webrtcpeer-android Documentation

Release 1.0.4

Jukka Ahola

June 10, 2016

Contents

Contents:

CHAPTER 1

Readme

This project is part of NUBOMEDIA www.nubomedia.eu

1.1 webrtcpeer-android

The repository contains documentation for installing and utilising the webrtcpeer-android library for Android. The repository contains the documentation and the source code of the webrtcpeer-android implementation.

CHAPTER 2

Developers Guide

This documents provides information how to utilize the webrtcpeer-android library for your project.

Setup the developing environment by importing the project to Android Studio. You can import this project to your own Android Studio project via Maven (jCenter or Maven Central) by adding the following line to module's build.gradle file:

```
compile 'fi.vtt.nubomedia:webrtcpeer-android:1.0.0'
```

Source code is available at https://github.com/nubomedia-vtt/webrtcpeer-android

The Javadoc is included in the source code and can be downloaded from the link below: https://github.com/nubomedia-vtt/webrtcpeer-android/tree/master/javadoc

Support is provided through the Nubomedia VTT Public Mailing List available at https://groups.google.com/forum/#!forum/nubomedia-vtt

2.1 Usage

2.1.1 WebRTC

Initialization

First, necessary WebRTC classes have to be imported to the project:

```
import fi.vtt.nubomedia.webrtcpeerandroid.NBMMediaConfiguration;
import fi.vtt.nubomedia.webrtcpeerandroid.NBMWebRTCPeer;
import fi.vtt.nubomedia.webrtcpeerandroid.NBMPeerConnection;
// Also import NBMMediaConfiguration content for ease-of-reading when making configuration
import fi.vtt.nubomedia.webrtcpeerandroid.NBMMediaConfiguration.*;
```

Class `NBMWebRTCPeer.Observer` defines interfaces for callback functions from `NBMWebRTCPeer`. One option is to have main class implement this interface:

```
public class MyWebRTCApp implements NBMWebRTCPeer.Observer {
    /* Class declaration */
}
```

Or it can be inherited to a separate class:

```
public class MyObserver implements NBMWebRTCPeer.Observer {
    /* Class declaration */
}

public class MyWebRTCApp {
    MyObserver myObserver;
    /* Class declaration */
}
```

For rendering local content, video renderer callbacks instance must be declared:

```
VideoRenderer.Callbacks localRender = VideoRendererGui.create( args... );
```

Before creating `NBMWebRTCPeer`, the main component used to setup a WebRTC media session, a configuration object (`NBMMediaConfiguration`) must be defined:

```
/*
    Audio codec: Opus audio codec (higher quality)
    Audio bandiwidth limit: none
    Video codec: Software (VP8)
    Video renderer: OpenGLES 2.0
    Video bandwidth limit: none
    Video format: 640 x 480 @ 30fps
*/
mediaConfiguration = new NBMMediaConfiguration();
```

Default values can be changed by using an alternative constructor. Different image formats are declared in module `android.graphics.ImageFormat`.

```
import android.graphics.ImageFormat;

...

NBMVideoFormat receiverVideoFormat = new NBMVideoFormat(1280, 720, ImageFormat.YUV_420_888, 30);
NBMMediaConfiguration mediaConfiguration = new NBMMediaConfiguration(NBMRendererType.OPENGLES, NBMAu
```

`NBMWebRTCPeer` is the main component used to setup a WebRTC media session, it must be initialized with a media configuration object (`NBMMediaConfiguration`):

```
NBMWebRTCPeer nbmWebRTCPeer = new NBMWebRTCPeer(mediaConfiguration, this, localRender, myObserver);
nbmWebRTCPeer.initialize();
```

The following is a minimalistic example of implementing a class with Android WebRTC configured:

```
import org.webrtc.VideoRenderer;
import org.webrtc.VideoRendererGui;
import org.webrtc.RendererCommon;
import org.webrtc.SessionDescription;
import org.webrtc.IceCandidate;
import org.webrtc.MediaStream;
import org.webrtc.PeerConnection.IceConnectionState;
import fi.vtt.nubomedia.webrtcpeerandroid.NBMMediaConfiguration;
import fi.vtt.nubomedia.webrtcpeerandroid.NBMPeerConnection;
import fi.vtt.nubomedia.webrtcpeerandroid.NBMWebRTCPeer;

public class MyWebRTCApp implements NBMWebRTCPeer.Observer {
    VideoRenderer.Callbacks localRender;
    NBMWebRTCPeer nbmWebRTCPeer;
```

```
    public MyWebRTCApp()
    {
        localRender = VideoRendererGui.create(72,72,25,25,RendererCommon.ScalingType.SCALE_ASPECT_FI
        mediaConfiguration = new NBMMediaConfiguration();
        nbmWebRTCPeer = new NBMWebRTCPeer(mediaConfiguration, this, localRender, this);
        nbmWebRTCPeer.initialize();
    }

    /* Observer methods and the rest of declarations */
    public void onLocalSdpOfferGenerated(SessionDescription localSdpOffer, NBMPeerConnection connecti
    public void onLocalSdpAnswerGenerated(SessionDescription localSdpAnswer, NBMPeerConnection connec
    public void onIceCandidate(IceCandidate localIceCandidate, NBMPeerConnection connection) { ... }
    public void onIceStatusChanged(IceConnectionState state, NBMPeerConnection connection) { ... }
    public void onRemoteStreamAdded(MediaStream stream, NBMPeerConnection connection) { ... }
    public void onRemoteStreamRemoved(MediaStream stream, NBMPeerConnection connection) { ... }
    public void onPeerConnectionError(String error) { ... }
    public void onDataChannel(DataChannel dataChannel, NBMPeerConnection connection) { ... }
    public void onBufferedAmountChange(long l, NBMPeerConnection connection) { ... }
    public void onStateChange(NBMPeerConnection connection) { ... }
    public void onMessage(DataChannel.Buffer buffer, NBMPeerConnection connection) { ... }
}
```

SDP negotiation #1 : Generate Offer

An Offer SDP (Session Description Protocol) is metadata that describes to the other peer the format to expect (video, formats, codecs, encryption, resolution, size, etc). An exchange requires an offer from a peer, then the other peer must receive the offer and provide back an answer:

```
nbmWebRTCPeer.generateOffer("connectionId");
```

When the offer is generated, a `onLocalSdpOfferGenerated` callback is triggered:

```
public void onLocalSdpOfferGenerated(SessionDescription localSdpOffer)
{
    /* Handle generated offer */
}
```

SDP negotiation #2 : Process Answer

An Answer SDP is just like an offer but a response, we can only process an answer once we have generated an offer:

```
public void onLocalSdpAnswerGenerated(SessionDescription localSdpAnswer)
{
    /* Handle answer */
}
```

Tricke ICE #1 : Gathering local candidates

After creating the peer connection a `NBMWebRTCPeer.Observer.onIceCandicate` callback will be fired each time the ICE framework has found some local candidates:

```
public void onIceCandicate(IceCandidate localIceCandidate)
{
    /* Handle local ICE candidate */
}
```

Tricke ICE #2 : Set remote candidates

```
addRemoteIceCandidate(remoteIceCandidate);
```

Set remote renderer

Each connection may invoke a remote stream addition callback function. To display the remote stream, `attachRendererToRemoteStream` can be called inside `onRemoteStreamAdded` callback function. An example for point-to-point video application:

```
private VideoRenderer.Callbacks remoteRender;

public void onRemoteStreamAdded(MediaStream stream, NBMPeerConnection connection){
    nbmWebRTCPeer.attachRendererToRemoteStream(remoteRender, stream);
}
```

Using data channels

WebRTC DataChannel between peers is available via the API. First, in your `MyWebRTCApp` declare a `DataChannel.Init`:

```
DataChannel.Init dcInit = new DataChannel.Init();
```

Please refer to Google WebRTC documentation on configuring `DataChannel.Init`. Once you have configured the object accordingly, create datachannels for your the peer connections of your choice. For example, to create one data channel for all active peers in `MyWebRTCApp`:

```
for(NBMPeerConnection c : connectionManager.getConnections()){
    c.createDataChannel("MyDataChannelLabel", mediaManager.getLocalMediaStream());
}
```

The first parameter of `createDataChannel` is a non-unique label which can be used to identify given data channel. However, all callback events of data channels have `NBMPeerConnection` as function arguments, which may provide also unique per-peer identification of the channel. In a case that multiple data channels are established to the same peer, or a quick verbose identifier is required, label is a good choice, otherwise use the `NBMPeerConnection` instance for identification. You may also save a reference, as `createDataChannel` function returns a pointer to the newly created data channel.

Closing data channels explicitly is not necessary, as they are closed along with connections. But whenever it makes sense for e.g. user experience, call `DataChannel:close()`.

Datachannels use the following callbacks:

```
// Triggered when peer opens a data channel
public void onDataChannel(DataChannel dataChannel, NBMPeerConnection connection) {
    ...
}
// Triggered when a data channel buffer amount has changed
public void onBufferedAmountChange(long l, NBMPeerConnection connection) {
    ...
}
// Triggered when a data channel state has changed. Possible values: DataChannel.State{CONNECTING,OPE
public void onStateChange(NBMPeerConnection connection) {
    ...
}
// Triggered when a message is received from a data channel
```

```
public void onMessage(DataChannel.Buffer buffer, NBMPeerConnection connection) {
    ...
}
```

CHAPTER 3

Installation Guide

This documents provides information how to compile the webrtcpeer-android library from the sources for your project.

3.1 Clone repository

```
sudo apt-get install git
git clone https://github.com/nubomedia-vtt/webrtcpeer-android.git
```

Setup the developing environment by importing the project to Android Studio. If you want to build the project from source, you need to import the third-party library via Maven by adding the following line to the module's build.gradle file

```
compile 'fi.vtt.nubomedia:utilities-android:1.0.0'
```

Support is provided through the Nubomedia VTT Public Mailing List available at https://groups.google.com/forum/#!forum/nubomedia-vtt

CHAPTER 4

License

Nubomedia webrtcpeer-android library is distributed as Open Source Software basing BSD-license available at: https://github.com/nubomedia-vtt/webrtcpeer-android/blob/master/LICENSE