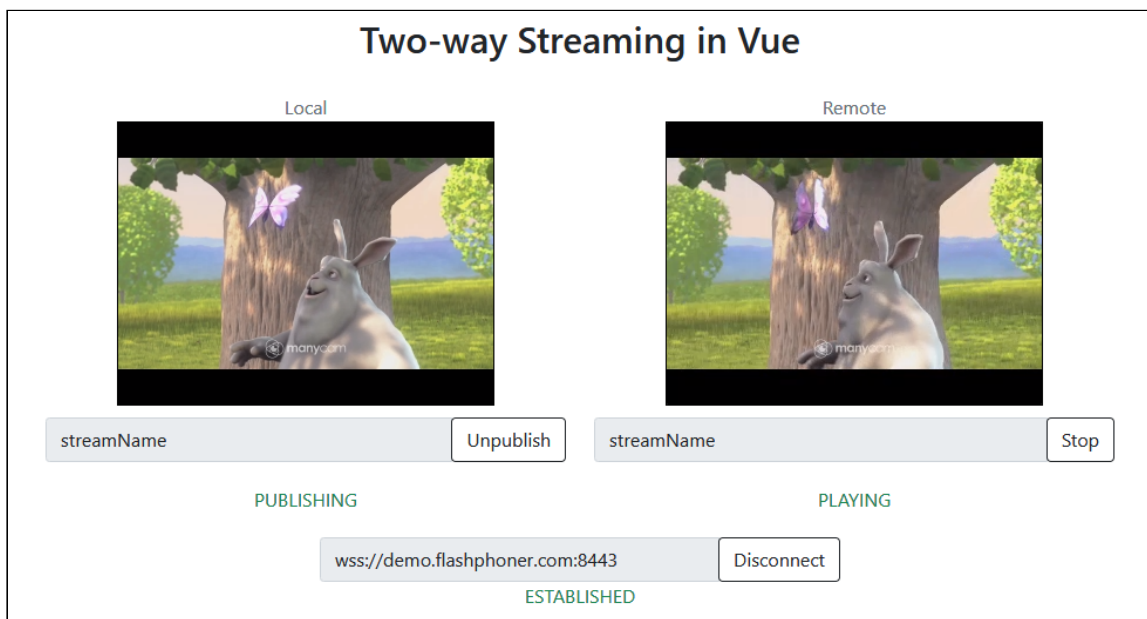


Two Way Streaming Vue

Overview

Two Way Streaming React application shows how to use Web SDK in Vue.js application to publish and play WebRTC stream



The project is available on [GitHub](#) and in [Web SDK build archives](#) since build 2.0.202 by the following path `examples/react/two-way-streaming-vue`.

Building the project

1. Download WebSDK source code

```
git clone https://github.com/flashphoner/flashphoner_client.git
```

2. Go to the example folder

```
cd flashphoner_client/examples/vue/two-way-streaming-vue
```

3. Install dependencies

```
npm install
```

4. Build for local testing

```
npm run serve
```

or to deploy to your web server

```
npm run build
```

Analyzing example code

To analyze the code take version with hash `6dc44b8` which is available [here](#)

Application code is in `TwoWayStreamingApp.vue` file, additional functions are in `fp-utils.js` file

1. API import

[code](#)

```
import * as FPUtills from './fp-utils.js';  
import * as Flashphoner from '@flashphoner/websdk';
```

2. API initialization

`Flashphoner.init()` [code](#)

```
onLoad() {  
  try {  
    Flashphoner.init({});  
    ...  
  }  
  catch(e) {  
    console.log(e);  
    ...  
  }  
}
```

3. Connecting to the server and receiving the event confirming connection is established successfully

`Flashphoner.createSession()`, `SESSION_STATUS.ESTABLISHED` [code](#)

```
onConnectClick() {  
  let url = this.serverUrl;  
  let session = this.session;
```

```

if (!session) {
  console.log("Create new session with url " + url);
  this.connectButtonDisabled = true;
  this.serverUrlDisabled = true;
  Flashphoner.createSession({urlServer:
url}).on(SESSION_STATUS.ESTABLISHED, (session) => {
  this.sessionStatus = SESSION_STATUS.ESTABLISHED;
  this.sessionStatusClass = 'text-success';
  this.onConnected(session);
}).on(SESSION_STATUS.DISCONNECTED, () => {
  ...
}).on(SESSION_STATUS.FAILED, () => {
  ...
});
}
...
}

```

4. Stream publishing

`Session.createStream()`, `Stream.publish()` [code](#)

```

publishStream() {
  let session = this.session;
  let streamName = this.publishStreamName;
  let localVideo = this.localVideo;

  if(session && localVideo) {
    session.createStream({
      name: streamName,
      display: localVideo,
      cacheLocalResources: true,
      receiveVideo: false,
      receiveAudio: false
    }).on(STREAM_STATUS.PUBLISHING, (stream) => {
      ...
    }).on(STREAM_STATUS.UNPUBLISHED, () => {
      ...
    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).publish();
  }
}

```

5. Receiving the event confirming the stream is successfully published

`STREAM_STATUS.PUBLISHING` [code](#)

```

publishStream() {
  let session = this.session;
  let streamName = this.publishStreamName;
  let localVideo = this.localVideo;

```

```

if(session && localVideo) {
  session.createStream({
    ...
  }).on(STREAM_STATUS.PUBLISHING, (stream) => {
    this.publishStatus = STREAM_STATUS.PUBLISHING;
    this.publishStatusClass = 'text-success';
    this.onPublishing(stream);
  }).on(STREAM_STATUS.UNPUBLISHED, () => {
    ...
  }).on(STREAM_STATUS.FAILED, () => {
    ...
  }).publish();
}
}

```

6. Stream playback with picture resizing to div size

`Session.createStream()`, `Stream.play()`, `STREAM_STATUS.PENDING`,
`FPUtills.resizeVideo()` [code](#)

```

playStream() {
  let session = this.session;
  let streamName = this.playStreamName;
  let remoteVideo = this.remoteVideo;

  if(session && remoteVideo) {
    session.createStream({
      name: streamName,
      display: remoteVideo
    }).on(STREAM_STATUS.PENDING, (stream) => {
      let video = document.getElementById(stream.id());
      if (!video.hasListeners) {
        video.hasListeners = true;
        video.addEventListener('resize', (event) => {
          FPUtills.resizeVideo(event.target);
        });
      }
    }).on(STREAM_STATUS.PLAYING, (stream) => {
      ...
    }).on(STREAM_STATUS.STOPPED, () => {
      ...
    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).play();
  }
}

```

7. Receiving the event confirming successful playback

`STREAM_STATUS.PLAYING` [code](#)

```

playStream() {
  let session = this.session;

```

```

let streamName = this.playStreamName;
let remoteVideo = this.remoteVideo;

if(session && remoteVideo) {
  session.createStream({
    name: streamName,
    display: remoteVideo
  }).on(STREAM_STATUS.PENDING, (stream) => {
    ...
  }).on(STREAM_STATUS.PLAYING, (stream) => {
    this.playStatus = STREAM_STATUS.PLAYING;
    this.playStatusClass = 'text-success';
    this.onPlaying(stream);
  }).on(STREAM_STATUS.STOPPED, () => {
    ...
  }).on(STREAM_STATUS.FAILED, () => {
    ...
  }).play();
}
}

```

8. Playback stopping

`Stream.stop()` [code](#)

```

onPlayClick() {
  let stream = this.playStreamObj;
  ...

  if (!stream) {
    ...
    this.playStream();
  } else {
    this.playButtonDisabled = true;
    stream.stop();
  }
}

```

9. Receiving the event confirming playback is stopped

`STREAM_STATUS.STOPPED` [code](#)

```

playStream() {
  let session = this.session;
  let streamName = this.playStreamName;
  let remoteVideo = this.remoteVideo;

  if(session && remoteVideo) {
    session.createStream({
      name: streamName,
      display: remoteVideo
    }).on(STREAM_STATUS.PENDING, (stream) => {
      ...
    })
  }
}

```

```

    }).on(STREAM_STATUS.PLAYING, (stream) => {
      ...
    }).on(STREAM_STATUS.STOPPED, () => {
      this.playStatus = STREAM_STATUS.STOPPED;
      this.playStatusClass = 'text-success';
      this.onStopped();
    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).play();
  }
}

```

10. Publishing stopping

`Stream.stop()` code

```

onPublishClick() {
  let stream = this.publishStreamObj;
  ...

  if (!stream) {
    ...
    this.publishStream();
  } else {
    this.publishButtonDisabled = true;
    stream.stop();
  }
}

```

11. Receiving the event confirming the stream is unpublished

`STREAM_STATUS.UNPUBLISHED` code

```

publishStream() {
  let session = this.session;
  let streamName = this.publishStreamName;
  let localVideo = this.localVideo;

  if(session && localVideo) {
    session.createStream({
      ...
    }).on(STREAM_STATUS.PUBLISHING, (stream) => {
      ...
    }).on(STREAM_STATUS.UNPUBLISHED, () => {
      this.publishStatus = STREAM_STATUS.UNPUBLISHED;
      this.publishStatusClass = 'text-success';
      this.onUnpublished();
    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).publish();
  }
}

```

12. Connection closing

`Session.disconnect()` [code](#)

```
onConnectClick() {
  let url = this.serverUrl;
  let session = this.session;

  if (!session) {
    ...
  } else {
    this.connectButtonDisabled = true;
    session.disconnect();
  }
}
```

13. Receiving the event confirming the connection is closed

`SESSION_STATUS.DISCONNECTED` [code](#)

```
onConnectClick() {
  let url = this.serverUrl;
  let session = this.session;

  if (!session) {
    ...
    Flashphoner.createSession({urlServer:
url}).on(SESSION_STATUS.ESTABLISHED, (session) => {
    ...
  }).on(SESSION_STATUS.DISCONNECTED, () => {
    this.sessionStatus = SESSION_STATUS.DISCONNECTED;
    this.sessionStatusClass = 'text-success';
    this.onDisconnected();
  }).on(SESSION_STATUS.FAILED, () => {
    ...
  });
  }
  ...
}
```