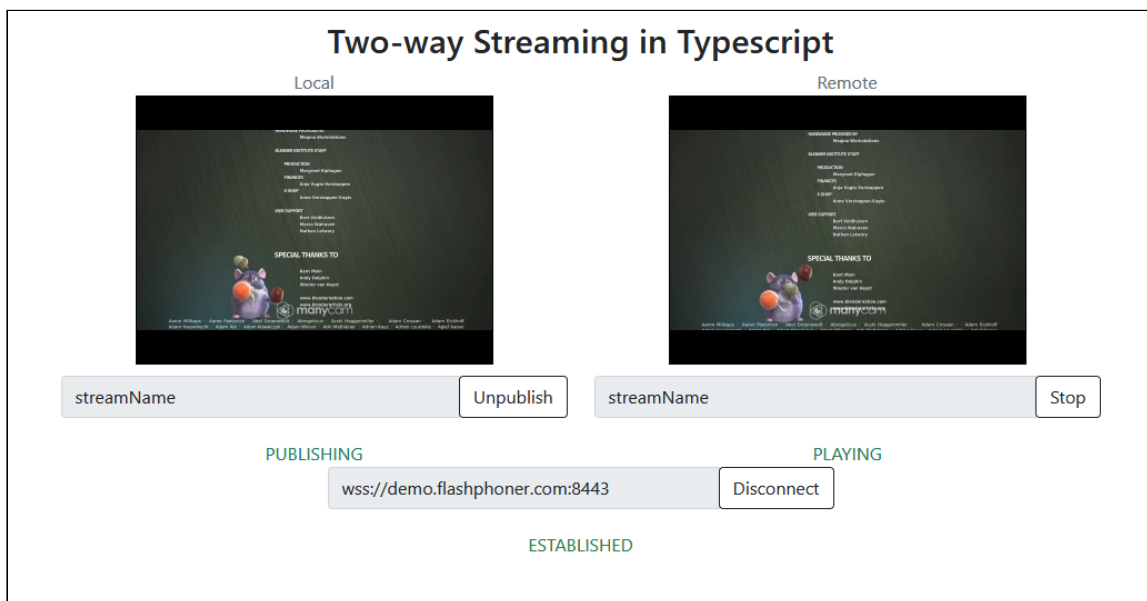


# Two Way Streaming in Typescript

## Overview

Two Way Streaming in Typescript application shows how to use Web SDK in React application written in Typescript to publish and play WebRTC stream



The project is available on [GitHub](#), and in [Web SDK build archives](#) since build 2.0.203 by the following path `examples/typescript/two-way-streaming-ts`.

## 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/typescript/two-way-streaming-ts
```

3. Install dependencies

```
npm install
```

4. Build for local testing

```
npm start
```

or to deploy to your web server

```
npm run build
```

## Analyzing example code

To analyze the code take version with hash `d9abaca` which is available [here](#)

Application code is in `TwoWayStreamingApp.tsx` file, additional functions are in `fp-utils.ts` file

### 1. API import

code

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

### 2. API initialization

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

```
componentDidMount() {  
  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 app = this;  
  let url = this.state.serverUrl;  
  let session = this.state.session;  
  
  if (!session) {
```

```

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

```

#### 4. Stream publishing

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

```

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

```

```

if(session && localVideo) {
  session.createStream({
    ...
  }).on(STREAM_STATUS.PUBLISHING, (stream) => {
    app.setState({publishStatus: STREAM_STATUS.PUBLISHING,
publishStatusClass: 'text-success'});
    app.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 app = this;
  let session = this.state.session;
  let streamName = this.state.playStreamName;
  let remoteVideo = this.state.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 app = this;
  let session = this.state.session;
  let streamName = this.state.playStreamName;
  let remoteVideo = this.state.remoteVideo;

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

```

## 8. Playback stopping

`Stream.stop()` code

```

onPlayClick = () => {
  let app = this;
  let stream = this.state.playStream;
  ...

  if (!stream) {
    ...
    app.playStream();
  } else {
    app.setState({playButtonDisabled: true});
    stream.stop();
  }
}

```

## 9. Receiving the event confirming playback is stopped

`STREAM_STATUS.STOPPED` code

```

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

  if(session && remoteVideo) {

```

```

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

```

## 10. Publishing stopping

`Stream.stop()` code

```

onPublishClick = () => {
  let app = this;
  let stream = this.state.publishStream;
  ...
  if (!stream) {
    ...
    app.publishStream();
  } else {
    app.setState({publishButtonDisabled: true});
    stream.stop();
  }
}

```

## 11. Receiving the event confirming the stream is unpublished

`STREAM_STATUS.UNPUBLISHED` code

```

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

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

```

```

    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).publish();
  }
}

```

## 12. Connection closing

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

```

onConnectClick = () => {
  let app = this;
  let url = this.state.serverUrl;
  let session = this.state.session;

  if (!session) {
    ...
  } else {
    app.setState({connectButtonDisabled: true});
    session.disconnect();
  }
}

```

## 13. Receiving the event confirming the connection is closed

`SESSION_STATUS.DISCONNECTED` [code](#)

```

onConnectClick = () => {
  let app = this;
  let url = this.state.serverUrl;
  let session = this.state.session;

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

```