ZebraVoiceConnect SDK Integration

Version: 1.2.0
Date: March 5, 2026
Target SDK: Android API 30-35

Table of Contents

Overview

The **ZebraVoiceConnect SDK** provides wake word detection capabilities for third-party Android applications running on Zebra devices. The SDK offers a streamlined interface with only two core APIs for maximum simplicity and security.

Key Features

☑ **Wake Word Detection** - Real-time wake word detection with confidence levels and keyphrase identification
☑ **Audio Streaming** - Live audio data streaming in PCM format chunks
☑ **Automatic Permission Management** - Built-in permission checking and validation
☑ **AIDL Service Integration** - Seamless connectivity to ZebraWakeWord service
☑ **Security-First Design** - Minimal API surface with dangerous permission protection

Core APIs

1. **onWakeWordDetected()** - Triggered when wake words are detected with metadata
2. **onRecordingData()** - Streams live audio data chunks in PCM format

---

Prerequisites

Required Components

1. **ZebraWakeWord App** - Must be installed on the device
    - Defines the `BIND_WAKE_WORD_SERVICE` permission

- Provides the wake word detection service

2. **Zebra Device** - Running Android API 30 or higher
   - Minimum SDK: API 30
   - Target SDK: API 35
3. **Runtime Permissions** - Both permissions are dangerous-level:
   - `RECORD_AUDIO` - Standard Android microphone permission
   - `BIND_WAKE_WORD_SERVICE` - Custom Zebra permission for service binding

Dependencies

```
dependencies {
    implementation files('libs/zebravoiceconnect.aar')
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.2'
}
```

Quick Start Guide

1. Add the SDK

Copy `zebravoiceconnect.aar` to your app's `libs/` folder and add the dependency:

```
android {
    compileSdkVersion 33
    defaultConfig {
        minSdk 30
        targetSdk 35
    }
}

dependencies {
    implementation files('libs/zebravoiceconnect.aar')
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
}
```

2. Add Permissions to Manifest

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
android:name="com.zebra.wakeword.permission.BIND_WAKE_WORD_SERVICE" />
```

3. Request Permissions at Runtime

```
import com.zebra.zebravoiceconnect.utils.PermissionHelper;

// In your Activity
if (!PermissionHelper.hasAllPermissions(this)) {
    PermissionHelper.requestPermissions(this);
} else {
    initializeSDK();
}
```

4. Initialize the SDK

```
import com.zebra.zebravoiceconnect.PublicWakeWordSDK;
import com.zebra.zebravoiceconnect.IPublicWakeWordSDK;
import com.zebra.zebravoiceconnect.WakeWordDetectionCallback;

public class MainActivity extends AppCompatActivity {
    private IPublicWakeWordSDK mSDK;

    private void initializeSDK() {
        mSDK = PublicWakeWordSDK.getInstance();
        try {
            mSDK.initialize(this, "MyAppName");
            mSDK.setWakeWordDetectionCallback(new MyWakeWordCallback());
        } catch (Exception e) {
            Log.e("MainActivity", "SDK initialization failed", e);
        }
    }
}
```

5. Implement Callback

```
private class MyWakeWordCallback implements WakeWordDetectionCallback {
    @Override
    public void onWakeWordDetected(float confidence, long timestamp,
                                   String wakeWord, String keyphraseName) {
        Log.i("WakeWord", "Detected: " + wakeWord + " (" + keyphraseName +
              ") with confidence " + confidence);

        // Handle wake word detection
        runOnUiThread(() -> {
            if ("duress word".equals(keyphraseName)) {
                handleEmergency();
            } else {
                handleWakeWord();
            }
```

```
        });
    }

    @Override
    public void onRecordingData(byte[] audioData, int length, boolean isComplete)
{
        // Process audio data chunk (PCM format, 16-bit, 16kHz, mono)
        if (isComplete) {
            Log.d("Audio", "Recording complete");
        } else {
            Log.d("Audio", "Received chunk: " + length + " bytes");
        }
    }
}
```

Detailed Integration

Step 1: Project Setup

1.1 Add SDK Dependency

Place the `zebravoiceconnect.aar` file in your `app/libs/` directory and add to `build.gradle`:

```
android {
    compileSdkVersion 33
    buildToolsVersion "30.0.1"

    defaultConfig {
        minSdk 30
        targetSdk 35
        versionCode 1
        versionName "1.0"
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_11
        targetCompatibility JavaVersion.VERSION_11
    }
}

dependencies {
    implementation files('libs/zebravoiceconnect.aar')

    // Required dependencies
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.2'

    // Optional - for enhanced UI
    testImplementation 'junit:junit:4.13.2'
```

```
    androidTestImplementation 'androidx.test.ext:junit:1.2.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.6.1'
}
```

1.2 Manifest Configuration

Add required permissions to your `AndroidManifest.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yourcompany.yourapp">

    <!-- Required permissions (both dangerous-level) -->
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission
android:name="com.zebra.wakeword.permission.BIND_WAKE_WORD_SERVICE" />

    <application
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Step 2: Permission Management

2.1 Check Prerequisites

Before initializing the SDK, verify that:

1. ZebraWakeWord app is installed
2. All required permissions are granted

```
import com.zebra.zebravoiceconnect.utils.PermissionHelper;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        checkPrerequisites();
    }

    private void checkPrerequisites() {
        // Check if ZebraWakeWord app is installed
        if (!PermissionHelper.isZebraWakeWordInstalled(this)) {
            showError("ZebraWakeWord app is not installed. Please install it
first.");
            return;
        }

        // Check permissions
        if (PermissionHelper.hasAllPermissions(this)) {
            initializeSDK();
        } else {
            requestPermissions();
        }
    }
}
```

2.2 Request Permissions

```
private void requestPermissions() {
    // Show rationale if needed
    String rationale = PermissionHelper.getPermissionRationale();
    showPermissionRationale(rationale, () -> {
        PermissionHelper.requestPermissions(this);
    });
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions,
                                       @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (PermissionHelper.handlePermissionResult(requestCode, permissions,
grantResults)) {
        // All permissions granted
        initializeSDK();
    } else {
        // Some permissions denied
        String[] missing = PermissionHelper.getMissingPermissions(this);
        showPermissionDenied(missing);
    }
}
```

Step 3: SDK Initialization

3.1 Initialize SDK Instance

```
import com.zebra.zebravoiceconnect.PublicWakeWordSDK;
import com.zebra.zebravoiceconnect.IPublicWakeWordSDK;
import com.zebra.zebravoiceconnect.WakeWordDetectionCallback;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";
    private IPublicWakeWordSDK mSDK;

    private void initializeSDK() {
        try {
            // Get SDK instance
            mSDK = PublicWakeWordSDK.getInstance();

            // Initialize with app name
            mSDK.initialize(this, "YourAppName");

            // Set callback for receiving events
            mSDK.setWakeWordDetectionCallback(new MyWakeWordCallback());

            Log.d(TAG, "SDK initialized successfully");
            updateUI("SDK Ready");

        } catch (SecurityException e) {
            Log.e(TAG, "Permission error during initialization", e);
            showError("Missing required permissions");
        } catch (IllegalStateException e) {
            Log.e(TAG, "ZebraWakeWord app not installed", e);
            showError("ZebraWakeWord app is required");
        } catch (Exception e) {
            Log.e(TAG, "Unexpected error during initialization", e);
            showError("SDK initialization failed");
        }
    }
}
```

3.2 Implement Wake Word Callback

```
private class MyWakeWordCallback implements WakeWordDetectionCallback {

    @Override
    public void onWakeWordDetected(float confidence, long timestamp,
                                   String wakeWord, String keyphraseName) {

        Log.i(TAG, String.format(
            "Wake word detected - Word: %s, Keyphrase: %s, Confidence: %.2f, Time:
%d",
            wakeWord, keyphraseName, confidence, timestamp
        ));
```

```
        // Handle on UI thread
        runOnUiThread(() -> {
            handleWakeWordDetection(confidence, timestamp, wakeWord,
keyphraseName);
        });
    }

    @Override
    public void onRecordingData(byte[] audioData, int length, boolean isComplete)
{
        // Audio data is PCM format: 16-bit, 16kHz, mono, no WAV header
        Log.d(TAG, String.format(
            "Audio chunk received - Length: %d bytes, Complete: %s",
            length, isComplete
        ));

        if (isComplete) {
            Log.i(TAG, "Audio recording complete");
            runOnUiThread(() -> updateUI("Recording Complete"));
        } else {
            // Process audio chunk
            processAudioChunk(audioData, length);
        }
    }
}
```

API Reference

IPublicWakeWordSDK Interface

The main SDK interface for third-party applications.

Methods

```
/**
 * Initialize the SDK
 * @param context Application context
 * @param appName Name of the calling application
 * @throws SecurityException if permissions not granted
 * @throws IllegalStateException if ZebraWakeWord app not installed
 */
void initialize(Context context, String appName);

/**
 * Set wake word detection callback
 * @param callback Callback for receiving wake word events and audio data
 */
void setWakeWordDetectionCallback(WakeWordDetectionCallback callback);
```

WakeWordDetectionCallback Interface

Callback interface for receiving SDK events.

Methods

```
/**
 * Called when wake word is detected
 * @param confidence Confidence level (0.0 to 1.0)
 * @param timestamp Detection time (milliseconds since epoch)
 * @param wakeWord The detected wake word text
 * @param keyphraseName Type of keyphrase ("wake word" or "duress word")
 */
void onWakeWordDetected(float confidence, long timestamp,
                        String wakeWord, String keyphraseName);

/**
 * Called when audio data is available
 * @param audioData Audio data chunk (PCM 16-bit, 16kHz, mono)
 * @param length Number of valid bytes in audioData
 * @param isComplete true if last chunk, false if more chunks coming
 */
void onRecordingData(byte[] audioData, int length, boolean isComplete);
```

PermissionHelper Utility

Helper class for managing dangerous permissions.

Methods

```
// Check if all required permissions are granted
public static boolean hasAllPermissions(Context context);

// Check if ZebraWakeWord app is installed
public static boolean isZebraWakeWordInstalled(Context context);

// Get array of missing permissions
public static String[] getMissingPermissions(Context context);

// Request all required permissions
public static boolean requestPermissions(Activity activity);

// Handle permission request results
public static boolean handlePermissionResult(int requestCode,
                                             String[] permissions,
                                             int[] grantResults);

// Get permission rationale message
public static String getPermissionRationale();
```

Constants

```
public static final String PERMISSION_BIND_WAKE_WORD_SERVICE =
    "com.zebra.wakeword.permission.BIND_WAKE_WORD_SERVICE";

public static final String ZEBRA_WAKEWORD_PACKAGE = "com.zebra.wakeword";

public static final int PERMISSION_REQUEST_CODE = 1001;
```

Code Examples

Complete Integration Example

```
public class WakeWordActivity extends AppCompatActivity {
    private static final String TAG = "WakeWordActivity";

    private IPublicWakeWordSDK mSDK;
    private TextView mStatusText;
    private ListView mEventsListView;
    private ArrayAdapter<String> mEventsAdapter;
    private List<String> mEventsList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_wake_word);

        setupUI();
        checkAndInitializeSDK();
    }

    private void setupUI() {
        mStatusText = findViewById(R.id.statusText);
        mEventsListView = findViewById(R.id.eventsListView);

        mEventsList = new ArrayList<>();
        mEventsAdapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1, mEventsList);
        mEventsListView.setAdapter(mEventsAdapter);

        updateStatus("Checking prerequisites...");
    }

    private void checkAndInitializeSDK() {
        // Check ZebraWakeWord app installation
        if (!PermissionHelper.isZebraWakeWordInstalled(this)) {
            showInstallationDialog();
```

```
            return;
        }

        // Check permissions
        if (PermissionHelper.hasAllPermissions(this)) {
            initializeSDK();
        } else {
            showPermissionDialog();
        }
    }

    private void showPermissionDialog() {
        new AlertDialog.Builder(this)
            .setTitle("Permissions Required")
            .setMessage(PermissionHelper.getPermissionRationale())
            .setPositiveButton("Grant", (dialog, which) -> {
                PermissionHelper.requestPermissions(this);
            })
            .setNegativeButton("Cancel", (dialog, which) -> {
                updateStatus("Permissions denied");
            })
            .show();
    }

    private void showInstallationDialog() {
        new AlertDialog.Builder(this)
            .setTitle("ZebraWakeWord App Required")
            .setMessage("The ZebraWakeWord app must be installed to use wake word
detection.")
            .setPositiveButton("OK", (dialog, which) -> finish())
            .show();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        if (PermissionHelper.handlePermissionResult(requestCode, permissions,
grantResults)) {
            initializeSDK();
        } else {
            String[] missing = PermissionHelper.getMissingPermissions(this);
            updateStatus("Missing permissions: " + Arrays.toString(missing));
        }
    }

    private void initializeSDK() {
        try {
            mSDK = PublicWakeWordSDK.getInstance();
            mSDK.initialize(this, "WakeWordDemo");
            mSDK.setWakeWordDetectionCallback(new WakeWordCallback());
```

```
            updateStatus("SDK initialized - Listening for wake words...");
            addEvent("SDK ready for wake word detection");

        } catch (Exception e) {
            Log.e(TAG, "SDK initialization failed", e);
            updateStatus("SDK initialization failed: " + e.getMessage());
        }
    }

    private class WakeWordCallback implements WakeWordDetectionCallback {
        @Override
        public void onWakeWordDetected(float confidence, long timestamp,
                                       String wakeWord, String keyphraseName) {

            String event = String.format("[%s] %s detected: '%s' (%.1f%%
confidence)",
                formatTime(timestamp), keyphraseName, wakeWord, confidence * 100);

            Log.i(TAG, event);

            runOnUiThread(() -> {
                addEvent(event);
                updateStatus("Wake word detected!");

                // Handle different keyphrase types
                if ("duress word".equals(keyphraseName)) {
                    handleDuressWord();
                } else {
                    handleNormalWakeWord();
                }
            });
        }

        @Override
        public void onRecordingData(byte[] audioData, int length, boolean
isComplete) {
            runOnUiThread(() -> {
                if (isComplete) {
                    addEvent("Audio recording complete (" + length + " bytes)");
                    updateStatus("Ready for next wake word...");
                } else {
                    updateStatus("Recording audio... (" + length + " bytes)");
                }
            });
        }
    }

    private void handleNormalWakeWord() {
        // Handle normal wake word detection
        // Start your voice assistant or take appropriate action
    }

    private void handleDuressWord() {
        // Handle duress/emergency word detection
```

```
        // Trigger emergency procedures
        new AlertDialog.Builder(this)
            .setTitle("Emergency Detected")
            .setMessage("Duress word detected. Emergency procedures activated.")
            .setPositiveButton("OK", null)
            .show();
    }

    private void updateStatus(String status) {
        mStatusText.setText(status);
    }

    private void addEvent(String event) {
        mEventsList.add(0, event);  // Add to top
        mEventsAdapter.notifyDataSetChanged();

        // Limit list size
        if (mEventsList.size() > 50) {
            mEventsList.remove(mEventsList.size() - 1);
        }
    }

    private String formatTime(long timestamp) {
        return android.text.format.DateFormat.format("HH:mm:ss",
timestamp).toString();
    }
}
```

Audio Data Processing Example

```
public class AudioProcessor {
    private static final String TAG = "AudioProcessor";
    private static final int SAMPLE_RATE = 16000;
    private static final int CHANNELS = 1;
    private static final int BITS_PER_SAMPLE = 16;

    private ByteArrayOutputStream mAudioBuffer;
    private boolean mRecording;

    public AudioProcessor() {
        mAudioBuffer = new ByteArrayOutputStream();
        mRecording = false;
    }

    public void startRecording() {
        mAudioBuffer.reset();
        mRecording = true;
        Log.d(TAG, "Started audio recording");
    }

    public void processAudioChunk(byte[] audioData, int length, boolean
```

```
isComplete) {
        if (!mRecording) return;

        try {
            mAudioBuffer.write(audioData, 0, length);
            Log.d(TAG, "Processed audio chunk: " + length + " bytes");

            if (isComplete) {
                finishRecording();
            }
        } catch (IOException e) {
            Log.e(TAG, "Error processing audio chunk", e);
        }
    }

    private void finishRecording() {
        mRecording = false;
        byte[] completeAudio = mAudioBuffer.toByteArray();

        Log.i(TAG, "Recording complete. Total size: " + completeAudio.length + "
bytes");

        // Save to file
        saveAudioToFile(completeAudio);

        // Process audio (e.g., send to speech recognition)
        processCompleteAudio(completeAudio);
    }

    private void saveAudioToFile(byte[] audioData) {
        try {
            File audioFile = new File(getExternalFilesDir(null),
                "recording_" + System.currentTimeMillis() + ".pcm");

            try (FileOutputStream fos = new FileOutputStream(audioFile)) {
                fos.write(audioData);
                Log.i(TAG, "Audio saved to: " + audioFile.getAbsolutePath());
            }
        } catch (IOException e) {
            Log.e(TAG, "Error saving audio file", e);
        }
    }

    private void processCompleteAudio(byte[] audioData) {
        // Convert PCM to desired format or send to speech recognition service
        // Note: Audio is already in PCM format (16-bit, 16kHz, mono)

        // Example: Calculate audio duration
        int sampleCount = audioData.length / (BITS_PER_SAMPLE / 8);
        float durationSeconds = (float) sampleCount / SAMPLE_RATE;

        Log.i(TAG, "Audio duration: " + durationSeconds + " seconds");
    }
}
```

Permissions

Required Permissions

Both permissions are **dangerous-level** and must be requested at runtime:

1. RECORD_AUDIO

- **Type:** Standard Android permission
- **Purpose:** Microphone access for wake word detection and audio recording
- **Level:** Dangerous
- **Group:** Microphone

2. BIND_WAKE_WORD_SERVICE

- **Type:** Custom Zebra permission
- **Purpose:** Binding to ZebraWakeWord service
- **Level:** Dangerous
- **Defined by:** ZebraWakeWord app (`com.zebra.wakeword`)

Permission Flow

```
graph TB
    A[App Launch] --> B{ZebraWakeWord
Installed?}
    B -->|No| C[Show Install Dialog]
    B -->|Yes| D{Permissions
Granted?}
    D -->|No| E[Request Permissions]
    D -->|Yes| F[Initialize SDK]
    E --> G{User Response}
    G -->|Granted| F
    G -->|Denied| H[Show Error]
    F --> I[SDK Ready]
    C --> J[Exit/Retry]
    H --> K[Manual Settings]
```

Best Practices

1. **Check Prerequisites Early**

```
if (!PermissionHelper.isZebraWakeWordInstalled(context)) {
    // Handle missing app
}
```

2. **Explain Permission Need**

```
// Show rationale before requesting
String rationale = PermissionHelper.getPermissionRationale();
showUserExplanation(rationale);
```

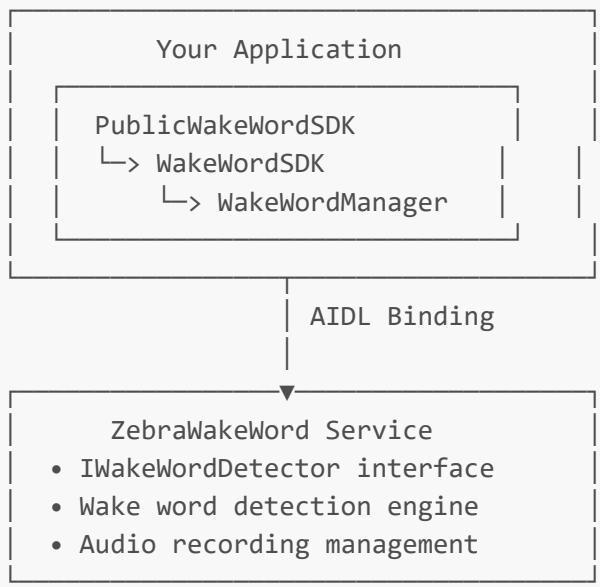
3. **Handle Graceful Degradation**

```
if (!PermissionHelper.hasAllPermissions(context)) {
    // Provide alternative functionality or guide to settings
}
```

Architecture

AIDL Service Communication

The SDK uses AIDL (Android Interface Definition Language) for communication with the ZebraWakeWord service:



Data Flow

1. **Initialization Phase**
   - App requests permissions
   - SDK binds to ZebraWakeWord service via AIDL
   - Service registers callbacks
2. **Detection Phase**

- Service monitors microphone
- On wake word detection:
  - `onWakeWordDetected()` callback triggered
  - Audio recording starts
  - `onRecordingData()` callbacks stream audio chunks

3. **Audio Streaming**
   - Audio data streamed in real-time chunks
   - Format: PCM 16-bit, 16kHz, mono
   - Final chunk marked with `isComplete = true`

Thread Safety

- All callbacks are executed on background threads
- Use `runOnUiThread()` for UI updates:

```
@Override
public void onWakeWordDetected(...) {
    runOnUiThread(() -> {
        // Update UI here
    });
}
```

Troubleshooting

Common Issues

1. SDK Initialization Fails

Error: `SecurityException` during initialize()

Causes:

- Missing permissions
- ZebraWakeWord app not installed

Solutions:

```
try {
    mSDK.initialize(context, appName);
} catch (SecurityException e) {
    // Check permissions
    if (!PermissionHelper.hasAllPermissions(context)) {
        PermissionHelper.requestPermissions(activity);
    }
} catch (IllegalStateException e) {
    // Check app installation
```

```
    if (!PermissionHelper.isZebraWakeWordInstalled(context)) {
        // Guide user to install ZebraWakeWord app
    }
}
```

2. No Wake Word Callbacks

Symptoms: `onWakeWordDetected()` never called

Causes:

- Callback not set
- Permissions revoked after initialization
- Service connection lost

Solutions:

```
// Verify callback is set
if (mSDK != null) {
    mSDK.setWakeWordDetectionCallback(callback);
}

// Check permissions periodically
if (!PermissionHelper.hasAllPermissions(context)) {
    // Re-request permissions
}
```

3. Audio Data Issues

Symptoms: Corrupted or empty audio data

Causes:

- Incorrect buffer handling
- Missing chunks

Solutions:

```
@Override
public void onRecordingData(byte[] audioData, int length, boolean isComplete) {
    // Use length parameter, not audioData.length
    byte[] validData = Arrays.copyOf(audioData, length);

    // Buffer chunks until complete
    if (isComplete) {
        processCompleteAudio();
    }
}
```

4. Permission Denied Errors

Error: Custom permission not available

Cause: ZebraWakeWord app defines the custom permission

Solution:

```
// Always check app installation first
if (!PermissionHelper.isZebraWakeWordInstalled(context)) {
    // Install ZebraWakeWord app first
    showInstallDialog();
    return;
}
```

Debug Tips

1. Enable Detailed Logging

```
private static final String TAG = "WakeWordDebug";

// Log all callback events
@Override
public void onWakeWordDetected(float confidence, long timestamp,
                               String wakeWord, String keyphraseName) {
    Log.d(TAG, String.format("Wake word: %s, keyphrase: %s, confidence: %.3f,
time: %d",
        wakeWord, keyphraseName, confidence, timestamp));
}

@Override
public void onRecordingData(byte[] audioData, int length, boolean isComplete) {
    Log.d(TAG, String.format("Audio chunk: %d bytes, complete: %s", length,
isComplete));
}
```

2. Check Service Connection

```
// Verify ZebraWakeWord service is running
PackageManager pm = getPackageManager();
Intent serviceIntent = new Intent();
serviceIntent.setPackage("com.zebra.wakeword");
List<ResolveInfo> services = pm.queryIntentServices(serviceIntent, 0);

if (services.isEmpty()) {
```

```
        Log.e(TAG, "ZebraWakeWord service not found");
    }
```

3. Monitor Permission Status

```
// Create a permission monitor
private void startPermissionMonitoring() {
    Handler handler = new Handler();
    Runnable permissionChecker = new Runnable() {
        @Override
        public void run() {
            if (!PermissionHelper.hasAllPermissions(context)) {
                Log.w(TAG, "Permissions lost during runtime");
                // Handle permission loss
            }
            handler.postDelayed(this, 5000); // Check every 5 seconds
        }
    };
    handler.post(permissionChecker);
}
```

Error Codes

Error	Description	Solution
`SecurityException`	Missing permissions	Request runtime permissions
`IllegalStateException`	ZebraWakeWord not installed	Install required app
`ServiceConnectionException`	AIDL binding failed	Check service availability
`CallbackException`	Callback registration failed	Verify callback is not null

Best Practices

1. Permission Management

```
// Always check prerequisites in correct order
private boolean checkPrerequisites() {
    // 1. Check app installation first
    if (!PermissionHelper.isZebraWakeWordInstalled(this)) {
        showInstallationGuide();
        return false;
    }

    // 2. Then check permissions
    if (!PermissionHelper.hasAllPermissions(this)) {
        requestPermissions();
```

```
            return false;
        }

        return true;
    }
```

2. Lifecycle Management

```
public class WakeWordActivity extends AppCompatActivity {
    private IPublicWakeWordSDK mSDK;

    @Override
    protected void onResume() {
        super.onResume();

        // Re-check permissions on resume
        if (mSDK != null && !PermissionHelper.hasAllPermissions(this)) {
            mSDK.setWakeWordDetectionCallback(null);
            requestPermissions();
        }
    }

    @Override
    protected void onDestroy() {
        // Clean up callback to prevent memory leaks
        if (mSDK != null) {
            mSDK.setWakeWordDetectionCallback(null);
        }
        super.onDestroy();
    }
}
```

3. Error Handling

```
// Implement comprehensive error handling
private void initializeSDKSafely() {
    try {
        mSDK = PublicWakeWordSDK.getInstance();
        mSDK.initialize(this, getPackageName());
        mSDK.setWakeWordDetectionCallback(new SafeWakeWordCallback());

    } catch (SecurityException e) {
        handleSecurityError(e);
    } catch (IllegalStateException e) {
        handleStateError(e);
    } catch (Exception e) {
        handleUnexpectedError(e);
    }
}
```

```
private class SafeWakeWordCallback implements WakeWordDetectionCallback {
    @Override
    public void onWakeWordDetected(float confidence, long timestamp,
                                   String wakeWord, String keyphraseName) {
        try {
            // Validate parameters
            if (wakeWord == null || keyphraseName == null) {
                Log.w(TAG, "Invalid wake word data received");
                return;
            }

            // Process on UI thread
            runOnUiThread(() -> handleWakeWordSafely(confidence, timestamp,
wakeWord, keyphraseName));

        } catch (Exception e) {
            Log.e(TAG, "Error in wake word callback", e);
        }
    }

    @Override
    public void onRecordingData(byte[] audioData, int length, boolean isComplete)
{
        try {
            if (audioData == null || length < 0) {
                Log.w(TAG, "Invalid audio data received");
                return;
            }

            processAudioDataSafely(audioData, length, isComplete);

        } catch (Exception e) {
            Log.e(TAG, "Error in recording callback", e);
        }
    }
}
```

4. Performance Optimization

```
// Use background threads for audio processing
private ExecutorService mAudioProcessor = Executors.newSingleThreadExecutor();

@Override
public void onRecordingData(byte[] audioData, int length, boolean isComplete) {
    // Don't block the callback thread
    byte[] dataCopy = Arrays.copyOf(audioData, length);

    mAudioProcessor.execute(() -> {
        processAudioData(dataCopy, isComplete);
    });
```

```
}

@Override
protected void onDestroy() {
    // Clean up executor
    if (mAudioProcessor != null) {
        mAudioProcessor.shutdown();
    }
    super.onDestroy();
}
```

5. User Experience

```
// Provide clear feedback to users
private void updateUserInterface(String status) {
    runOnUiThread(() -> {
        // Show status in UI
        mStatusIndicator.setText(status);

        // Update visual indicators
        switch (status) {
            case "SDK Ready":
                mStatusIcon.setImageResource(R.drawable.ic_ready);
                mStatusIcon.setColorFilter(Color.GREEN);
                break;
            case "Listening":
                mStatusIcon.setImageResource(R.drawable.ic_mic);
                mStatusIcon.setColorFilter(Color.BLUE);
                break;
            case "Error":
                mStatusIcon.setImageResource(R.drawable.ic_error);
                mStatusIcon.setColorFilter(Color.RED);
                break;
        }
    });
}
```

Support

For technical support and additional documentation:

- **SDK Version:** 1.2.0
- **Minimum Android:** API 30
- **Target Android:** API 35
