1.19.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.19.0/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.19.0/ios • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.304 MB	13.340 MB
armeabi-v7a	4.598 MB	9.299 MB
x86_64	5.495 MB	14.207 MB
x86	5.694 MB	14.625 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.393 MB	7.949 MB

June 13, 2024

IVS Broadcast SDK: Web 1.13.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.13.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference

Platform	Downloads and Changes
	• Updated the duration of event change behavior for `StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED` and `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`. Participants now remain in the `ATTEMPTING_SUBSCRIBE` or `ATTEMPTING_PUBLISH` state for a longer time, until the `ERRORED` event is fired. • Added the `StageEvents.ERROR` event for listening to errors encountered by the SDK. See Error Handling in the *Real-Time Broadcast SDK: Web Guide* for more information.

May 20, 2024

IVS Broadcast SDK: Web 1.12.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.12.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Improved retry handling for publish and subscribe operations. • Improved analytics, specifically latency and audio-quality measurement.

May 16, 2024

Amazon IVS Broadcast SDK: Android 1.18.0, iOS 1.18.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.18.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/android • The SDK now sends specific error codes when a connected Stage is deleted by the AWS control plane, or when the token in use is revoked. • Minor bug fixes.
iOS Broadcast SDK 1.18.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.18.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/ios • The SDK now sends specific error codes when a connected Stage is deleted by the AWS control plane, or when the token in use is revoked. • Added the IVSCamera `setVideoZoomFactor` method and the associated `IVSCameraDelegate` methods.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.275 MB	13.279 MB
armeabi-v7a	4.573 MB	9.254 MB
x86_64	5.472 MB	14.142 MB
x86	5.664 MB	14.554 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.393 MB	7.916 MB

May 6, 2024

IVS Broadcast SDK: Web 1.11.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.11.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed an edge case where the SDK did not attempt to recover on a stage `DISCONNECT` . • Updated the error message for a `join()` timeout error. Instead of "InitialConnectTimedOut after 10 seconds," the SDK now returns "Operation timed out."

April 30, 2024

IVS Broadcast SDK: Web 1.10.1 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.10.1	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Minor bug fixes.

April 30, 2024

Amazon IVS Broadcast SDK: Android 1.15.2, iOS 1.15.2 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.15.2	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/android • Minor bug fixes. Upgrade to this version only if you have a specific reason to do so; otherwise, use the highest version that is released.
iOS Broadcast SDK 1.15.2	**Download for real-time streaming:** https://broadcast.live-video.net/1.15.2/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/ios • Minor bug fixes. Upgrade to this version only if you have a specific reason to do so;

Platform	Downloads and Changes
	otherwise, use the highest version that is released.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.244 MB	13.198 MB
armeabi-v7a	4.543 MB	9.192 MB
x86_64	5.437 MB	14.051 MB
x86	5.631 MB	14.461 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.359 MB	7.836 MB

April 22, 2024

Amazon IVS Broadcast SDK: Android 1.17.0, iOS 1.17.0 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.17.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/android

Platform	Downloads and Changes
	• Fixed a rare crash that can occur while publishing.
iOS Broadcast SDK 1.17.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.17.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/ios • The `AmazonIVSBroadcast` framework now includes a privacy manifest, as required by Apple.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.273 MB	13.275 MB
armeabi-v7a	4.571 MB	9.251 MB
x86_64	5.468 MB	14.137 MB
x86	5.662 MB	14.549 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.388 MB	7.916 MB

March 21, 2024

Amazon IVS Broadcast SDK: Android 1.16.0, iOS 1.16.0, Web 1.10.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.10.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed an intermittent error when cleaning up connections after unsubscribing or leaving a stage.
Android Broadcast SDK 1.16.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/android • Fixed a previews freeze on the Exynos variant of Samsung devices with Android 14. • Added a function for querying camera zoom capabilities and setting the zoom factor.
iOS Broadcast SDK 1.16.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.16.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/ios • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.253 MB	13.21 MB
armeabi-v7a	4.551 MB	9.204 MB
x86_64	5.447 MB	14.070 MB
x86	5.640 MB	14.480 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.361 MB	7.836 MB

March 13, 2024

Amazon IVS Broadcast SDK: Android 1.15.1, iOS 1.15.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.15.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.15.1/android • Fixed a rare crash when subscribing to a remote participant.
iOS Broadcast SDK 1.15.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.15.1/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.1/ios • Fixed a rare crash when subscribing to a remote participant.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.358 MB	7.820 MB

March 13, 2024

Server-Side Composition API Updates

We introduced new properties to the GridConfiguration and a new picture-in-picture layout, enhancing the customization options for compositions. For specific documentation changes, see the Document History (see the table of API Reference changes).

Important: Ensure your application does not depend on the specific features of the current layout, such as size and position of tiles. *Visual improvements to layouts can be introduced at any time*.

March 8, 2024

Server-Side Composition Layout Updates

Today we enabled the changes to the default grid layout that are described in the February 7, 2024 entry.

February 22, 2024

Amazon IVS Broadcast SDK: Android 1.15.0, iOS 1.15.0, Web 1.9.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.9.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Improved internal error handling.
Android Broadcast SDK 1.15.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/android • Minor bug fixes.
iOS Broadcast SDK 1.15.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.15.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/ios • Added an `AVPictureInPictureController` extension to allow creating a new instance with an `IVSImagePreviewView`.

Platform	Downloads and Changes
	• Added a new API on `IVSImageDevice` to create an `AVSampleBufferDisplayLayer` to which the device renders. • Fixed a low bitrate issue on devices running iOS 17 and later. • Minor bug fixes.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 MB

Broadcast SDK Size: iOS

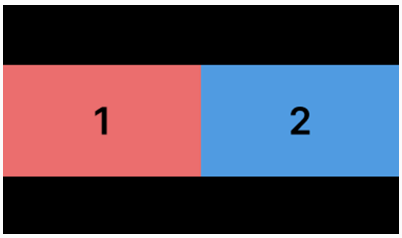
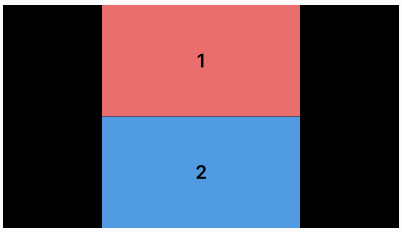
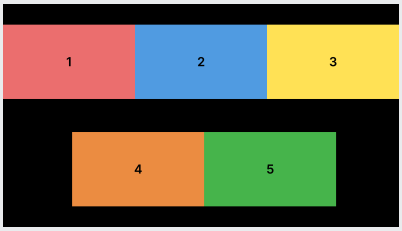
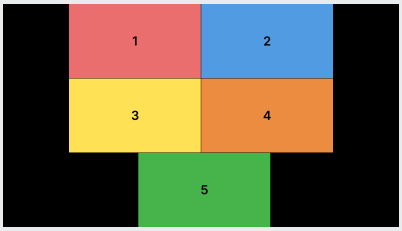
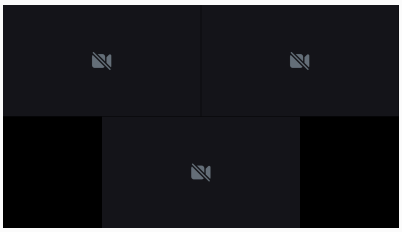
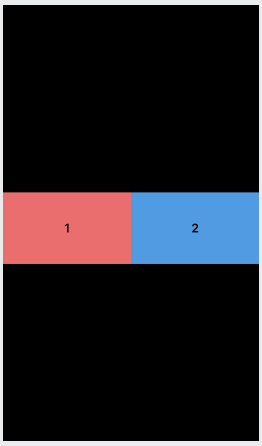
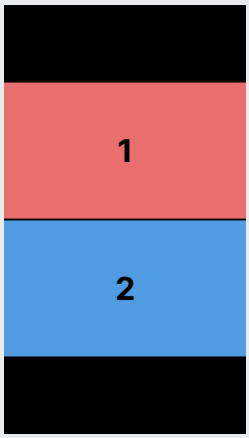
Architecture	Compressed Size	Uncompressed Size
arm64	3.358 MB	7.820 MB

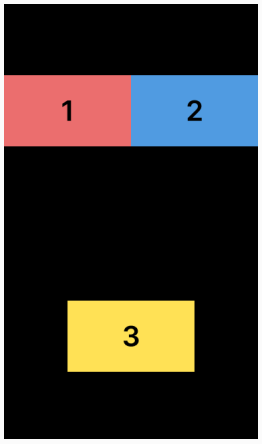
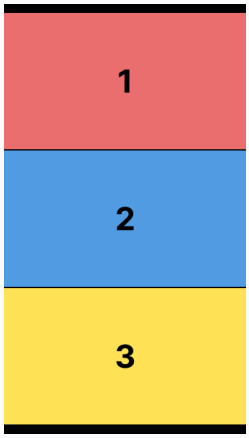
February 7, 2024

Server-Side Composition Layout Updates

This release introduces visual improvements to the default grid layout. These changes will optimize how video is displayed and reduce blank space. These changes will be enabled on March 7, 2024.

Important: Ensure your application does not depend on the specific features of the current layout, such as size and position of tiles. *Visual improvements to layouts can be introduced at any time*.

Description of the Change	Old	New
Automatically selects the optimal placement of participants to maximize video size.		
Enhances space utilization by reducing gaps and minimizing black bars.		
Adds a new “camera off” indicator for clear visibility of participants not sharing video.		
Improves space utilization and proportions for portrait use cases.		

Description of the Change	Old	New
Enhances space utilization in portrait use cases by minimizing spacing between participants and reducing letterboxing or pillarboxing.		

February 6, 2024

OBS and WHIP Support

IVS can be used with WHIP-compatible encoders like OBS to publish to IVS real-time streaming. WHIP (WebRTC-HTTP Ingestion Protocol) is an IETF draft developed to standardize WebRTC ingestion. See the new page on OBS and WHIP Support.

February 1, 2024

Amazon IVS Broadcast SDK: Android 1.14.1, iOS 1.14.1, Web 1.8.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.8.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Layered encoding with simulcast is now disabled by default. • Fixed an issue where a Stage instance would not cleanly disconnect when a Stage

Platform	Downloads and Changes
	was deleted, or when a participant was disconnected from the server. The SDK now emits a `STAGE_CONNECTION_STATE_CHANGED` event with a state of `DISCONNECTED` (instead of `ERRORED` and then `CONNECTING`). • Fixed issue where publishing would fail when updating the strategy with empty audio or video tracks.
Android Broadcast SDK 1.14.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android • Layered encoding with simulcast is now disabled by default. • Updated `libWebRTC` from M108 to M119. • Fixed several crashes to improve overall stability. • Added support for stereo publishing. This can be enabled through the `StageAudioConfiguration` object. • Fixed a bug causing a black feed from participants after joining a session. • Updated internal `libWebRTC` references to avoid symbol conflicts when other `libWebRTC` versions are included in the same host application.

Platform	Downloads and Changes
iOS Broadcast SDK 1.14.1	**Download for real-time streaming:** https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ios • Layered encoding with simulcast is now disabled by default. • Updated `libWebRTC` from M108 to M119. • Fixed several crashes to improve overall stability. • Added support for stereo publishing. This can be enabled through `IVSLocalStageStreamAudioConfiguration`. • Fixed a crash when enabling audio-only mode for other participants. • Improved TTV and reduced binary size.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.223 MB	13.118 MB
armeabi-v7a	4.524 MB	9.134 MB
x86_64	5.418 MB	13.955 MB
x86	5.61 MB	14.369 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.350 MB	7.790 MB

January 3, 2024

Amazon IVS Broadcast SDK: Android 1.13.4, iOS 1.13.4, Web 1.7.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.7.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Improved time-to-video for subscribers joining stages. • Removed the `minAudioBitrateKbps` property (it was unused). • Improved network recovery during internet outages or changes.
Android Broadcast SDK 1.13.4	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/android • StageAudioConfiguration now supports setting whether echo cancellation should be enabled.
iOS Broadcast SDK 1.13.4	**Download for real-time streaming:** https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/ios • On iOS, we improved the audio engine for both recording and playback with a focus on stability and recoverability. This enhances support for route changes while in use, improves battery recovery for edge cases, and reduces the amount of main thread blocking. • Fixed an issue where the microphone might stay active even after it was detached from a stage, leaving the iOS privacy indicator on. (The SDK was not processing incoming audio at the time.)

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.187 MB	13.025 MB
armeabi-v7a	4.491 MB	9.056 MB
x86_64	5.359 MB	13.829 MB
x86	5.553 MB	14.214 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.45 MB	7.84 MB

December 7, 2023

New CloudWatch Metrics

We renamed the PacketLoss (Stage) metric to be DownloadPacketLoss (Stage). We also released additional CloudWatch metrics for IVS real-time streaming:

- DownloadPacketLoss (Stage,Participant)
- DroppedFrames (Stage,Participant)
- SubscribeBitrate (Stage,Participant,MediaType)

For details, see Monitoring IVS Real-Time Streaming.

December 4, 2023

Amazon IVS Broadcast SDK: Android 1.13.2 and iOS 1.13.2 (Real-Time Streaming)

Platform	Downloads and Changes
All mobile (Android and iOS)	• Noise-suppression configuration is available for developers to enable/disable for publishing.
Android Broadcast SDK 1.13.2	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/android • Improved the time it takes to load the video (TTV) when joining the first stage in a session.
iOS Broadcast SDK 1.13.2	**Download for real-time streaming:** https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/ios • No changes in the real-time SDK.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.177 MB	13.01 MB
armeabi-v7a	4.485 MB	9.045 MB
x86_64	5.352 MB	13.808 MB
x86	5.547 MB	14.192 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.45 MB	7.82 MB

November 21, 2023

Amazon IVS Broadcast SDK: Android 1.13.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.13.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.13.1/android

Platform	Downloads and Changes
	• Fixed an issue that caused a crash when quickly leaving, releasing, and rejoining the same stage.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.177 MB	13.102 MB
armeabi-v7a	4.485 MB	9.046 MB
x86_64	5.353 MB	13.809 MB
x86	5.547 MB	14.192 MB

November 17, 2023

Amazon IVS Broadcast SDK: Android 1.13.0 and iOS 1.13.0 (Real-Time Streaming)

Platform	Downloads and Changes
All mobile (Android and iOS)	• Updated Streaming Optimizations. Among other things, the "Adaptive Streaming : Layered Encoding with Simulcast" feature now requires explicit opt-in and is supported only in recent versions of the SDK. • Improved the stability of stages by reducing occurrences of rare crashes. • Improved the time it takes to load the video (TTV) when joining a stage.

Platform	Downloads and Changes
	• Improved the experience with Bluetooth devices. • Optimized SDK CPU and memory usage, and reduced the library size. • Added the `StageAudioManager` class, which can be used to set audio capture and playback parameters, including presets for voice communication, media playback and more. For details, see the new page, IVS Broadcast SDK: Mobile Audio Modes. • Added a new `requestQualityStats` function to display structured quality events from WebRTC stats. • Added a new function to update the audio bitrate. It is set on `LocalStageStream` objects just like the video configuration, but through a new audio configuration object.

Platform	Downloads and Changes

Android Broadcast SDK 1.13.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/android
	• All methods on the `StageRenderer` interface are now optional.
	• Added support to `Surfaceview` -based preview for better performance. The existing `getPreview` methods in `Session` and `StageStream` continue to return a subclass of `TextureView` , but this may change in a future SDK version.
	  • If your application depends on `TextureView` specifically, you can continue with no changes. You also can switch from `getPreview` to `getPreviewTextureView` to prepare for the eventual change of what the default `getPreview` returns.
	  • If your application does not require `TextureView` specifically, we recommend switching to `getPreviewSurfaceView` for lower CPU and memory usage.
	• The SDK now implements a new type of preview called `ImagePreviewSurfaceTarget` which works with the application-provided Android Surface object. It is not a subclass of Android View, which provides better flexibility.
	• Fixed the case where `onFrame` callback for remote participant is called at the wrong time with the wrong size.

Platform	Downloads and Changes
	• `SurfaceSource # getInputSurface` is now annotated with `@Nullable` . Your code should check it before using it. • Added `UserId` and `attributes` to `ParticipantInfo` . The `UserId` and `attributes` properties are embedded in the token and applications can retrieve them via `ParticipantInfo` whenever a participant joins. • Camera capture and preview rendering now defaults to 720 x 1280 or publish resolution (whichever is greater) at 15 fps. You can adjust the resolution and/or the fps using `StageVideoConfiguration # setCameraCaptureQuality` . • `IllegalArgumentException` thrown when setting configuration properties now includes the provided value in the exception message.

Platform	Downloads and Changes
iOS Broadcast SDK 1.13.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/ios • Fixed the issue where the SDK does not change video configuration if the video configuration is updated before publishing. • Incorporated the Google fix for a LibVPX security vulnerability (CVE-2023-5217). (Note that the Android SDK did not require any changes for this issue.) • Applications using other libraries that include `libWebRTC` will no longer have conflicts with the IVS Broadcast SDK. • All methods on the `IVSStageRenderer` protocol are now marked `@optional`. • Microphones and cameras returned by our SDKs now have a guaranteed sorting order, as documented in the SDKs themselves. • Multiple cameras can now have a value of `true` for their `isDefault` property, one for each position as determined by the operating system. • Added `IVSStageAudioManager`, which allows precise control over the underlying `AVAudioSession` to enable a wider variety of use cases for Stages functionality. • Added `UserId` to `ParticipantInfo`.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.17 MB	13.00 MB
armeabi-v7a	4.48 MB	9.04 MB
x86_64	5.35 MB	13.80 MB
x86	5.54 MB	14.18 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	3.45 MB	7.84 MB

November 16, 2023

Composite Recording

This new feature enables recording the composited view of an IVS Stage to an Amazon S3 bucket. For more information, see:

- Composite Recording – This is a new page.
- Getting Started with IVS Real-Time Streaming – We added S3 endpoints to the policy in "Set Up IAM Permissions."
- Service Quotas – We added call-rate quotas for the new endpoints.
- IVS Real-Time Streaming API Reference – We added 4 StorageConfiguration endpoints and 7 objects (DestinationDetail, RecordingConfiguration, S3DestinationConfiguration, S3Detail, S3StorageConfiguration, StorageConfiguration, StorageConfigurationSummary). We also modified 3 objects (Composition, Destination, DestinationConfiguration); this affects the GetComposition response and the StartComposition request and response.

November 16, 2023

Server-Side Composition

IVS server-side composition enables clients to offload the composition and broadcasting of an IVS stage to an IVS-managed service. Server-side composition and RTMP broadcast to a channel are invoked through IVS control plane endpoints in the stage's home region. For more information, see:

- Getting Started with IVS Real-Time Streaming – We added SSC endpoints to the policy in "Set Up IAM Permissions."
- Using Amazon EventBridge with IVS Real-Time Streaming – We added new metrics.
- Server-Side Composition – This new document includes an overview and setup instructions.
- Service Quotas (Real-Time Streaming) – We added new call-rate limits and other quotas.
- Real-Time Streaming API Reference – We added 8 Composition and EncoderConfiguration endpoints and 11 objects (ChannelDestinationConfiguration, Composition, CompositionSummary, Destination, DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration, LayoutConfiguration, and Video).

In the *IVS Low-Latency Streaming User Guide*, see:

- Enabling Multiple Hosts on an IVS Stream – We added "Broadcasting a Stage: Client-Side versus Server-Side Composition" and updated "4. Broadcast the Stage."

October 16, 2023

Amazon IVS Broadcast SDK: Web 1.6.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.6.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Improved Time-To-Video (TTV).

Platform	Downloads and Changes
	• Added `maxAudioBitrate` configuration, supporting up to 128kbps of mono or stereo audio channels.

October 12, 2023

New CloudWatch Metrics and Participant Data

We released CloudWatch metrics for IVS real-time streaming. For details, see Monitoring IVS Real-Time Streaming.

We also added six fields to the Participant API object: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion`, and `sdkVersion`. This affects the GetParticipant response. See the IVS Real-Time Streaming API Reference.

October 12, 2023

Amazon IVS Broadcast SDK: Android 1.12.1 (Real-Time Streaming)

Platform	Downloads and Changes
Android Broadcast SDK 1.12.1	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.12.1/android • Fixed a bug where calling `BroadcastSession.setListener` resulted in an error.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.853 MB	16.375 MB

Architecture	Compressed Size	Uncompressed Size
armeabi-v7a	4.895 MB	10.803 MB
x86_64	6.149 MB	17.318 MB
x86	6.328 MB	17.186 MB

September 14, 2023

Amazon IVS Broadcast SDK: Web 1.5.2 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.5.2	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed a bug that prevented republishing with `refreshStrategy` when the published state enters an ERRORED state.

August 23, 2023

Amazon IVS Broadcast SDK: Web 1.5.1, Android 1.12.0, and iOS 1.12.0 (Real-Time Streaming)

Platform	Downloads and Changes
Web Broadcast SDK 1.5.1	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Fixed a bug with internal Maybe types on TypeScript 5.

Platform	Downloads and Changes
	• Added better detection for Simulcast support. • Fixed two race conditions with `refreshStrategy` when trying to publish. • Fixed a race condition with `refreshStrategy` when trying to update participants to subscribe to.
All mobile (Android and iOS)	• Fixed a rare issue where publishing action is never completed. • Improved the stability of stages by reducing occurrences of rare crashes. • Improved the stability of stages by resolving race-condition issues caused by rapid join / leave. • Added a new `setOnFrameCallback` method on `ImageDevice`. This allows observation as frames pass through the device itself, giving insight into the aspect ratio of the latest images. This method also can be used to detect when the first frame is rendered for a remote participant in a stage.
Android Broadcast SDK 1.12.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/android • Android 9 is now supported. • Improved CPU usage and performance.

Platform	Downloads and Changes
iOS Broadcast SDK 1.12.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip **Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/ios • Corrected the signature of `IVSDeviceDiscovery.createAudioSourceWithName` to return an `IVSCustomAudioSource` instead of `IVSCustomImageSource`.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.853 MB	16.375 MB
armeabi-v7a	4.895 MB	10.803 MB
x86_64	6.149 MB	17.318 MB
x86	6.328 MB	17.186 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	5.06 MB	10.92 MB

August 7, 2023

Amazon IVS Broadcast SDK: Web 1.5.0, Android 1.11.0, and iOS 1.11.0

Platform	Downloads and Changes
Web Broadcast SDK 1.5.0	**Reference documentation:** https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Added Simulcast – When enabled, this feature allows the publisher to send high- and low-quality layers of video. Subscribers automatically select their optimal quality based on their network conditions. See Optimizing Media.
All mobile (Android and iOS)	Added Simulcast – When enabled, this feature allows the publisher to send high- and low-quality layers of video. Subscribers automatically select their optimal quality based on their network conditions. See “Enable/Disable Layered Encoding with Simulcast” in the Android and iOS Broadcast SDK Guides.
Android Broadcast SDK 1.11.0	**Reference documentation:** https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/android • Fixed an issue where creating many stages eventually results in a crash. (The exact number of stages depends on the device.)
iOS Broadcast SDK 1.11.0	**Download for real-time streaming:** https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip

Platform	Downloads and Changes
	Reference documentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/ios • Corrected the signature of `IVSDeviceDiscovery.createAudioSourceWithName` to return `IVSCustomAudioSource` instead of `IVSCustomImageSource`.

Broadcast SDK Size: Android

Architecture	Compressed Size	Uncompressed Size
arm64-v8a	5.811 MB	16.186 MB
armeabi-v7a	4.857 MB	10.646 MB
x86_64	6.108 MB	17.122 MB
x86	6.289 MB	16.994 MB

Broadcast SDK Size: iOS

Architecture	Compressed Size	Uncompressed Size
arm64	5.030 MB	10.810 MB

August 7, 2023

Real-Time Streaming

Amazon Interactive Video Service (IVS) Real-Time Streaming enables you to deliver live streams with a latency that can be under 300 milliseconds from host to viewer.

Major documentation changes accompany this release. The IVS documentation landing page now has separate sections for real-time streaming and low-latency streaming. Each section has its own User Guide and API Reference. For documentation details, see the Document History (for both real-time and low-latency documentation changes). For real-time streaming, start with the IVS Real-Time Streaming User Guide and IVS Real-Time Streaming API Reference.