

In a browser using Flash Player via RTMP

Overview

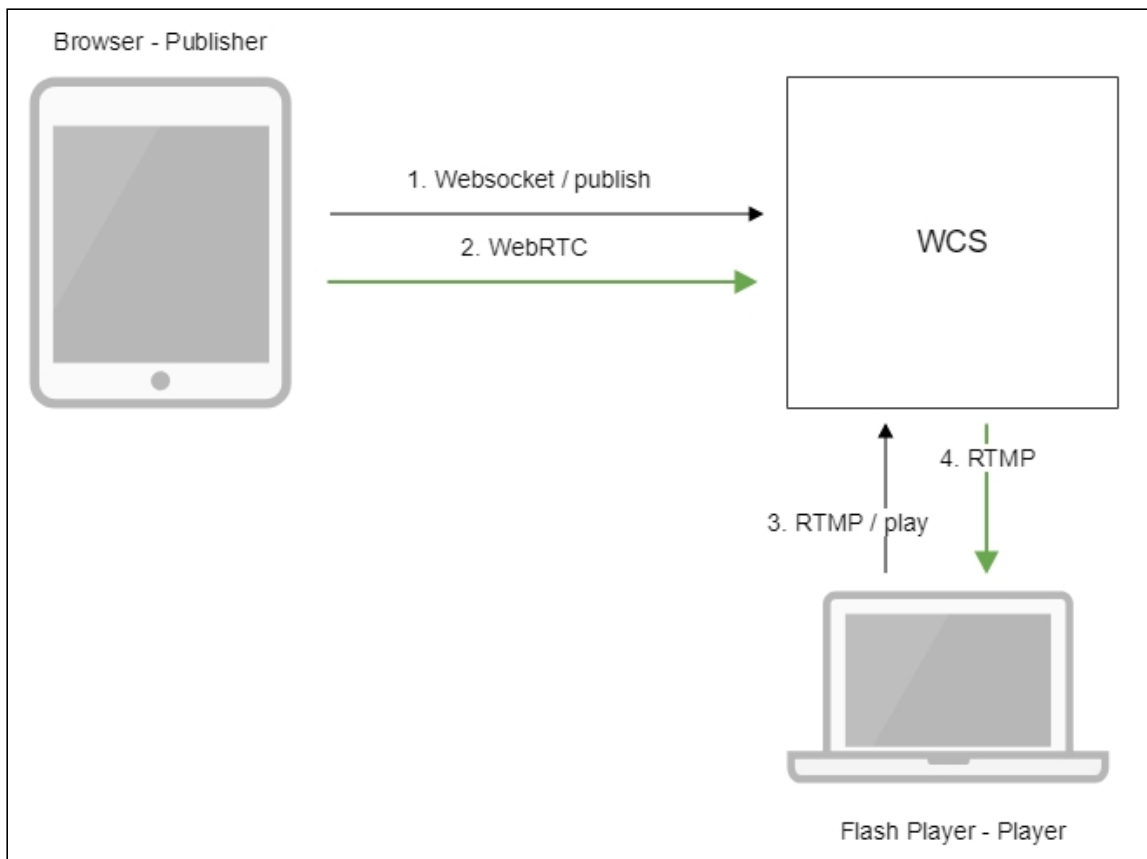
Warning

Adobe Flash Player is unsupported now in the all modern browsers. Do not use it any more. Use third party player to play a stream from Web Call Server as RTMP

Supported platforms

	Adobe Flash
Windows	✓
Mac OS	✓
Linux	✓

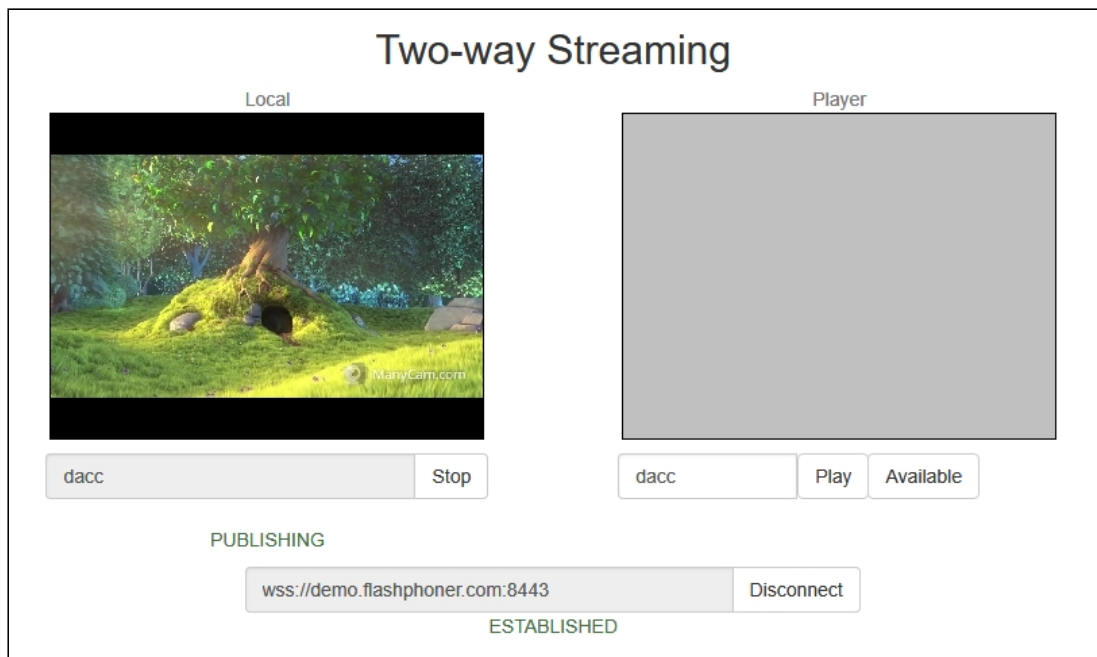
Operation flowchart



1. The browser establishes a connection via Websocket and sends the `publishStream` command.
2. The browser sends the WebRTC stream to the server.
3. Flash Player connects to the server via the RTMP protocol and sends the `play` command.
4. Flash Player receives the RTMP stream from the server.

Quick manual on testing

1. For the test we use:
2. the demo server at `demo.flashphoner.com`;
3. the [Two Way Streaming](#) web application in the Chrome browser to publish the stream;
4. the [Flash Streaming](#) web application in the Internet Explorer browser to play the stream
5. Open the Two Way Streaming application. Click `Connect`, then `Publish`. Copy the identifier of the stream:



6. Install Flash Player. Open the page of the Flash Streaming web application, and allow running Flash in the browser:

Flash Streaming

Server:

Login

Publish

Start

Play

Start



☒ audio

☒ video

width

height

fps

quality

keyframe

7. Click the **Login** button. When you see the **Connected** label, set the stream identifier in

Flash Streaming

Server:

rtmp://demo.flashphoner.com:1935

Logout

CONNECTED

Publish


Stream-KYJE

Start

Play

dacc

Start



☒ audio

☒ video

320

240

15

80

15

width

height

fps

quality

keyframe

8. Click the **Start** button in the **Play** field. The stream starts playing:

Flash Streaming

Server:

rtmp://demo.flashphoner.com:1935

Logout

CONNECTED

Publish

Stream-KYJE


Start

Play

dacc

Stop

PLAYING



☒ audio

☒ video

320

240

15

80

15

width

height

fps

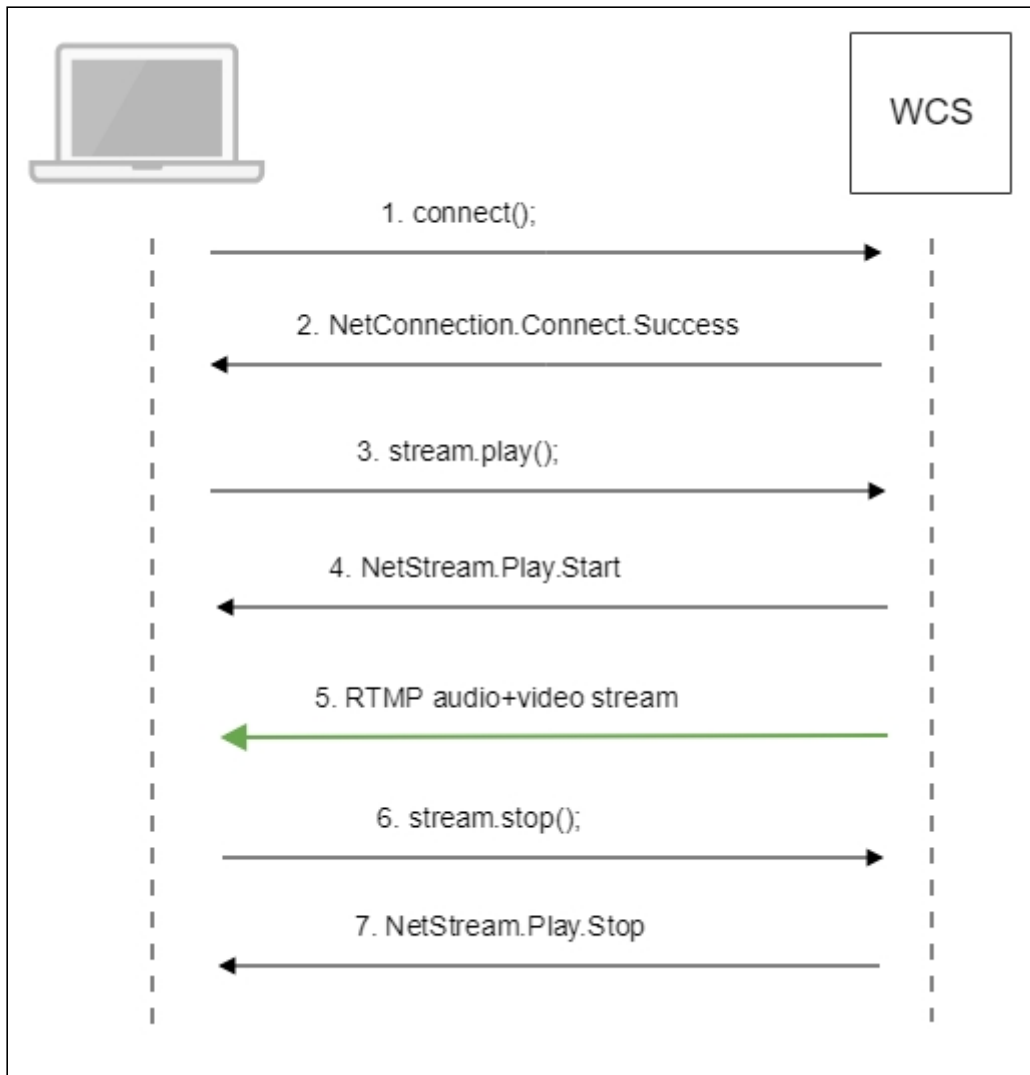
quality

keyframe

Call Flow

Below is the call flow when using the Flash Streaming example to play the stream

[streaming.mxml](#)



1. Establishing a connection to the server

`connect()` [code](#)

```
private function connect():void{
    var url:String = StringUtil.trim(connectUrl.text);
    Logger.info("connect " &#9989; url);
    nc = new NetConnection();
    //if (url.indexOf("rtmp") == 0){
    //    nc.objectEncoding = ObjectEncoding.AMF0;
    //}
    nc.client = this;
    nc.addEventListener(NetStatusEvent.NET_STATUS, handleConnectionStatus);
    var obj:Object = new Object();
    obj.login = generateRandomString(20);
    obj.appKey = "flashStreamingApp";
    nc.connect(url,obj);
}
```

2. Receiving from the server an event confirming successful connection

`NetConnection.Connect.Success` [code](#)

```
private function handleConnectionStatus(event:NetStatusEvent):void{
    Logger.info("handleConnectionStatus: " + "#9989;event.info.code);
    if (event.info.code=="NetConnection.Connect.Success"){
        Logger.info("near id: " + "#9989;nc.nearID);
        Logger.info("far id: " + "#9989;nc.farID);
        Logger.info("Connection opened");
        disconnectBtn.visible = true;
        connectBtn.visible = false;
        playBtn.enabled = true;
        publishBtn.enabled = true;
        setConnectionStatus("CONNECTED");
    } else if (event.info.code=="NetConnection.Connect.Closed" ||
event.info.code=="NetConnection.Connect.Failed"){
        ...
    }
}
```

3. Playing the stream

`stream.play()` [code](#)

```
private function addListenerAndPlay():void{
    ...
    subscribeStreamObject = createStreamObject();
    subscribeStream.play(playStreamName.text);
    videoFarEnd.attachNetStream(subscribeStream);
    videoFarEnd.width = 320;
    videoFarEnd.height = 240;
    videoFarEnd.visible = true;
}
```

4. Receiving from the server an event confirming successful playing of the stream

`NetStream.Play.Start` [code](#)

```
private function handleSubscribeStreamStatus(event:NetStatusEvent):void{
    Logger.info("handleSubscribeStreamStatus: " + "#9989;event.info.code);
    switch (event.info.code) {
        case "NetStream.Play.PublishNotify":
        case "NetStream.Play.Start":
            setPlayStatus("PLAYING");
            playBtn.visible = false;
            stopBtn.enabled = true;
            stopBtn.visible = true;
            break;
        ...
    }
}
```

5. Receiving the audio and video stream via RTMP

6. Stopping the playback of the stream

`stream.close()` [code](#)

```
private function stop():void{
    if (subscribeStream != null) {
        stopBtn.enabled = false;
        subscribeStream.close();
        subscribeStream = null;
    }
    subscribeStreamObject = null;
    videoFarEnd.visible = false;
}
```

7. Receiving from the server an event confirming the playback of the stream is stopped

`NetStream.Play.Stop` code

```
private function handleSubscribeStreamStatus(event:NetStatusEvent):void{
    Logger.info("handleSubscribeStreamStatus: "&#9989;event.info.code);
    switch (event.info.code) {
        ...
        case "NetStream.Play.UnpublishNotify":
        case "NetStream.Play.Stop":
            setPlayStatus("STOPPED");
            playBtn.enabled = true;
            playBtn.visible = true;
            stopBtn.visible = false;
            break;
        ...
    }
}
```