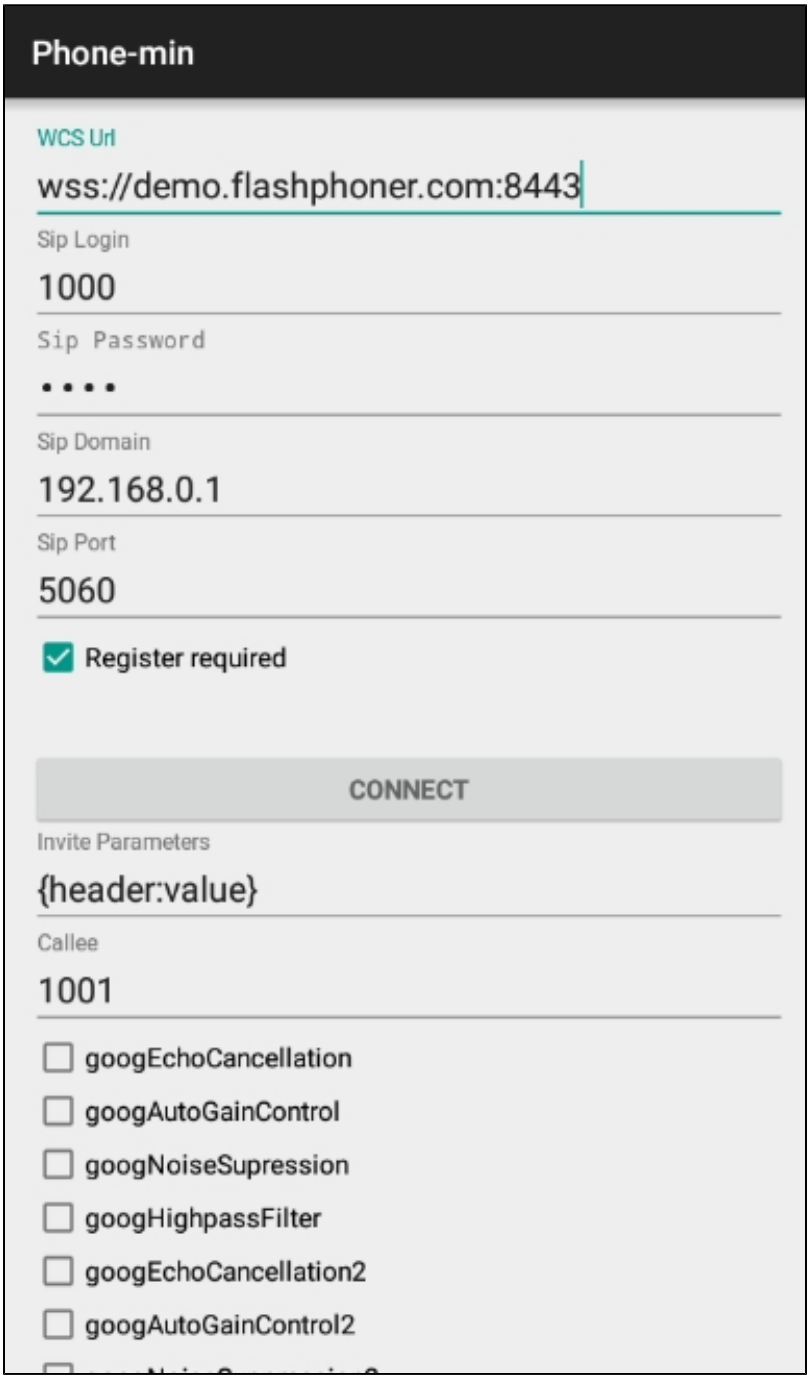
Android Phone

Example of Android application for audio calls

On the screenshot below the example is displayed when a call is established.

In the input field 'WCS URL', 192.168.2.104 is the address of the WCS server.
In the input field 'Callee', 001 is the SIP username of the callee.

SIP connection is established/closed when Connect/Disconnect button is clicked.
Call is placed/terminated when Call/Hangup button is clicked, and put on hold/retrieve when Hold/Unhold button is clicked.



Work with code of the example

To analyze the code, let's take classPhoneMinActivity.java of the phone-min example, which can be downloaded with corresponding build1.0.1.38.

1. Initialization of the API.

Flashphoner.init()code

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

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

2. Session creation.

Flashphoner.createSession()code

Object SessionOptionswith 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 objectwith parameters required for establishing 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());
session.connect(connection);
```

4. Receiving the event confirming successful connection.

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. 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(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. 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();
```

PhoneMinActivity.java