Android Video Conference

Example of video conference client for Android

This example can be used to participate in video conference for three participants on Web Call Server and allows to publish WebRTC stream.

On the screenshot below the participant is connected, publishing a stream and playing streams from the other two participants.

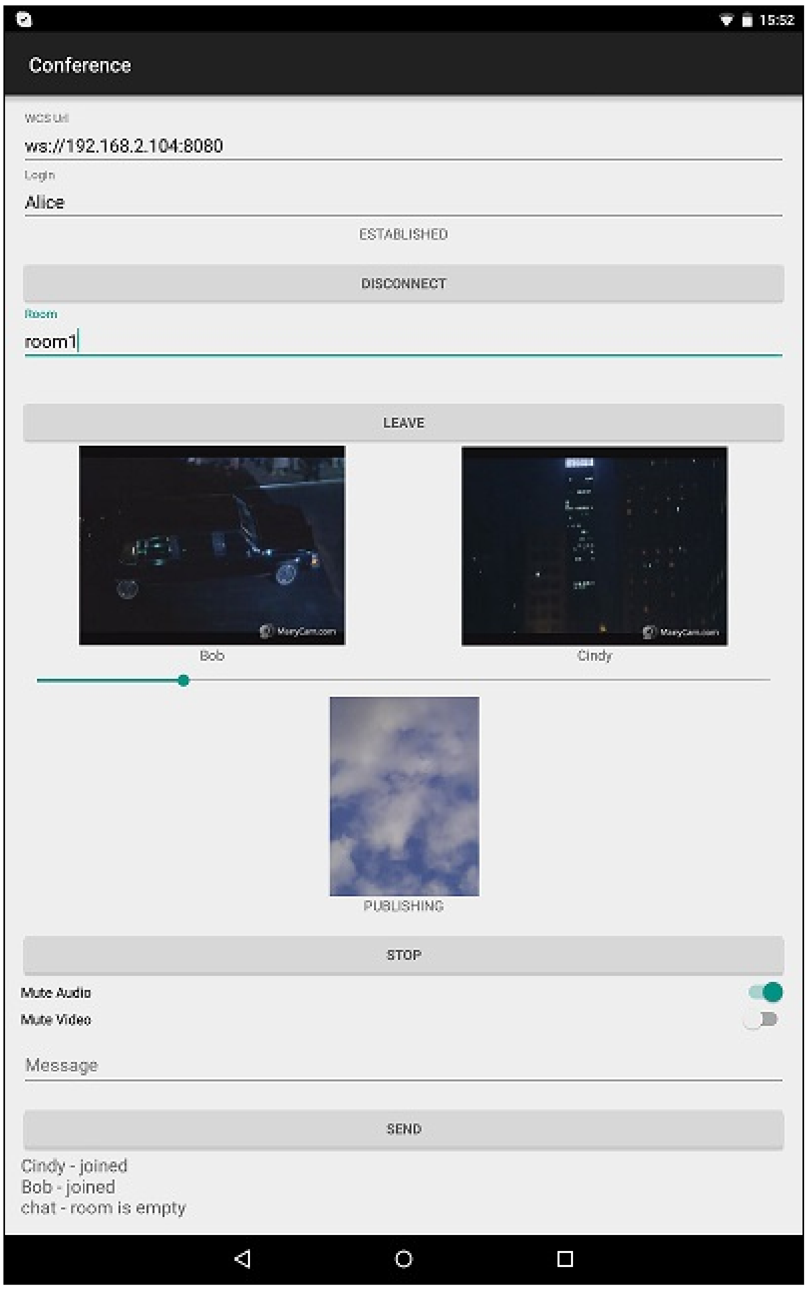
Input fields required for connecting to WCS server and joining conference

- `WCS URL`, where `192.168.2.104` is the address of the WCS server
- `Login`, where `Alice` is the username
- `Room`, where `room1` is the name of conference room

Three videos are played

- video from the camera of this participant - the lower one
- videos from the other participants (Bob and Cindy)

Volume control is located between the video elements. Controls for muting/unmuting audio and video for the published stream, input field for a text message and messages log are located below the video elements.



WCS Url: ws://192.168.2.104:8080

Login: Alice

Room: room1

Mute Audio
Mute Video

Cindy - joined
Bob - joined
chat - room is empty

Analyzing the example code

To analyze the code, let's take class ConferenceActivity.java of the `conference` example, which can be downloaded with corresponding build 1.0.1.38.

Unlike direct connection to server with method `createSession()`, the `RoomManager` object is used for managing connection to server and conference. Connection to server is established when `RoomManager` object is created, and method `RoomManager.join()` is called for joining a conference.
When joining, `Room` object is created for work with the chat room. `Participant` objects are used for work with conference participants.

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 to 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()) {
            ...
        }
        ...
    }
    ...
});
```

`ParticipantView` (`SurfaceViewRenderer` + `TextView`) is assigned for each of the other participants, to display the name and video stream of that participant (Bob and Cindy on the screenshot above).

1. Initialization of the API

`Flashphoner.init()` code

For initialization, `Context` object is passed to the `init()` method.

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

2. Connection to the server

`Flashphoner.createRoomManager()` code

`RoomManagerOptions` object with the following parameters is passed to `createRoomManager()` method:

- URL of WCS server
- username

```
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);
```

3. 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);
        }
    });
}
```

4. Joining a conference

`RoomManager.join()` code

`RoomOptions` object with the name of the conference room is passed to the `join()` method

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

/**
  * The participant joins a conference 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);
```

5. Receiving the event describing chat room state

`Room.onState()` code

On this event:

- the size of the collection of `Participant` objects returned by methodw `Room.getParticipants()` is determined to get the number of already connected participants
- if the maximum allowed number of participants had already been reached, the user leaves the room
- otherwise, appropriate changes in the interface are done and playback of video stream published by the other participants is started

```
@Override
public void onState(final Room room) {
    /**
      * After joining, Room object with data of the room is received.
      * Method Room.getParticipants() is used to check the number of already
connected participants.
      * The method returns collection of Participant objects.
      * The collection size is determined, and, if the maximum allowed number
(in this case, three) has already been reached, the user leaves the room with
method Room.leave().
      */
    if (room.getParticipants().size() >= 3) {
        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.
          */
        final ParticipantView participantView = freeViews.poll();
        if (participantView != null) {
            chatState.append(participant.getName()).append(",");
            busyViews.put(participant.getName(), participantView);

            /**
              * Playback of the stream being published by the other
participant is started with method Participant.play().
              * SurfaceViewRenderer to be used to display the video stream is
passed when the method is called.
              */
            participant.play(participantView.surfaceViewRenderer);
            ...
        }
    }
    ...
}
```

6. Video streaming when permissions to use camera and microphone are granted

`Room.publish()` code

`SurfaceViewRenderer` object to be used to display video from the camera is passed to the `publish()` method

```
case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
           grantResults[0] != PackageManager.PERMISSION_GRANTED ||
           grantResults[1] != PackageManager.PERMISSION_GRANTED) {

        Log.i(TAG, "Permission has been denied by user");
    } else {
        mPublishButton.setEnabled(false);
        /**
          * Stream is created and published with method Room.publish().
          * SurfaceViewRenderer to be used to display video from the camera
is passed to the method.
          */
        boolean record = mRecord.isChecked();
        StreamOptions streamOptions = new StreamOptions();
        streamOptions.setRecord(record);
        stream = room.publish(localRenderer, streamOptions);
        ...
        Log.i(TAG, "Permission has been granted by user");
    }
}
```

7. Receiving the event notifying that other participant joined to the room

`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.
      */
    final ParticipantView participantView = freeViews.poll();
    if (participantView != null) {
        runOnUiThread(
             new Runnable() {
                 @Override
                 public void run() {
                     participantView.login.setText(participant.getName());
                     addMessageHistory(participant.getName(), "joined");
                 }
             }
        );
        busyViews.put(participant.getName(), participantView);
    }
}
```

8. Receiving the event notifying that other participant published video stream

`Room.onPublished()` код

When one of the conference participants starts publishing, method `Participant.play()` is used to start playback of that stream. `SurfaceViewRenderer` object, in which the video will be displayed, is passed to the 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.
      */
    final ParticipantView participantView =
busyViews.get(participant.getName());
    if (participantView != null) {
        participant.play(participantView.surfaceViewRenderer);
    }
}
```

9. Receiving the event notifying that other participant sent a message

`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());
            }
       });
}
```

10. Sending a message to other room participants

`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("");
        }
    }
});
```

11. Streaming stop on `Unpublish` button pressing

`Room.unpublish()` code

```
@Override
public void onClick(View view) {
    if (mPublishButton.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublishButton.getTag())) {
        ActivityCompat.requestPermissions(ConferenceActivity.this,
              new String[]{Manifest.permission.RECORD_AUDIO,
Manifest.permission.CAMERA},
              PUBLISH_REQUEST_CODE);
    } else {
        mPublishButton.setEnabled(false);
        /**
          * Stream is unpublished with method Room.unpublish().
          */
        room.unpublish();
    }
    View currentFocus = getCurrentFocus();
    if (currentFocus != null) {
        InputMethodManager inputManager = (InputMethodManager)
getSystemService(Context.INPUT_METHOD_SERVICE);
        inputManager.hideSoftInputFromWindow(currentFocus.getWindowToken(),
InputMethodManager.HIDE_NOT_ALWAYS);
    }
}
```

12. Leaving chat room

`Room.leave()` code

Server REST hook application response handler function is passed to the metod

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

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

13. Disconnection

`RoomManager.disconnect()` code

```
mConnectButton.setEnabled(false);

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

14. Mute/unmute audio and video for stream published

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

```
/**
  * MuteAudio switch is used to mute/unmute audio of the published stream.
  * Audio is muted with method Stream.muteAudio() and unmuted with method
Stream.unmuteAudio().
  */
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();
        }
    }
});

/**
  * MuteVideo switch is used to mute/unmute video of the published stream.
  * Video is muted with method Stream.muteVideo() and unmuted with method
Stream.unmuteVideo().
  */
mMuteVideo = (Switch) findViewById(R.id.mute_video);
mMuteVideo.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            stream.muteVideo();
        } else {
            stream.unmuteVideo();
        }
    }
});
```