

# Android Phone

## Пример Android-приложения для аудиозвонков

Поля ввода, необходимые для установления SIP-соединения

- **WCS URL**, где **192.168.2.104** - адрес WCS-сервера
- **SIP Login** - SIP имя пользователя
- **SIP Password** - пароль
- **SIP Domain** - SIP-домен
- **SIP port** - порт

В поле **Callee** вводится SIP имя вызываемого пользователя.

При нажатии на кнопку **Connect/Disconnect** устанавливается/закрывается SIP-соединение.

При нажатии на кнопку **Call/Hangup** начинается/завершается звонок. Кнопка **Hold/Unhold** используется для удержания/снятия с удержания звонка.

## Phone-min

WCS Uri  
wss://demo.flashphoner.com:8443

Sip Login  
1000

Sip Password  
••••

Sip Domain  
192.168.0.1

Sip Port  
5060

Register required

**CONNECT**

Invite Parameters  
{header:value}

Callee  
1001

googEchoCancellation  
 googAutoGainControl  
 googNoiseSupression  
 googHighpassFilter  
 googEchoCancellation2  
 googAutoGainControl2

## Работа с кодом примера

Для разбора кода возьмем класс `PhoneMinActivity.java` примера `phone-min`, который доступен для скачивания в соответствующей сборке [1.0.1.38](#).

### 1. Инициализация API

`Flashphoner.init()` [code](#)

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

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

## 2. Создание сессии

```
Flashphoner.createSession() code
```

Методу передается объект `SessionOptions` с URL WCS-сервера.

```
SessionOptions sessionOptions = new  
SessionOptions(mWcsUrlView.getText().toString());  
session = Flashphoner.createSession(sessionOptions);
```

## 3. Подключение к серверу

```
Session.connect() code
```

Методу передается объект `Connection` с параметрами SIP-соединения

```
Connection connection = new Connection();  
connection.setSipLogin(mSipLoginView.getText().toString());  
connection.setSipPassword(mSipPasswordView.getText().toString());  
connection.setSipDomain(mSipDomainView.getText().toString());  
connection.setSipOutboundProxy(mSipDomainView.getText().toString());  
connection.setSipPort(Integer.parseInt(mSipPortView.getText().toString()));  
connection.setSipRegisterRequired(mSipRegisterRequiredView.isChecked());  
session.connect(connection);
```

## 4. Получение от сервера события, подтверждающего успешное соединение

```
Session.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);  
            if (!mSipRegisterRequiredView.isChecked()) {  
                mConnectStatus.setText(connection.getStatus());  
                mCallButton.setEnabled(true);  
            } else {  
                mConnectStatus.setText(connection.getStatus() + ".  
Registering...");  
            }  
        }  
    }  
}
```

```
});  
}
```

## 5. Обработка нажатия кнопки `Call/Hangup`

`Button.setOnClickListener()` [code](#)

```
mCallButton.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        if (mCallButton.getTag() == null ||  
Integer.valueOf(R.string.action_call).equals(mCallButton.getTag())) {  
            if ("".equals(mCalleeView.getText().toString())) {  
                return;  
            }  
            ActivityCompat.requestPermissions(PhoneMinActivity.this,  
                new String[]{Manifest.permission.RECORD_AUDIO},  
                CALL_REQUEST_CODE);  
            ...  
        } else {  
            mCallButton.setEnabled(false);  
            call.hangup();  
            call = null;  
        }  
        View currentFocus = getCurrentFocus();  
        if (currentFocus != null) {  
            InputMethodManager inputManager = (InputMethodManager)  
getSystemService(Context.INPUT_METHOD_SERVICE);  
  
            inputManager.hideSoftInputFromWindow(currentFocus.getWindowToken(),  
                InputMethodManager.HIDE_NOT_ALWAYS);  
        }  
    }  
});
```

## 6. Исходящий звонок

`Session.createCall()`, `Call.call()` [code](#)

При создании звонка в метод `session.createCall()` передается объект `CallOptions` с параметрами:

- SIP логин вызываемого аккаунта
- настройки аудио
- дополнительные параметры сообщения SIP INVITE

```
case CALL_REQUEST_CODE: {  
    if (grantResults.length == 0 ||  
        grantResults[0] != PackageManager.PERMISSION_GRANTED) {  
        Log.i(TAG, "Permission has been denied by user");  
    } else {
```

```

        mCallButton.setEnabled(false);
        /**
         * Get call options from the callee text field
         */
        CallOptions callOptions = new
CallOptions(mCalleeView.getText().toString());
        AudioConstraints audioConstraints =
callOptions.getConstraints().getAudioConstraints();
        MediaConstraints mediaConstraints =
audioConstraints.getMediaConstraints();
        ...
        try {
            Map<String, String> inviteParameters = new
Gson().fromJson(mInviteParametersView.getText().toString(),
                new TypeToken<Map<String, String>>() {
                    }.getType());
            callOptions.setInviteParameters(inviteParameters);
        } catch (Throwable t) {
            Log.e(TAG, "Invite Parameters have wrong format of json object");
        }
        call = session.createCall(callOptions);
        call.on(callStatusEvent);
        /**
         * Make the outgoing call
         */
        call.call();
        Log.i(TAG, "Permission has been granted by user");
        break;
    }
}
}

```

## 7. Получение от сервера события, сигнализирующего о входящем звонке

`Session.onCall()` code

```

@Override
public void onCall(final Call call) {
    call.on(callStatusEvent);
    /**
     * Display UI alert for the new incoming call
     */
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            AlertDialog.Builder builder = new
AlertDialog.Builder(PhoneMinActivity.this);

            builder.setTitle("Incoming call");

            builder.setMessage("Incoming call from '" + call.getCaller() +
                "'");
            builder.setPositiveButton("Answer", new
DialogInterface.OnClickListener() {
                @Override

```

```

        public void onClick(DialogInterface dialogInterface, int i) {
            PhoneMinActivity.this.call = call;
            ActivityCompat.requestPermissions(PhoneMinActivity.this,
                new String[]{Manifest.permission.RECORD_AUDIO},
                INCOMING_CALL_REQUEST_CODE);
        }
    });
    builder.setNegativeButton("Hangup", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            call.hangup();
            incomingCallAlert = null;
        }
    });
    incomingCallAlert = builder.show();
}
});
}
}

```

## 8. Ответ на входящий звонок

`Call.answer()` code

```

case INCOMING_CALL_REQUEST_CODE: {
    if (grantResults.length == 0 ||
        grantResults[0] != PackageManager.PERMISSION_GRANTED) {
        call.hangup();
        incomingCallAlert = null;
        Log.i(TAG, "Permission has been denied by user");
    } else {
        mCallButton.setText(R.string.action_hangup);
        mCallButton.setTag(R.string.action_hangup);
        mCallButton.setEnabled(true);
        mCallStatus.setText(call.getStatus());
        call.answer();
        incomingCallAlert = null;
        Log.i(TAG, "Permission has been granted by user");
    }
}
}

```

## 9. Удержание и возобновление звонка

`Call.hold()`, `Call.unhold()` code

```

mHoldButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mHoldButton.getTag() == null ||
            Integer.valueOf(R.string.action_hold).equals(mHoldButton.getTag())) {
            call.hold();
            mHoldButton.setText(R.string.action_unhold);
            mHoldButton.setTag(R.string.action_unhold);
        }
    }
}

```

```
    } else {
        call.unhold();
        mHoldButton.setText(R.string.action_hold);
        mHoldButton.setTag(R.string.action_hold);
    }
}
});
```

## 10. Посылка тонального сигнала

`Call.sendDTMF()` [code](#)

```
mDTMF = (EditText) findViewById(R.id.dtmf);
mDTMFButton = (Button) findViewById(R.id.dtmf_button);
mDTMFButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (call != null) {
            call.sendDTMF(mDTMF.getText().toString(), Call.DTMFType.RFC2833);
        }
    }
});
```

## 11. Завершение исходящего звонка

`Call.hangup()` [code](#)

```
mCallButton.setEnabled(false);
call.hangup();
call = null;
```

## 12. Завершение входящего звонка

`Call.hangup()` [code](#)

```
builder.setNegativeButton("Hangup", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        call.hangup();
        incomingCallAlert = null;
    }
});
```

## 13. Закрытие соединения

`Session.disconnect()` [code](#)

```
mConnectButton.setEnabled(false);  
session.disconnect();
```