Bi-Directional Audio Streaming Integration Document

This document outlines the Web Socket (WS) event structure for bi-directional audio streaming between our system and an endpoint. It enables to send us the voice data along with the information of the caller to the endpoint (webSocket) which can return the voice data back and it would be played out to the caller. The integration involves sending and receiving audio data and metadata in real-time using predefined event types.

---

1. Overview

The integration stream audio data and metadata between our system and the webSocket endpoint. The following events are exchanged to manage the audio stream lifecycle:

- **Events Sent to the Vendor (endpoint)**: Used to initiate, manage, and terminate the audio stream.
- **Events Received from the Vendor (endpoint)**: Used to receive audio data and metadata from the vendor.

---

2. Events Sent to the Vendor

2.1. Connected

The `connected` event acts as a handshake response and sets expectations between the client and server. It is the first message sent after establishing the Web Socket connection.

We can send the `connected` event once a Web Socket connection is established. This is the first message the Web Socket server receives.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `connected`.

Payload Structure:

```
{
 "event": "connected"
}
```

2.2. Start Message

The `start` message contains metadata about the Stream and is sent immediately after the `connected` message. It is only sent once at the start of the Stream.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `start`.
`sequenceNumber`	Number used to keep track of message sending order. The first message has a value

	of `1` and then is incremented for each subsequent message.
`start`	An object containing Stream metadata
`start.streamSid`	The unique identifier of the Stream
`start.accountSid`	The unique identifier of the Account for which the stream was created
`start.callSid`	The unique identifier of the Call for which the Stream was started
`from`	The number from which the call was originated to the above-mentioned account.
`to`	The number of the account to which the as originated to.
`start.mediaFormat`	An object containing the format of the payload in the `media` messages.
`start.mediaFormat.encoding`	The encoding of the data in the upcoming payload. Value is always `audio/x-mulaw`. (also known as G.711 μ-law (PCMU))
`start.mediaFormat.sampleRate`	The sample rate in hertz of the upcoming audio data. Value is always `8000`
`start.mediaFormat.bitRate`	The number of bits used to represent one second of audio in the input audio data. Value is always `64` **kbps**.
`start.mediaFormat.bitDepth`	It refers to the number of bits used to represent each sample. (`8-bit`)
`start.customParameters`	An object containing the custom parameters that were set when defining the Stream
`streamSid`	The unique identifier of the Stream

Payload Structure:

```
{
  "event": "start",
  "sequenceNumber": "1",
  "start": {
    "accountSid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "callSid": "CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "from": "XXXXXXXXXX",
    "to": "XXXXXXXXXX",
    "mediaFormat": {
        "encoding": "audio/x-mulaw",
        "sampleRate": 8000,
        "bitRate": 64,
        "bitDepth": 8 },
    "customParameters": {
     "FirstName": "Jane",
```

```
      "LastName": "Doe",
      "RemoteParty": "Bob",
    },
  },
  "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

2.3. Media message

This message type encapsulates the raw audio data.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `"media"`.
`sequenceNumber`	Number used to keep track of message sending order. The first message has a value of `1` and then is incremented for each subsequent message.
`media`	An object containing media metadata and payload
`media.chunk`	The chunk for the message. The first message will begin with `1` and increment with each subsequent message.
`media.timestamp`	Presentation Timestamp in Milliseconds from the start of the stream.
`media.payload`	Raw audio encoded packets in base64
`streamSid`	The unique identifier of the Stream

Payload Structure:

```
{
 "event": "media",
 "sequenceNumber": "3",
 "media": {
   "chunk": "1",
   "timestamp": "5",
   "payload": "no+JhoaJjpz..."
 },
 "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

2.4. Stop Message

This message indicates when the Stream has stopped, or the call has ended.

Property	Description

`event`	Describes the type of Web Socket message. In this case, `stop`.
`sequenceNumber`	Number used to keep track of message sending order. The first message has a value of `1` and then is incremented for each subsequent message.
`stop`	An object containing Stream metadata
`stop.accountSid`	The Account identifier that created the Stream
`stop.callSid`	The Call identifier that started the Stream
`stop.reason`	The reason for ending the Stream.
`streamSid`	The unique identifier of the Stream

Payload Structure:

```
{
 "event": "stop",
 "sequenceNumber": "5",
 "stop": {
    "accountSid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "callSid": "CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "reason": "The caller disconnected the call"
  },
  "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

2.5 DTMF message

A `dtmf` message is sent when someone presses a touch-tone number key in the inbound stream, typically in response to a prompt in the outbound stream.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `dtmf`.
`streamSid`	The unique identifier of the Stream
`sequenceNumber`	Number used to keep track of message sending order. The first message has a value of `1` and then is incremented for each subsequent message.
`dtmf.digit`	the number-key tone detected

An example `dtmf` message is shown below. The `dtmf.digit` value is `1`, indicating that someone pressed the `1` key on their handset.

Payload:

```
{
```

```
  "event": "dtmf",
  "streamSid":"MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "sequenceNumber":"5",
  "dtmf": {
      "digit": "1"
  }
}
```

2.6. Mark message

When endpoint sends a media message, it could then send a mark message with a label; When that `media` message's playback is complete, we send the `mark` message to the endpoint using the same label `mark.name` indicating that the media has been played.

If the endpoint (WebSocket server) sends a clear message, we will empty the audio buffer and send back the `mark` messages matching any remaining `mark` messages from the server.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `"mark"`.
`sequenceNumber`	Number used to keep track of message sending order. The first message has a value of `1` and then is incremented for each subsequent message.
`streamSid`	The unique identifier of the Stream
`mark`	An object containing the mark metadata
`mark.name`	A custom value. We send back the `mark.name` you specify when it receives a `mark` message

Payload Structure:

```
{
 "event": "mark",
 "sequenceNumber": "4",
 "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
 "mark": {
   "name": "mark label"
 }
}
```

---

3. Events Received from the Vendor

3.1. Media

The payload must be encoded `audio/x-mulaw` with a sample rate of `8000` encoded with base64 PCM mono audio. The audio can be of any size.

Property	Description

`event`	Describes the type of Web Socket message. In this case, `"media"`.
`streamSid`	The SID of the Stream that should play the audio
`media`	An object containing the media payload
`media.payload`	Raw `mulaw/8000` audio in encoded in base64
`media.chunk`	The chunk for the message. The first message will begin with `1` and increment with each subsequent message.

Payload Structure:

```
{
  "event": "media",
  "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "media": {
    "payload": "a3242sa...",
    "chunk" : 1
  }
}
```

3.2. Mark

Sends a `mark` event message after sending a `media` event message to be notified when the audio that they have sent has been completed. We send back a `mark` event with a matching `name` when the audio ends (or if there is no audio buffered).

The Web Socket Server also receives an incoming `mark` event message if the buffer was cleared using the `clear` event message.

Property	Description
`event`	Describes the type of Web Socket message. In this case `"mark"`.
`streamSid`	The SID of the Stream that should receive the mark
`mark`	An object containing mark metadata and payload
`mark.name`	A name specific to your needs that will assist in recognizing future received `mark` event

Payload Structure:

```
{
 "event": "mark",
 "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
 "mark": {
   "name": "my label"
```

```
 }
}
```

3.3. Clear

Sends a `clear` message if the server want to interrupt the audio that has been sent in various `media` messages. This empties all buffered audio and causes any `mark` messages to be sent back to the Web Socket server.

Property	Description
`event`	Describes the type of Web Socket message. In this case, `"clear"`.
`streamSid`	The SID of the Stream in which you wish to interrupt the audio.

Payload Structure:

```
{
 "event": "clear",
 "streamSid": "MZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
}
```

---

4. Stream Lifecycle

1. **Connection Establishment**:
   - Establish a Web Socket connection with the vendor.
   - Send the `connected` event to initiate the handshake.
2. **Stream Initialization**:
   - Send the `start` event with stream metadata.
3. **Audio Streaming**:
   - Send `media` events with base64 encoded audio data.
   - Receive `media` events from the vendor with base64 encoded audio data.
4. **Stream Termination**:
   - Send the `stop` event to terminate the stream.
   - Handle the `clear` event from the vendor to reset the stream.
5. **End of Input**:
   - Send the `mark` event when all media from the bot has been played.
   - Handle the `mark` event from the vendor to denote the end of input.

---

5. Example Workflow

1. **Client to Vendor**:
   - Send `connected` → `start` → `media` → `stop`.
2. **Vendor to Client**:
   - Receive `media` → `mark` → `clear`.

---

6. Notes

- Ensures that the `streamSid` is unique for each stream and consistent across all events for a given stream.
- Base64 encoding is used for audio data to ensure compatibility and ease of transmission.
- The `mark` event is used to synchronize the end of input between the client and vendor.

Version	Date	Comment
Current Version (v. 8)	**Feb 25, 2025 16:56**	**Kartikay Saini**
v. 7	Feb 24, 2025 16:58	Kartikay Saini
v. 6	Feb 21, 2025 14:09	Kartikay Saini
v. 5	Jan 28, 2025 15:45	Ansh Tyagi
v. 4	Jan 28, 2025 15:43	Ansh Tyagi
v. 3	Jan 28, 2025 15:43	Ansh Tyagi
v. 2	Jan 28, 2025 14:24	Ansh Tyagi
v. 1	Jan 28, 2025 13:36	Ansh Tyagi