

iOS Phone

Example of iOS application for audio calls

On the screenshot below the example is displayed before a call will be established.

In the input field **WCS URL**, **wcs5-eu.flashphoner.com** is the address of the WCS server.




In **SIP...** fields, SIP parameters to register on SIP server must be entered. In the input field **Callee**, 1001 is the SIP username of the callee.

Invite parameters is field to enter additional **SIP INVITE** message parameters.

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.

No SIM 

16:50

 40 %  

wss://wcs5-eu.flashphoner.com:8443

Sip Login

1000

Sip Auth Name

1000

Sip Password

1234

Sip Domain

192.168.0.1

Sip Outbound Proxy

192.168.0.1

Sip Port

5060

Sip Register Required



NO STATUS

No SIM 16:51 40 %

Sip Port

5060

Sip Register Required ☒

NO STATUS

CONNECT

Invite Parameters

{"header":"value"}

Callee

1001

NO STATUS

CALL

HOLD

DTMF

1

DTMF

Analyzing the code of the example

To analyze the code, let's take PhoneMin example, which is available [here](#).

View class for the main view of the application: ViewController (header file [ViewController.h](#); implementation file [ViewController.m](#)).

1. Import of API

code

```
#import <FPWCSApi2/FPWCSApi2.h>
```

2. Connection to the server

`FPWCSApi2.createSession`, `FPWCSApi2Session.connect` [code](#)

`FPWCSApi2SessionOptions` object with the following parameters is passed to `createSession()` method

- URL of WCS server
- SIP parameters to make outgoing call and to receive incoming calls
- appKey of internal server-side application (defaultApp)

```
FPWCSApi2SessionOptions *options = [[FPWCSApi2SessionOptions alloc] init];
options.urlServer = _connectUrl.text;
options.sipRegisterRequired = _sipRegRequired.control.isOn;
options.sipLogin = _sipLogin.input.text;
options.sipAuthenticationName = _sipAuthName.input.text;
options.sipPassword = _sipPassword.input.text;
options.sipDomain = _sipDomain.input.text;
options.sipOutboundProxy = _sipOutboundProxy.input.text;
options.sipPort = [NSNumber numberWithInt: [_sipPort.input.text
integerValue]];
options.appKey = @"defaultApp";
NSError *error;
...
session = [FPWCSApi2 createSession:options error:&error];
...
[session connect];
```

3. Outgoing call

`FPWCSApi2Session.createCall`, `FPWCSApi2Call.call` [code](#)

The following parameters are passed to `createCall()` method:

- callee SIP username
- additional `SIP INVITE` parameters from string set by user

```
- (FPWCSApi2Call *)call {
    FPWCSApi2Session *session = [FPWCSApi2 getSessions][0];
    FPWCSApi2CallOptions *options = [[FPWCSApi2CallOptions alloc] init];
    NSString *parameters = _inviteParameters.input.text;
    if (parameters && [parameters length] > 0) {
        NSError* err = nil;
        parameters = [parameters stringByReplacingOccurrencesOfString:@" "
withString:@"\\ "];
    }
}
```

```

NSMutableDictionary *dictionary = [NSJSONSerialization
JSONObjectWithData:[parameters dataUsingEncoding:NSUTF8StringEncoding]
options:0 error:&err];
if (err) {
    NSLog(@"Error converting JSON Invite parameters to dictionary %@,
JSON %@", err, parameters);
} else {
    options.inviteParameters = dictionary;
}
}
options.callee = _callee.input.text;
//used for only recv audio
// options.localConstraints = [[FPWCSEApi2MediaConstraints alloc]
initWithAudio:NO video:NO];
// options.remoteConstraints = [[FPWCSEApi2MediaConstraints alloc]
initWithAudio:YES video:NO];
NSError *error;
call = [session createCall:options error:&error];
...
[call call];
return call;
}

```

4. Receiving the event on incoming call

`FPWCSEApi2Session.onIncomingCallCallback` [code](#)

```

[session onIncomingCallCallback:^(FPWCSEApi2Call *rCall) {
    call = rCall;

    [call on:kFPWCSECallStatusBusy callback:^(FPWCSEApi2Call *call){
        [self changeCallStatus:call];
        [self toCallState];
    }];

    [call on:kFPWCSECallStatusFailed callback:^(FPWCSEApi2Call *call){
        [self changeCallStatus:call];
        [self toCallState];
    }];

    [call on:kFPWCSECallStatusRing callback:^(FPWCSEApi2Call *call){
        [self changeCallStatus:call];
        [self toHangupState];
    }];

    [call on:kFPWCSECallStatusHold callback:^(FPWCSEApi2Call *call){
        [self changeCallStatus:call];
        [self changeViewState:_holdButton enabled:YES];
    }];

    [call on:kFPWCSECallStatusEstablished callback:^(FPWCSEApi2Call *call){
        [self changeCallStatus:call];
        [self toHangupState];
        [self changeViewState:_holdButton enabled:YES];
    }];
}];

```

```

        [call on:kFPWCSCallStatusFinish callback:^(FPWCSCall *call){
            [self changeCallStatus:call];
            [self toCallState];
            [self dismissViewControllerAnimated:YES completion:nil];
        }];
        ...
    }];
}];

```

5. Answering incoming call

`FPWCSCall.answer` [code](#)

```

alert = [UIAlertController
    alertControllerWithTitle:[NSString
        stringWithFormat:@"Incoming call from '%@'", [rCall getCallee]]
        message:error.localizedDescription
        preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction* answerButton = [UIAlertAction
    initWithTitle:@"Answer"
    style:UIAlertActionStyleDefault
    handler:^(UIAlertAction * action) {
        [call answer];
    }];

[alert addAction:answerButton];
UIAlertAction* hangupButton = [UIAlertAction
    initWithTitle:@"Hangup"
    style:UIAlertActionStyleDefault
    handler:^(UIAlertAction * action) {
        [call hangup];
    }];

[alert addAction:hangupButton];
[self presentViewController:alert animated:YES completion:nil];

```

6. Call hold and retrieve

`FPWCSCall.hold`, `FPWCSCall.unhold` [code](#)

```

- (void)holdButton:(UIButton *)button {
    [self changeViewState:button enabled:NO];
    if ([button.titleLabel.text isEqualToString:@"UNHOLD"]) {
        if (call) {
            [call unhold];
            [_holdButton setTitle:@"HOLD" forState:UIControlStateNormal];
        }
    } else {
        if (call) {
            [call hold];
            [_holdButton setTitle:@"UNHOLD" forState:UIControlStateNormal];
        }
    }
}

```

```
}  
}
```

7. DTMF sending

`FPWCSCall.sendDTMF` [code](#)

```
- (void)dtmfButton:(UIButton *)button {  
    if (call) {  
        [call sendDTMF:_dtmf.input.text type:kFPWCSCallDTMFRFC2833];  
    }  
}
```

8. Outgoing call hangup

`FPWCSCall.hangup` [code](#)

```
- (void)callButton:(UIButton *)button {  
    [self changeViewState:button enabled:NO];  
    if ([button.titleLabel.text isEqualToString:@"HANGUP"]) {  
        if ([FPWCSCall getSessions].count) {  
            [call hangup];  
        } else {  
            [self toCallState];  
        }  
        ...  
    }  
}
```

9. Incoming call hangup

`FPWCSCall.hangup` [code](#)

```
UIAlertAction* hangupButton = [UIAlertAction  
    initWithTitle:@"Hangup"  
    style:UIAlertActionStyleDefault  
    handler:^(UIAlertAction * action) {  
        [call hangup];  
    }];  
  
[alert addAction:hangupButton];
```

10. Disconnection

`FPWCSCallSession.disconnect` [code](#)

```
- (void)connectButton:(UIButton *)button {  
    [self changeViewState:button enabled:NO];  
}
```

```
        if ([button.titleLabel.text isEqualToString:@"DISCONNECT"]) {
            if ([FPWCSEApi2 getSessions].count) {
                FPWCSEApi2Session *session = [FPWCSEApi2 getSessions][0];
                NSLog(@"Disconnect session with server %@", [session
getServerUrl]);
                [session disconnect];
            } else {
                NSLog(@"Nothing to disconnect");
                [self onDisconnected];
            }
            ...
        }
    }
```