

Android Phone

Example of Android application for audio calls

Input fields to establish SIP connection

- **WCS URL** - WCS server address
- **SIP Login** - SIP user name
- **SIP Password** - SIP password
- **SIP Domain** - SIP server address
- **SIP port** - SIP port

SIP callee user name should be set to **Callee** input field. **Connect/Disconnect** button establishes/closes SIP-connection. **Call/Hangup** button makes a SIP call or finishes it. **Hold/Unhold** button is used to hold the call.

The screenshot shows the 'Phone-min' application interface on an Android device. The status bar at the top displays the time 10:03, signal strength, Wi-Fi, and 71% battery. The app title 'Phone-min' is in a dark header. Below it, a 'Privacy Policy' link is visible. The main form contains the following fields and controls:

- WCS Url:** A text field containing 'wss://demo.flashphoner.com:8443'.
- Sip Login:** A text field containing '10006'.
- Sip Password:** A text field with masked characters '.....'.
- Sip Domain:** A text field containing 'sip.flashphoner.com'.
- Sip Port:** A text field containing '5060'.
- Register required:** A checkbox that is checked.
- CONNECT:** A large grey button at the bottom of the form.
- Auth Token:** A label at the very bottom of the screen.

CONNECT WITH TOKEN

Invite Parameters

{header:value}

Callee

10008

☐ googEchoCancellation

☐ googAutoGainControl

☐ googNoiseSupression

☐ googHighpassFilter

☐ googEchoCancellation2

☐ googAutoGainControl2

☐ googNoiseSuppression2

☐ proximitySensor

☒ speakerPhone

CALL

HOLD

dtmf

1

Analyzing the example code

To analyze the code, let's take class `PhoneMinActivity.java` of the `phone-min` example, which can be downloaded with corresponding build `1.1.0.55`.

1. Initialization of the API

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

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

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

2. Session creation

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

`SessionOptions` object with URL of WCS server is passed to the method.

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

3. Connection to the server

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

`Connection` object with parameters required to establish SIP connection is passed to the method

```
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());
connection.setKeepAlive(true);
session.connect(connection);
```

4. Receiving the event confirming successful connection

`Session.onConnected()`, `Session.getAuthToken()` [code](#)

A session token should be kept to connect to the session later

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            ...
            String token = connection.getAuthToken();
            if (token != null && !token.isEmpty()) {
                mAuthTokenView.setText(token);
                mConnectTokenButton.setEnabled(true);
            }
        }
    });
}
```

5. `Call/Hangup` button click handler

`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(PHONE_MIN_ACTIVITY.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. Outgoing call

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

`CallOptions` object with these parameters is passed to the method:

- SIP username
- audio constraints
- `SIP INVITE` parameters

```
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. Receiving the event on incoming call

`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. Answering incoming call

`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. Call hold and retrieve

`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. DTMF sending

`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. Outgoing call hangup

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

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

12. Incoming call hangup

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

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

13. Disconnection

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

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

14. Connection to an existing session using token

`Connection.setAuthToken()`, `Session.connect` [code](#)

```
mConnectTokenButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mConnectTokenButton.getTag() == null ||
Integer.valueOf(R.string.action_connect_token).equals(mConnectTokenButton.getTag()
{
            connectWithToken = true;
            String authToken = mAuthTokenView.getText().toString();
            if (authToken.isEmpty()) {
                return;
            }
            mConnectButton.setEnabled(false);
            mConnectTokenButton.setEnabled(false);
            createSession();
            Connection connection = new Connection();
            connection.setAuthToken(authToken);
            connection.setKeepAlive(true);
            session.connect(connection);
        } else {
            mConnectButton.setEnabled(false);
            mConnectTokenButton.setEnabled(false);
            session.disconnect();
        }
    }
});
```