Android Video Conference

Пример Android-приложения для видеоконференции

Данный пример может использоваться для участия в видеоконференции для трех пользователей на Web Call Server и позволяет публиковать WebRTC-поток.

На скриншоте ниже представлен пример с конференцией, к которой присоединились два других участника.

Поля ввода, необходимые для установления соединения и присоединения к конференции

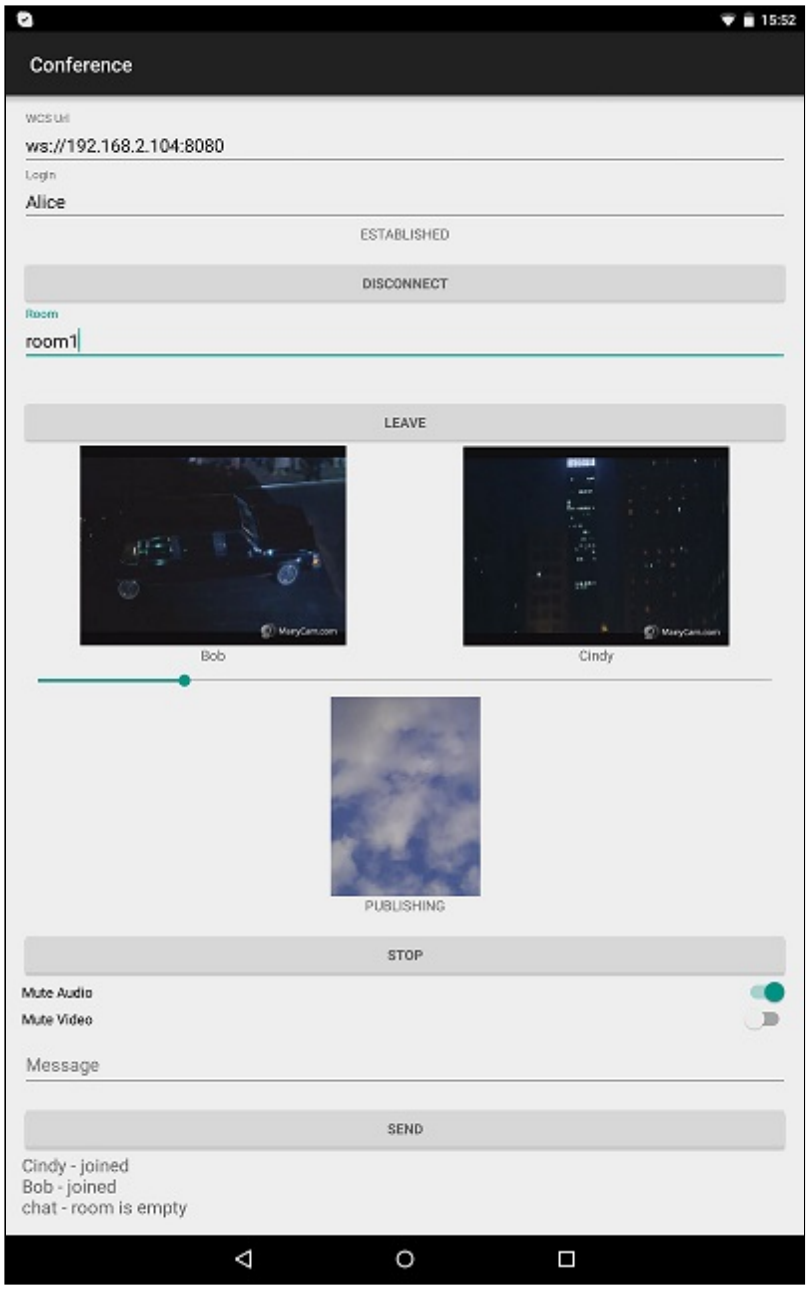
- `WCS URL`, где `192.168.2.104` - адрес WCS-сервера
- `Login`, где `Alice` - имя пользователя
- `Room`, где `room1` - имя комнаты конференции

На скриншоте вопроизводятся три видео

- нижнее - видео с камеры данного участника
- два верхних - видео от других двух участников (Bob и Cindy)

Между ними находится элемент управления для регулировки громкости.

Ниже расположены элементы управления для отключения/включения аудио и видео для публикуемого потока, поле ввода текстового сообщения и лог сообщений.



Работа с кодом примера

Для разбора кода возьмем класс ConferenceActivity.java примера `conference`, который доступен для скачивания в соответствующей сборке 1.0.1.38.

В отличии от прямого подключения к серверу методом `createSession()`, для управления подключением к серверу и конференции используется объект `RoomManager`. Соединение с сервером установливается при создании объекта `RoomManager`, а для присоединения к конфренции вызывается метод `RoomManager.join()`. При присоединении к новой комнате методом `RoomManager.join()`, создается объект `Room` для работы с этой комнатой. Для работы с участниками конференции используются объекты `Participant`.

Все события, происходящие в комнате (присоединение/выход пользователя, отправленные сообщения), транслируются другим участникам, подключенным к этой комнате.

Например, в следующем коде подключаемся к комнате и запрашиваем список других участников:

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

Каждому из других участников назначается `ParticipantView` (`SurfaceViewRenderer` + `TextView`) для отображения имени участника (Bob и Cindy на скриншоте выше) и публикуемого им потока.

1. Инициализация API

`Flashphoner.init()` code

При инициализации методу `init()` передается объект `Context`.

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

2. Подключение к серверу

`Flashphoner.createRoomManager()` code

Методу передается объект `RoomManagerOptions` со следующими параметрами

- URL WCS-сервера
- имя пользователя для присоединения к чат-комнате

```
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. Получение от сервера события, подтверждающего успешное соединение

`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. Присоединение к конференции

`RoomManager.join()` code

Методу `RoomManager.join()` передается объект `RoomOptions` с именем комнаты конференции

```
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. Получение от сервера события, описывающего состояние комнаты

`Room.onState()` code

При получении данного события:

- количество и состав других участников определяется с помощью метода `Room.getParticipants()`
- если количество участников более 3, текущий участник выходит из комнаты
- если текущий участник остается в комнате, запускается проигрывание потока от других участников при помощи `Participant.play()`

```java
@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. Публикация видеопотока

`Room.publish()` code

Методу передаются:

- `SurfaceViewRenderer localRenderer`, который будет использоваться для отображения видео с камеры
- параметр `record`, определяющий, будет ли записываться публикуемый поток

```
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. Получение от сервера события, сигнализирующего о присоединении к конференции другого участника

`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. Получение от сервера события, сигнализирующего о публикации видеопотока другим участником

`Room.onPublished()` код

При получении данного события поток, опубликованный участником, воспроизводится с помощью метода `Participant.play()`. Этому методу передается объект `SurfaceViewRenderer`, в котором будет отображаться видео

```
@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. Получение от сервера события, сигнализирующего об отправке сообщения другим участником

`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. Отправка сообщения другим участникам конференции

`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. Остановка публикации видеопотока при нажатии Unpublish

`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. Выход из комнаты конференции при нажатии `Leave`

`Room.leave()` code

Методу передается обработчик ответа REST hook приложения WCS-сервера.

```
room.leave(new RestAppCommunicator.Handler() {
    @Override
```

```
    public void onAccepted(Data data) {
        runOnUiThread(action);
    }

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

13. Закрытие соединения

`RoomManager.disconnect()` code

```
mConnectButton.setEnabled(false);

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

14. Включение/выключение аудио и видео для публикуемого потока

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