

OneCxi Flutter SDK — Developer Documentation

Version: 1.1.0 | **Package:** onecxi_flutter_sdk | **Publisher:** Ozonetel

Table of Contents

1. Introduction
 2. How It Works — Big Picture
 3. System Requirements
 4. Installation
 5. Platform Setup
 6. Quick Start — First Call in 5 Minutes
 7. Core Concepts
 8. Call Types
 9. Receiving Incoming Calls (Push Notifications)
 10. Call Controls
 11. Complete API Reference
 12. Error Handling
 13. App Lifecycle & State Sync
 14. API Keys & Secrets Management
 15. Best Practices
 16. Troubleshooting
 17. Testing Guide
 18. Support
-

1. Introduction

The **OneCxi Flutter SDK** gives your Flutter app full VoIP calling capability — inbound, outbound, and app-to-app — with a single Dart API that works on both iOS and Android.

What sets it apart:

Feature	Details
Call persistence	Calls stay alive even when the user kills the app
Lock-screen calling	Native incoming call UI on the device lock screen
Real-time audio	WebSocket-based bidirectional audio (8 kHz, 16-bit PCM)
Cross-platform	One Dart API for both iOS and Android
Native integration	CallKit (iOS) · Foreground Service (Android)
Push support	VoIP Push / PushKit (iOS) · FCM (Android)

```

Your Flutter App
|
v
OneCxi Flutter SDK  <---- Your API Keys ---->  Ozonetel Server
|
|-- WebSocket ----- Audio Stream -----|
|
+-- Method Channel
      |
+-----+-----+
v               v
iOS Native      Android Native
(CallKit,       (Foreground Service,
AVAudioEngine,  AudioRecord/AudioTrack,
PushKit)        FCM)

```

1. Your app calls `sdk.initialize(...)` with your credentials.
2. For an outgoing call, your app calls one of the `start...Call(...)` methods.
3. The SDK opens a WebSocket to the Ozonetel server and starts streaming audio.
4. For incoming calls, your server sends a push notification (FCM on Android, VoIP push on iOS). The SDK shows the native call UI automatically.
5. Call controls (mute, hold, speaker, DTMF) are sent as messages over the same WebSocket.
6. When the call ends, the SDK tears down the audio and WebSocket cleanly.

Platform	Minimum	Recommended
Flutter	3.10.0	Latest stable
Dart	3.0.0	Latest stable
iOS	13.0	16.0+
Android	API 21 (Android 5)	API 33+
Xcode	14.0	Latest
Android Studio	Arctic Fox	Latest

```
dependencies:
  flutter:
    sdk: flutter
  onecxi_flutter_sdk: ^1.1.0
```

Step 2 — Install packages:

```
flutter pub get
```

Step 3 — Complete platform setup (see next section).

5. Platform Setup

iOS Setup

5.1 — Enable background modes

In Xcode → select your target → **Signing & Capabilities** → + **Capability** → **Background Modes**.

Check all of these:

- ☒ Voice over IP
- ☒ Audio, AirPlay, and Picture in Picture
- ☒ Background processing
- ☒ Remote notifications

5.2 — Add entitlements

In your `Runner.entitlements` file:

```
<key>com.apple.developer.pushkit.unrestricted-voip-push</key>  
<true/>
```

5.3 — Add permissions to `Info.plist`:

```
<key>NSMicrophoneUsageDescription</key>  
<string>Microphone access is required to make and receive calls.</string>
```

```
<key>UIBackgroundModes</key>  
<array>  
  <string>voip</string>  
  <string>audio</string>  
  <string>background-processing</string>  
  <string>remote-notification</string>  
</array>
```

5.4 — AppDelegate setup

In `ios/Runner/AppDelegate.swift`, ensure your app delegate handles PushKit and CallKit (see the example app's `AppDelegate.swift` for the complete implementation).

Android Setup

5.5 — Add permissions to `AndroidManifest.xml`:

```

<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MICROPHONE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

```

5.6 — Register the SDK service inside <application>:

```

<service
    android:name="com.onecxi.flutter_sdk.OneCxiCallService"
    android:foregroundServiceType="microphone"
    android:exported="false"/>

```

5.7 — Add Firebase

Place your google-services.json in android/app/ and add to android/build.gradle:

```
classpath 'com.google.gms:google-services:4.4.0'
```

And to android/app/build.gradle:

```
apply plugin: 'com.google.gms.google-services'
```

6. Quick Start — First Call in 5 Minutes

Step 1 — Initialize the SDK in main.dart

```

import 'package:flutter/material.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:onecxi_flutter_sdk/onecxi_flutter_sdk.dart';

// Required: register FCM background handler BEFORE runApp()
@pragma('vm:entry-point')
Future<void> firebaseMessagingBackgroundHandler(RemoteMessage message) async {
  if (message.data.containsKey('callFrom')) {
    await OneCxiSdk().handleIncomingCallWithUI(
      Map<String, dynamic>.from(message.data),
    );
  }
}

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Android: pre-initialize background service before runApp()

```

```

    await CallBackgroundService().initialize();

    FirebaseMessaging.onBackgroundMessage(firebaseMessagingBackgroundHandler);

    runApp(const MyApp());
}

```

Step 2 — Initialize and request permissions

```

final OneCxiSdk _sdk = OneCxiSdk();

Future<void> setupSDK() async {
  await _sdk.initialize(
    accountName: 'your_account_name',
    apiKey: 'your_api_key',
    serverName: 'https://your-server.com',
  );

  await _sdk.requestPermissions(); // microphone + notifications
}

```

Step 3 — Listen for call events

```

class MyCallScreen extends StatefulWidget { ... }

class _MyCallScreenState extends State<MyCallScreen>
  implements CallProgressCallback {

  final OneCxiSdk _sdk = OneCxiSdk();

  @override
  void initState() {
    super.initState();
    _sdk.setCallProgressCallback(this); // register once
  }

  @override
  void onCallStarted(String callType, String did, String registeredNumber) {
    // Call has started - show your in-call UI
    print('Call started: $callType');
  }

  @override
  void onCallEnded(String callType, String did) {
    // Call has ended - return to idle UI
    print('Call ended');
  }
}

```

```

@override
void onCallStatusChanged(CallStatus status) {
    // Fires whenever mute / hold / speaker state changes
    setState(() {
        // update your UI toggles from status.*
    });
}

@override
void onCallError(String error, String? callType) {
    print('Error: $error');
}

@override
void onWebSocketStatusChanged(bool isConnected) {
    print('WebSocket: ${isConnected ? "connected" : "disconnected"}');
}

@override
void onCallDeclined(String callType, String callFrom) {
    print('Call declined by remote');
}
}

```

Step 4 — Make a call

```

// Inbound call (app → server)
await _sdk.startInBoundCall(
    did: '90050',
    registeredNumber: '9876543210',
);

```

That's it. The SDK handles the WebSocket, audio, and native call UI automatically.

7. Core Concepts

The SDK is a singleton

`OneCxiSdk()` always returns the **same instance**. You can call it anywhere without storing a reference:

```

// All of these refer to the exact same object:
final sdk = OneCxiSdk();
OneCxiSdk().startInBoundCall(...);

```

CallProgressCallback

This is the bridge between the SDK and your UI. Implement it in any widget (usually your main screen or a state management class) and register it once:

```
_sdk.setCallProgressCallback(this);
```

You only need one callback registered at a time. Re-registering replaces the previous one.

CallStatus

A snapshot of the current call state. Read it anytime:

```
final status = _sdk.getCallStatus();

status.isCallOngoing    // bool - is a call active?
status.isMuted           // bool - is mic muted?
status.isOnHold          // bool - is call on hold?
status.isSpeakerOn       // bool - is speaker routing audio?
status.callType           // String? - 'inbound', 'outbound', 'app_to_app'
status.currentDID         // String? - the DID used for this call
status.registeredNumber  // String? - the registered number used
```

Audio format

The SDK streams audio over WebSocket in **8 kHz, mono, 16-bit PCM** format. This is handled entirely by the SDK — you do not need to manage audio buffers yourself.

8. Call Types

8.1 Inbound Call — App calls the server

The app user dials a number (DID) that routes through your Ozonetel server.

When to use: Calling customer support, dialing into a hotline, connecting to IVR.

```
await _sdk.startInBoundCall(
  did: '90050', // The DID (server number) to call
  registeredNumber: '9876543210', // The user's registered number
);
```

8.2 Outbound Call — App calls a phone number

The call is initiated from the app but goes out through the server to a real phone number.

When to use: Click-to-call, calling a customer's number from a CRM app.

```
await _sdk.startOutBoundCall(
  userMobile: '1234567890', // The phone number to call
  registeredNumber: '9876543210', // The user's registered number
);
```

8.3 App-to-App Inbound — This user calls another app user

One app user initiates a call to another app user by their registered number.

When to use: In-app calling between agents, peer-to-peer communication.

```
await _sdk.startAppToAppInboundCall(  
  inputNumber: '1234567890',      // The other user's registered number  
  registeredNumber: '9876543210', // This user's registered number  
);
```

8.4 App-to-App Outbound — Answering an app-to-app call

Used when this device is answering an incoming app-to-app call (typically called from inside the FCM/push handler after the user accepts).

```
await _sdk.startAppToAppOutboundCall(  
  inputNumber: '1234567890',      // The caller's registered number  
  registeredNumber: '9876543210', // This user's registered number  
  callFromNumber: '5555555555',  // The number to display as caller ID  
);
```

9. Receiving Incoming Calls (Push Notifications)

Incoming calls arrive as push notifications. The SDK handles the native call UI — you just pass the payload.

iOS — VoIP Push (PushKit)

iOS delivers VoIP pushes via PushKit, which wakes the app even when it is killed. This is handled in your `AppDelegate`. After extracting the payload, call:

```
await _sdk.handleIncomingCallWithUI(pushPayload);
```

Android — FCM

Register the background handler in `main()`:

```
@pragma('vm:entry-point')  
Future<void> firebaseMessagingBackgroundHandler(RemoteMessage message) async {  
  if (message.data.containsKey('callFrom')) {  
    await OneCxiSdk().handleIncomingCallWithUI(  
      Map<String, dynamic>.from(message.data),  
    );  
  }  
}  
  
// In main():  
FirebaseMessaging.onBackgroundMessage(firebaseMessagingBackgroundHandler);
```


Push payload format

Your backend must send a data-only FCM notification (no `notification` key) with this shape:

```
{
  "data": {
    "callFrom":      "90050",
    "callTo":        "9876543210",
    "callType":      "Outbound",
    "registeredNumber": "9876543210",
    "callerName":    "Customer Support",
    "callId":        "call_abc123",
    "environment":   "PRODUCTION",
    "timestamp":     1711800000000,
    "platform":      "android",
    "event":         "initial"
  }
}
```

Required fields:

Field	Type	Description
callFrom	String	Caller's number or DID
callTo	String	Recipient's registered number
callType	String	"Outbound", "Inbound", or "APPTOAPP"
registeredNumber	String	User's registered number

Optional fields:

Field	Type	Description
callerName	String	Display name shown in the native call UI
callId	String	Unique ID for this call
environment	String	"PRODUCTION" or "DEVELOPMENT"
timestamp	int	Unix ms timestamp
event	String	"initial" for new call · "end" to terminate

Ending a call via push

To remotely terminate an ongoing call from the server side, send a push with `"event": "end"`:

```
{
  "data": {
    "callFrom": "90050",
    "callTo":   "9876543210",
    "callType": "Outbound",
    "registeredNumber": "9876543210",
    "event": "end"
  }
}
```

```
}
```

10. Call Controls

All controls use explicit boolean values — there are no toggle methods.

```
// -- Mute -----
await _sdk.muteCall(true);    // mute the microphone
await _sdk.muteCall(false);   // unmute

// -- Hold -----
await _sdk.holdCall(true);    // put the call on hold
await _sdk.holdCall(false);   // resume from hold

// -- Speaker -----
await _sdk.setSpeaker(true);  // route audio to loudspeaker
await _sdk.setSpeaker(false); // route audio to earpiece

// -- DTMF -----
_sdk.sendDTMF('1'); // valid digits: 0-9, *, #

// -- End call -----
await _sdk.endCall();
```

After each control call, `onCallStatusChanged(CallStatus)` fires so you can update your UI without tracking state yourself.

11. Complete API Reference

OneCxiSdk

Initialization

```
// Always returns the same singleton instance
final OneCxiSdk sdk = OneCxiSdk();
```

Method	Returns	Description
<code>initialize({accountName, apiKey, serverName})</code>	<code>Future<void></code>	Must be called first. Throws <code>OneCxiError</code> on failure.
<code>requestPermissions()</code>	<code>Future<bool></code>	Requests microphone + notification permissions. Returns <code>true</code> if granted.
<code>checkForPermissions()</code>	<code>Future<bool></code>	Returns <code>true</code> if all permissions are already granted.
<code>setCallProgressCallback(callback)</code>	<code>void</code>	Registers your <code>CallProgressCallback</code> implementation.

Method	Returns	Description
<code>dispose()</code>	<code>void</code>	Releases resources. Call when you no longer need the SDK.

Starting Calls All call methods throw `OneCxiError.invalidInput` if any parameter is empty or whitespace-only. They throw `OneCxiError.callInProgress` if a call is already active.

Method	Description
<code>startInBoundCall({did, registeredNumber})</code>	App dials a server DID.
<code>startOutBoundCall({userMobile, registeredNumber})</code>	App dials a phone number via server.
<code>startAppToAppInboundCall({inputNumber, registeredNumber})</code>	App calls another app user.
<code>startAppToAppOutboundCall({inputNumber, registeredNumber, callFromNumber})</code>	Answers an incoming app-to-app call.
<code>handleIncomingCallWithUI(pushPayload)</code>	Shows native call UI from a push payload.
<code>endCall()</code>	Ends the active call.

Call Controls

Method	Description
<code>muteCall(bool mute)</code>	<code>true</code> = mute, <code>false</code> = unmute.
<code>holdCall(bool hold)</code>	<code>true</code> = hold, <code>false</code> = resume.
<code>setSpeaker(bool enabled)</code>	<code>true</code> = loudspeaker, <code>false</code> = earpiece.
<code>sendDTMF(String digit)</code>	Sends a DTMF tone. Valid: 0–9, *, #.

Status

Method	Returns	Description
<code>getCallStatus()</code>	<code>CallStatus</code>	Snapshot of current call state.
<code>refreshCallStatus()</code>	<code>Future<void></code>	Re-syncs state from the native layer. Call on app resume.
<code>isInitialized</code>	<code>bool</code>	Whether <code>initialize()</code> has completed.
<code>isCallActive</code>	<code>bool</code>	Whether a call is currently in progress.
<code>isMuted</code>	<code>bool</code>	Whether the microphone is currently muted.
<code>isOnHold</code>	<code>bool</code>	Whether the call is currently on hold.
<code>isSpeakerOn</code>	<code>bool</code>	Whether speaker output is active.

Tokens & Platform

Method	Returns	Description
getPlatform()	Future<String>	Returns "ios" or "android".
getPlatformToken()	Future<String?>	Returns VoIP token (iOS) or FCM token (Android).
getVoipToken()	Future<String?>	iOS VoIP (PushKit) token.
getFcmToken()	Future<String?>	Android FCM token.

Android-Specific

Method	Description
checkPendingIncomingCall()	Checks for a call that arrived while the app was killed.
checkCallAcceptAction()	Checks if the user accepted a call from the lock screen.
storeRegisteredNumber(userNumber)	Persists the registered number for kill-state recovery.

CallProgressCallback

Implement this interface in your widget or state management class:

```
abstract class CallProgressCallback {  
    // Fires on every state change (mute, hold, speaker, active status)  
    void onCallStatusChanged(CallStatus status);  
  
    // Fires when a call successfully starts  
    void onCallStarted(String callType, String did, String registeredNumber);  
  
    // Fires when a call ends (locally or remotely)  
    void onCallEnded(String callType, String did);  
  
    // Fires when an error occurs  
    void onCallError(String error, String? callType);  
  
    // Fires when the WebSocket connects or disconnects  
    void onWebSocketStatusChanged(bool isConnected);  
  
    // Fires when the remote party declines the call  
    void onCallDeclined(String callType, String callFrom);  
}
```

CallStatus

An immutable snapshot of call state. Supports `copyWith`, `equality`, and `toString`.

```
class CallStatus {  
    final bool    isCallOngoing;    // true = call in progress  
    final bool    isMuted;          // true = mic is muted  
    final bool    isOnHold;         // true = call is on hold  
    final bool    isSpeakerOn;      // true = speaker is on  
    final String? callType;         // 'inbound' | 'outbound' | 'app_to_app' | null  
    final String? currentDID;       // DID used in this call  
    final String? registeredNumber; // registered number used  
}
```

OneCxiError

Thrown by SDK methods on failure. Catch it with `on OneCxiError`:

Error	When it is thrown
<code>notInitialized</code>	Any SDK method called before <code>initialize()</code>
<code>invalidInput</code>	Empty or whitespace-only parameter passed to a call method
<code>callInProgress</code>	Starting a new call while one is already active
<code>noCallInProgress</code>	Calling <code>endCall()</code> when no call is active
<code>permissionDenied</code>	Microphone permission not granted
<code>websocketConnectionFailed</code>	Cannot connect to the Ozonetel server
<code>audioSessionSetupFailed</code>	Device audio session could not be configured
<code>initializationFailed</code>	<code>initialize()</code> failed (bad credentials or server error)
<code>callFailed</code>	Call could not be started
<code>callEndFailed</code>	Call could not be ended cleanly
<code>incomingCallFailed</code>	Incoming call push could not be handled
<code>speakerControlFailed</code>	Speaker toggle failed
<code>holdControlFailed</code>	Hold/resume failed
<code>muteControlFailed</code>	Mute/unmute failed
<code>unknown</code>	Unclassified error

Each error has a `.message` getter with a human-readable description:

```
print(OneCxiError.permissionDenied.message);  
// → "Permission denied"
```

12. Error Handling

Wrap every SDK call in a `try / on OneCxiError` block:

```

Future<void> startCall() async {
  try {
    await _sdk.startInBoundCall(
      did: '90050',
      registeredNumber: '9876543210',
    );
  } on OneCxiError catch (e) {
    switch (e) {
      case OneCxiError.notInitialized:
        // SDK not set up yet - call initialize() first
        await setupSDK();
        break;

      case OneCxiError.permissionDenied:
        // Ask again or show settings prompt
        await _sdk.requestPermissions();
        break;

      case OneCxiError.callInProgress:
        // End the existing call first
        await _sdk.endCall();
        break;

      case OneCxiError.invalidInput:
        // A parameter was empty - check your inputs
        showError('Please fill in all required fields.');
```

```

        break;

      case OneCxiError.webSocketConnectionFailed:
        // Check network connectivity
        showError('No connection. Please check your internet.');
```

```

        break;

      default:
        showError('Something went wrong: ${e.message}');
```

```

    }
  }
}

```

13. App Lifecycle & State Sync

When the user switches away from the app and comes back, sync the call state so your UI reflects the true call status:

```

class _MyScreenState extends State<MyScreen>
  with WidgetsBindingObserver
  implements CallProgressCallback {

```

```

final OneCxiSdk _sdk = OneCxiSdk();

@override
void initState() {
  super.initState();
  WidgetsBinding.instance.addObserver(this);
  _sdk.setCallProgressCallback(this);
}

@override
void dispose() {
  WidgetsBinding.instance.removeObserver(this);
  super.dispose();
}

@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (state == AppLifecycleState.resumed) {
    _syncCallState();
  }
}

Future<void> _syncCallState() async {
  await _sdk.refreshCallStatus();           // re-sync with native layer
  final status = _sdk.getCallStatus();      // read latest snapshot
  setState(() {
    // update your UI from status.*
  });
}

// ... CallProgressCallback methods
}

```

14. API Keys & Secrets Management

Never commit API keys to source control.

Option A — Dart defines (recommended)

Create a local file `secrets.local.json` (add to `.gitignore`):

```

{
  "ONECXI_DEMO_API_KEY": "your_sdk_api_key_here",
  "ONECXI_DEMO_REGISTRATION_API_KEY": "your_registration_key_here"
}

```

Run with:

```
flutter run --dart-define-from-file=secrets.local.json
```

Read in Dart:

```
const apiKey = String.fromEnvironment('ONECXI_DEMO_API_KEY');

await sdk.initialize(
  accountName: 'your_account',
  apiKey: apiKey,
  serverName: 'https://your-server.com',
);
```

Option B — CI/CD environment variables

```
flutter build apk \
  --dart-define=ONECXI_DEMO_API_KEY=$SDK_API_KEY \
  --dart-define=ONECXI_DEMO_REGISTRATION_API_KEY=$REG_API_KEY
```

Checklist

- ☒ `secrets.local.json` is in `.gitignore`
 - ☒ Only `secrets.template.json` (with placeholder values) is committed
 - ☒ API keys read via `String.fromEnvironment(...)` — never hardcoded as plain strings
-

15. Best Practices

Always validate before calling

```
// Good - check permissions before starting a call
final hasPermissions = await _sdk.checkForPermissions();
if (!hasPermissions) {
  await _sdk.requestPermissions();
  return;
}
await _sdk.startInBoundCall(did: did, registeredNumber: number);
```

Guard against calls in progress

```
// Good - check state first
if (_sdk.isCallActive) {
  showSnackBar('A call is already in progress.');
```

```
  return;
}
```

Use explicit booleans, not toggles

```
// [x] Correct
await _sdk.muteCall(true);
```



```
// [ ] Avoid this pattern
final currentlyMuted = _sdk.isMuted;
await _sdk.muteCall(!currentlyMuted); // risky if state is out of sync
```

Instead, use the state from `onCallStatusChanged` and pass the desired state directly:

```
void onCallStatusChanged(CallStatus status) {
  setState(() => _isMuted = status.isMuted);
}
```

```
// In your mute button:
await _sdk.muteCall(!_isMuted);
```

Register the callback early

Register `setCallProgressCallback` in `initState`, not in response to a button press — otherwise you miss events.

Pre-initialize the background service (Android)

Call `await CallBackgroundService().initialize()` before `runApp()` in `main()`. If you do it after, calls may not survive app kill on Android.

Always handle `onCallEnded`

Reset your entire in-call UI in `onCallEnded`. The call can end from the remote side at any time.

```
@override
void onCallEnded(String callType, String did) {
  setState(() {
    _isCallActive = false;
    _isMuted = false;
    _isOnHold = false;
    _isSpeakerOn = false;
    _callDuration = Duration.zero;
  });
}
```

16. Troubleshooting

Call drops when app is killed

Android: Verify `FOREGROUND_SERVICE` and `FOREGROUND_SERVICE_MICROPHONE` permissions are declared and the `OneCxiCallService` is registered in `AndroidManifest.xml`. Call `CallBackgroundService().initialize()` before `runApp()`.

iOS: Verify VoIP background mode is enabled in entitlements, and that PushKit is set up in `AppDelegate`.

Lock-screen call UI not appearing

Android: Verify `USE_FULL_SCREEN_INTENT` permission. On Android 14+, this permission must be explicitly granted by the user in system settings.

iOS: Verify CallKit and VoIP entitlements are set. Test on a real device (simulator does not fully support CallKit).

No audio (one or both sides)

1. Verify the microphone permission is granted: `await _sdk.checkForPermissions()`.
 2. Test on a **real device** — simulators/emulators do not have real microphones.
 3. Check that `registeredNumber` matches what the server expects.
 4. For app-to-app calls: verify both users are registered with **different** numbers. Same number on both sides = no audio.
-

App-to-app call fails or connects to wrong person

Ensure `registeredNumber` is unique per device/user. The server routes audio based on these numbers — duplicate numbers cause self-calling or audio loop issues.

WebSocket connection fails

1. Check that `serverName` passed to `initialize()` is the correct server URL (including `https://`).
 2. Check that the API key is valid.
 3. Verify the device has an active internet connection.
 4. If on Android, confirm `INTERNET` permission is declared.
-

MissingPluginException in tests

This is expected in unit tests. Platform plugins (CallKit, FCM, notifications) have no implementation in the test environment. The SDK catches these internally — they do not affect app behaviour on real devices. All 145 unit tests pass despite these warnings.

17. Testing Guide

Unit tests (no device needed)

The SDK ships with 145 unit tests covering all models and SDK logic:

```
# Run all unit tests
flutter test
```

```
# Run only model tests
flutter test test/models/

# Run SDK integration tests
flutter test test/sdk_test.dart

# Verbose output
flutter test --reporter=expanded
```

What is covered:

Test file	What it tests
test/models/call_status_test.dart	Constructor, copyWith, equality, hashCode, toString
test/models/call_type_test.dart	Enum values, string conversion, round-trip parsing
test/models/incoming_call_data_test.dart	Push payload parsing, UUID generation, toCallKitMap structure
test/models/onecxi_error_test.dart	All 15 error types, messages, PlatformException mapping
test/sdk_test.dart	Initialization, state guards, input validation, credential storage

Manual device testing checklist

Run through this before each release:

- ☐ Inbound call — starts, audio works, ends cleanly
- ☐ Outbound call — starts, audio works, ends cleanly
- ☐ App-to-app call — both sides hear each other
- ☐ Incoming call via push — native call UI appears on lock screen
- ☐ Accept incoming call from lock screen — audio works
- ☐ Decline incoming call — call ends on both sides
- ☐ Mute — other side cannot hear you, you can still hear them
- ☐ Hold — audio pauses on both sides
- ☐ Speaker — audio routes to loudspeaker
- ☐ DTMF — tones are received by the server
- ☐ Kill the app during a call — call continues on both iOS and Android
- ☐ Reopen the app during an ongoing call — UI reflects the live call state
- ☐ Poor network — call degrades gracefully, no crash

18. Support

Resource	Link
Package (pub.dev)	pub.dev/packages/onecxi_flutter_sdk
Changelog	CHANGELOG.md in the repository

Resource	Link
Example app	example/ directory
License	MIT

For integration support or to report issues, contact the Ozonetel developer team.

Document version: 1.1.0 — aligned with SDK 1.1.0