Android Audio Chat

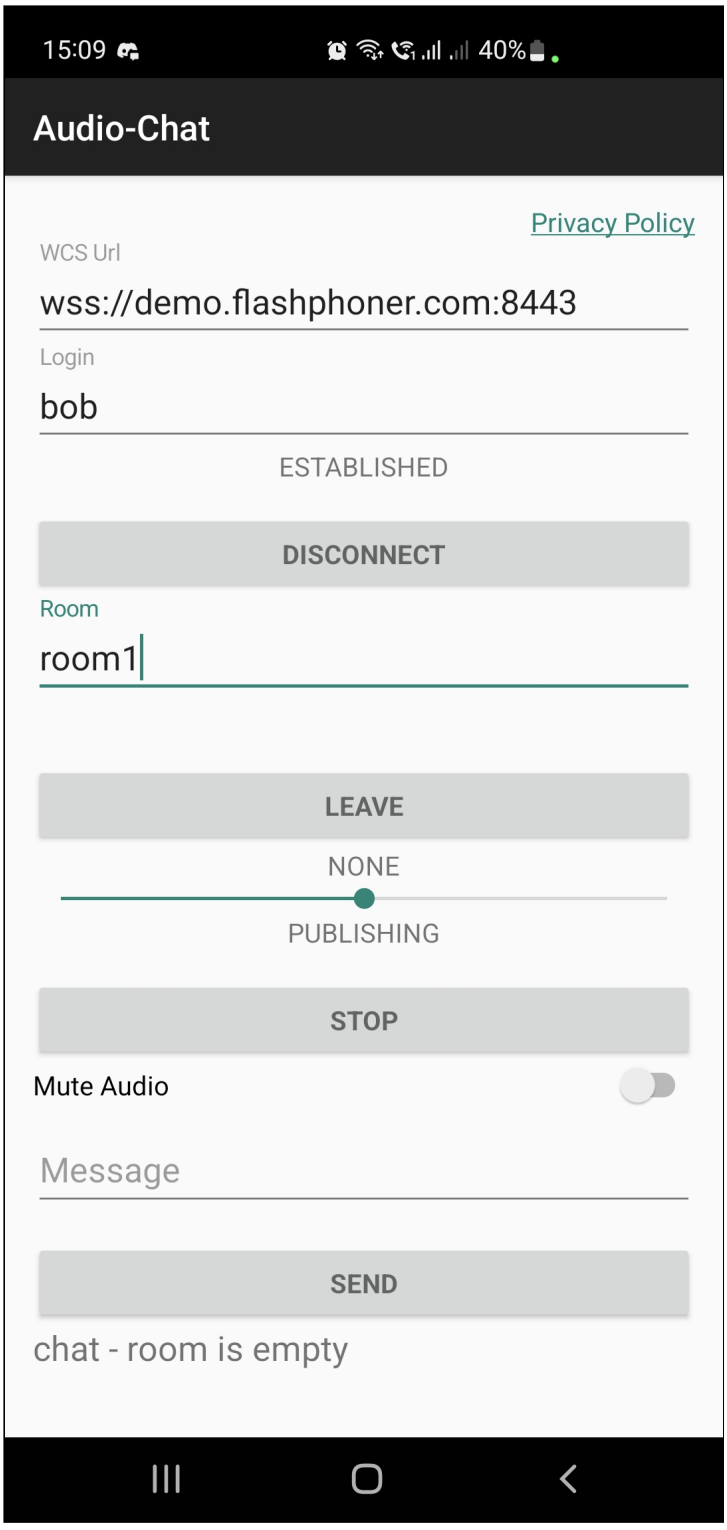
Overview

The application is used to test two participants audio chat and allows to publish and play audio only WebRTC stream from Web Call Server.

The example of audio chat in application is shown on the screenshot below.

The application uses the following input fields:

- `WCS URL`, where `wss://demo.flashphoner.com:8443/` is the WCS server address
- `Login`, where `bob` is the participant name
- `Room`, where `room1` is the chat room name



Analyzing the code

To explore the code, use the class AudioChatActivity.java of the audio chat example which is available to download in the Android SDK build 1.1.0.61.

Unlike a direct connection to server with method `createSession()`, the `RoomManager` object is used for managing connection to the server room. Connection to the server is established when `RoomManager` object is created, and method `RoomManager.join()` is called to join a room. The `Room` object is created to work with the room locally when joined. A `Participant` objects are used to work with room participants locally. All events occurring in the room (a user joined/left, or sent a message), are sent to all users connected to the room.

For example, in the following code, a user joins a room and gets the list of already connected users:

```
room = roomManager.join(roomOptions);
room.on(new RoomEvent() {
    public void onState(final Room room) {
        for (final Participant participant : room.getParticipants()) {
            ...
        }
        ...
    }
    ...
});
```

1. API initialization

`Flashphoner.init()` code

```
Flashphoner.init(this);
```

2. Create an object to render a received audio stream

`SurfaceViewRenderer` code

```
renderer = new SurfaceViewRenderer(this);
```

3. Connection to the server

`Flashphoner.createRoomManager()` code

The `RoomManagerOptions` object is passed to the method with the following parameters:

- URL of WCS server
- participant name to join a room

```
RoomManagerOptions roomManagerOptions = new
RoomManagerOptions(mWcsUrlView.getText().toString(),
mLoginView.getText().toString());

/**
```

```
 * RoomManager object is created with method createRoomManager().
 * Connection session is created when RoomManager object is created.
 */
roomManager = Flashphoner.createRoomManager(roomManagerOptions);
```

4. Receiving the event confirming successful connection

`RoomManager.onConnected()` code

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnectButton.setText(R.string.action_disconnect);
            mConnectButton.setTag(R.string.action_disconnect);
            mConnectButton.setEnabled(true);
            mConnectStatus.setText(connection.getStatus());
            mJoinButton.setEnabled(true);
        }
    });
}
```

5. Joining a room

`RoomManager.join()` code

The `RoomOptions` object with the room name is passed to the method

```
RoomOptions roomOptions = new RoomOptions();
roomOptions.setName(mJoinRoomView.getText().toString());

/**
 * The participant joins a audio chat room with method RoomManager.join().
 * RoomOptions object is passed to the method.
 * Room object is created and returned by the method.
 */
room = roomManager.join(roomOptions);
```

6. Receiving the event describing chat room state

`Room.onState()` code

The current participants array is received by `Room.getParticipants()` method. If there are more than 2 participants in the room, the current participant leaves it.

If the current participant stays in the room, the stream playback from other participant starts with `Participant.play()` method

```
if (room.getParticipants().size() >= 2) {
    room.leave(null);
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                mJoinStatus.setText("Room is full");
                mJoinButton.setEnabled(true);
            }
        }
    );
    return;
}


final StringBuffer chatState = new StringBuffer("participants: ");

/**
 * Iterating through the collection of the other participants returned by
method Room.getParticipants().
 * There is corresponding Participant object for each participant.
 */
for (final Participant participant : room.getParticipants()) {
    /**
     * A player view is assigned to each of the other participants in the
room.
     */
    chatState.append(participant.getName()).append(",");
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                participant.play(renderer);
                mParticipantName.setText(participant.getName());
            }
        }
    );
}
```

7. Audio stream publishing

`Room.publish()` code

The audio only constraints object is passed to the method

```
case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
            grantResults[0] != PackageManager.PERMISSION_GRANTED) {

        Log.i(TAG, "Permission has been denied by user");
    } else {
        /**
         * Stream is created and published with method Room.publish().
         */
        StreamOptions streamOptions = new StreamOptions();
```

```
        streamOptions.setConstraints(new Constraints(true, false));
        stream = room.publish(null, streamOptions);
        ...
        Log.i(TAG, "Permission has been granted by user");
    }
}
```

8. Receiving the other participant joined notification event

`Room.onJoined()` code

```
@Override
public void onJoined(final Participant participant) {
    /**
     * When a new participant joins the room, a player view is assigned to
that participant.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                mParticipantName.setText(participant.getName());
                addMessageHistory(participant.getName(), "joined");
            }
        }
    );
}
```

9. Receiving the other participant publishing notification event

`Room.onPublished()` code

When the notification event is received the stream publishing by other participants is starting to play by `Participant.play()` method. The `SurfaceViewRenderer` object to render the audio stream is passed to this method

```
@Override
public void onPublished(final Participant participant) {
    /**
     * When one of the other participants starts publishing, playback of the
stream published by that participant is started.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                participant.play(renderer);
            }
        }
    );
}
```

10. Receiving the other participant messaging notification event

`Room.onMessage()` code

```
@Override
public void onMessage(final Message message) {
    /**
     * When one of the participants sends a text message, the received
message is added to the messages log.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                addMessageHistory(message.getFrom(), message.getText());
            }
        }
    );
}
```

11. Sending a message to other participant

`Participant.sendMessage()` code

```
mSendButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        String text = mMessage.getText().toString();
        if (!"".equals(text)) {
            for (Participant participant : room.getParticipants()) {
                participant.sendMessage(text);
            }
            addMessageHistory(mLoginView.getText().toString(), text);
            mMessage.setText("");
        }
    }
});
```

12. Stop audio stream publishing by Unpublish button click

`Room.unpublish()` code

```
@Override
public void onClick(View view) {
    if (mPublishButton.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublishButton.getTag())) {
        ActivityCompat.requestPermissions(AudioChatActivity.this,
                new String[]{Manifest.permission.RECORD_AUDIO},
                PUBLISH_REQUEST_CODE);
    } else {
        mPublishButton.setEnabled(false);
        /**
```

```
         * Stream is unpublished with method Room.unpublish().
         */
        room.unpublish();
    }
    ...
}
```

13. Leave the room by Leave button click

`Room.leave()` code

Server REST hook response handler is passed to the method

```
room.leave(new RestAppCommunicator.Handler() {
    @Override
    public void onAccepted(Data data) {
        runOnUiThread(action);
    }

    @Override
    public void onRejected(Data data) {
        runOnUiThread(action);
    }
});
```

14. Closing the server connection

`RoomManager.disconnect()` code

```
mConnectButton.setEnabled(false);

/**
 * Connection to WCS server is closed with method RoomManager.disconnect().
 */
roomManager.disconnect();
```

15. Mute/unmute audio stream published

`Stream.unmuteAudio()`, `Stream.muteAudio()` code

```
mMuteAudio = (Switch) findViewById(R.id.mute_audio);
mMuteAudio.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            stream.muteAudio();
        } else {
            stream.unmuteAudio();
        }
```

```
    }
});
```