

Connection SDK API Documentation

Document Revision History

No.	Version	Changes	Author	Reviewer	Date
1	v1.0.0	Initial version	Qiu Xian	Lin Dechun	2026.04.11
2	v1.0.1	Add UXErrorCode error code description, PrinterError gains errorCode field	Qiu Xian	Lin Dechun	2026.08.04

1. Getting Started

1.1 Project Configuration

Add the following dependencies in your project's `build.gradle`:

```
implementation files('libs/UxPrint-sdk-v1.xx.xx.aar')
implementation files('libs/lzo-sdk-v1.xx.xx.aar')
```

1.2 Permission Configuration

Add the appropriate permissions in `AndroidManifest.xml` based on the connection method used:

Bluetooth Connection Permissions

```
<!-- Basic Bluetooth permissions -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

<!-- Android 12+ Bluetooth permissions -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
```

WiFi Connection Permissions

```
<!-- WiFi network permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

USB Connection Permissions

```
<!-- USB permission -->
<uses-feature android:name="android.hardware.usb.host" />
```

USB device filter configuration (optional):

```
<activity android:name=".UsbPermissionActivity">
    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
    />
    </intent-filter>
    <meta-data
        android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
        android:resource="@xml/device_filter" />
</activity>
```

1.3 ProGuard Configuration

Add the following rules in `proguard-rules.pro`:

```
-keep class com.printer.sdk.** { *; }
-keep interface com.printer.sdk.** { *; }
```

1.4 Initialization

Initialize the SDK in your Application class:

```
class MyApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        // Basic initialization
        PrinterSDK.initialize(this)

        // Or initialize with logging enabled
        PrinterSDK.initialize(
            applicationContext = this,
            enableIoLog = true,
            ioLogConfig = PrinterIoLogConfig(
                enableReadLog = true,
                enableWriteLog = true,
                logAsHexString = true
            )
        )
    }
}
```

Log File Path:

- File name: `SDK_log.txt`
 - Primary path:
`/storage/emulated/0/Android/data/<package_name>/files/Documents/SDK_log.txt`
 - Fallback path: `/data/data/<package_name>/files/SDK_log.txt`
-

2. API Reference

Connection SDK API Documentation

Document Revision History

1. Getting Started
 - 1.1 Project Configuration
 - 1.2 Permission Configuration
 - Bluetooth Connection Permissions
 - WiFi Connection Permissions
 - USB Connection Permissions
 - 1.3 ProGuard Configuration
 - 1.4 Initialization
2. API Reference
3. SDK Initialization
 - 3.1 initialize
 - 3.2 PrinterIoLogConfig
4. Connection Parameters
 - 4.1 ConnectionParams.WiFi
 - 4.2 ConnectionParams.Bluetooth
 - 4.3 ConnectionParams.USB
5. Connection Interface (PrinterConnection)
 - 5.1 connect
 - 5.2 disconnect
 - 5.3 isConnected
 - 5.4 write
 - 5.5 read
6. Bluetooth Connection Class (BluetoothConnection)
7. WiFi Connection Class (WiFiConnection)
8. USB Connection Class (USBConnection)
9. Appendix
 - 9.1 Error Types
 - 9.2 Return Value Types

3. SDK Initialization

3.1 initialize

Description: Initialize the SDK. Must be called in Application's onCreate method.

Method Signature:

```
fun initialize(  
    applicationContext: Context,  
    enableIoLog: Boolean = false,  
    ioLogConfig: PrinterIoLogConfig = PrinterIoLogConfig()  
)
```

Parameters:

Parameter	Type	Required	Default	Description
applicationContext	Context	Yes	-	Application Context
enableIoLog	Boolean	No	false	Enable IO logging
ioLogConfig	PrinterIoLogConfig	No	PrinterIoLogConfig()	IO log configuration

Example Code:

```
class MyApplication : Application() {  
    override fun onCreate() {  
        super.onCreate()  
        // Basic initialization  
        PrinterSDK.initialize(this)  
  
        // Or initialize with logging enabled  
        PrinterSDK.initialize(  
            applicationContext = this,  
            enableIoLog = true,  
            ioLogConfig = PrinterIoLogConfig(  
                enableReadLog = true,  
                enableWriteLog = true,  
                logAsHexString = true  
            )  
        )  
    }  
}
```

3.2 PrinterIoLogConfig

Description: IO log configuration class.

Properties:

Property	Type	Required	Default	Description
enableReadLog	Boolean	No	true	Enable read logging
enableWriteLog	Boolean	No	true	Enable write logging
logAsHexString	Boolean	No	true	Log as hexadecimal string format

Log File Path:

- File name: `SDK_log.txt`
- Primary path:
`/storage/emulated/0/Android/data/<package_name>/files/Documents/SDK_log.txt`
- Fallback path (when external storage is unavailable):
`/data/data/<package_name>/files/SDK_log.txt`

Example Code:

```
val config = PrinterIoLogConfig(
    enableReadLog = true,
    enableWriteLog = true,
    logAsHexString = true
)
PrinterSDK.initialize(this, enableIoLog = true, ioLogConfig = config)
```

4. Connection Parameters

4.1 ConnectionParams.WiFi

Description: WiFi connection parameters.

Properties:

Property	Type	Required	Default	Description
ip	String	Yes	-	Printer IP address
port	Int	No	9100	Port number
timeout	Long	No	5000	Connection timeout (milliseconds)

Example Code:

```
val params = ConnectionParams.WiFi(
    ip = "192.168.1.100",
    port = 9100,
    timeout = 5000
)
```

4.2 ConnectionParams.Bluetooth

Description: Bluetooth connection parameters.

Properties:

Property	Type	Required	Description
address	String	Yes	Bluetooth device address (format: XX:XX:XX:XX:XX:XX)

Example Code:

```
val params = ConnectionParams.Bluetooth(  
    address = "00:11:22:33:44:55"  
)
```

4.3 ConnectionParams.USB

Description: USB connection parameters.

Properties:

Property	Type	Required	Description
device	UsbDevice	Yes	USB device object

Example Code:

```
val usbManager = getSystemService(Context.USB_SERVICE) as UsbManager  
val deviceList = usbManager.deviceList  
val device = deviceList.values.first()  
  
val params = ConnectionParams.USB(device = device)
```

5. Connection Interface (PrinterConnection)

5.1 connect

Description: Connect to the printer.

Method Signature:

```
suspend fun connect(params: ConnectionParams): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
params	ConnectionParams	Yes	Connection parameters (WiFi/Bluetooth/USB)

Return Value: `Result<Unit>` Connection result

Example Code:

```
// WiFi connection
val connection = WiFiConnection()
val result = connection.connect(ConnectionParams.WiFi("192.168.1.100"))
when (result) {
    is Result.Success -> Log.d("Connection", "Connected")
    is Result.Failure -> Log.e("Connection", "Failed:
    ${result.error.message}")
}
```

5.2 disconnect

Description: Disconnect from the printer.

Method Signature:

```
suspend fun disconnect(): Result<Unit>
```

Return Value: `Result<Unit>` Disconnect result

Example Code:

```
connection.disconnect()
```

5.3 isConnected

Description: Check if connected.

Method Signature:

```
val isConnected: Boolean
```

Return Value: Whether connected

Example Code:

```
if (connection.isConnected) {
    // Connected
}
```

5.4 write

Description: Write data to the printer.

Method Signature:

```
suspend fun write(data: ByteArray): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
data	ByteArray	Yes	Data to write

Return Value: `Result<Unit>` Write result

Example Code:

```
val data = byteArrayOf(0x1B, 0x40) // ESC @ initialize printer
connection.write(data)
```

5.5 read

Description: Read data from the printer.

Method Signature:

```
suspend fun read(timeout: Long = 2000): Result<ByteArray>
```

Parameters:

Parameter	Type	Required	Default	Description
timeout	Long	No	2000	Timeout (milliseconds)

Return Value: `Result<ByteArray>` Read data

Example Code:

```

val result = connection.read(timeout = 3000)
when (result) {
    is Result.Success -> {
        val data = result.value
        // Process data
    }
    is Result.Failure -> Log.e("Read", "Failed: ${result.error.message}")
}

```

6. Bluetooth Connection Class (BluetoothConnection)

Description: Bluetooth connection implementation class.

Permission Configuration:

```

<!-- Basic Bluetooth permissions -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

<!-- Android 12+ Bluetooth permissions -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />

```

Example Code:

```

// Create Bluetooth connection
val connection = BluetoothConnection()

// Connect to printer
lifecycleScope.launch {
    val result = connection.connect(
        ConnectionParams.Bluetooth(address = "00:11:22:33:44:55")
    )
    when (result) {
        is Result.Success -> {
            // Connected successfully, send data
            connection.write(printData)
        }
        is Result.Failure -> {
            Log.e("Bluetooth", "Connection failed: ${result.error.message}")
        }
    }
}
}

```

7. WiFi Connection Class (WiFiConnection)

Description: WiFi connection implementation class.

Permission Configuration:

```
<!-- WiFi network permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Example Code:

```
// Create WiFi connection
val connection = WiFiConnection()

// Connect to printer
lifecycleScope.launch {
    val result = connection.connect(
        ConnectionParams.WiFi(
            ip = "192.168.1.100",
            port = 9100,
            timeout = 5000
        )
    )
    when (result) {
        is Result.Success -> {
            // Connected successfully, send data
            connection.write(printData)
        }
        is Result.Failure -> {
            Log.e("WiFi", "Connection failed: ${result.error.message}")
        }
    }
}
```

8. USB Connection Class (USBConnection)

Description: USB connection implementation class.

Permission Configuration:

```
<!-- USB permission -->
<uses-feature android:name="android.hardware.usb.host" />
```

Example Code:

```

// Get USB device
val usbManager = getSystemService(Context.USB_SERVICE) as UsbManager
val deviceList = usbManager.deviceList

if (deviceList.isNotEmpty()) {
    val device = deviceList.values.first()

    // Request USB permission
    val permissionIntent = PendingIntent.getBroadcast(
        this, 0, Intent(ACTION_USB_PERMISSION), PendingIntent.FLAG_IMMUTABLE
    )
    usbManager.requestPermission(device, permissionIntent)

    // Create USB connection
    val connection = USBConnection(this)

    // Connect to printer
    lifecycleScope.launch {
        val result = connection.connect(ConnectionParams.USB(device))
        when (result) {
            is Result.Success -> {
                // Connected successfully, send data
                connection.write(printData)
            }
            is Result.Failure -> {
                Log.e("USB", "Connection failed: ${result.error.message}")
            }
        }
    }
}
}

```

9. Appendix

9.1 Error Types

Error Type	Description	Possible Scenarios
ConnectionError	Connection error	Printer connection failure, network issues, Bluetooth pairing failure
CommunicationError	Communication error	Data transmission failure, printer response error, communication timeout
InvalidParameterError	Invalid parameter error	Parameter value out of range, incorrect parameter format
TimeoutError	Timeout error	Operation execution timeout, printer no response
NotConnectedError	Not connected error	Attempting to perform operations in unconnected state
UnsupportedOperationError	Unsupported operation error	Printer model does not support the feature, protocol does not support the command
UnknownError	Unknown error	Other unclassified error situations

Error Code Note: When an API returns `Result.Failure`, the `PrinterError` carries an `errorCode` field (corresponds to `UXPErrorCode.value`). Each error type defaults to its fallback error code below; SDK call sites pass more specific error codes. See the UXPErrorCode Error Code Documentation (Guide folder) for the full `UXPErrorCode` table.

Fallback Code	Value	Description
ConnectionError	1001	Connection error
CommunicationError	1002	Communication error
InvalidParameterError	1003	Invalid parameter
TimeoutError	1004	Timeout
NotConnectedError	1005	Not connected
UnsupportedOperationError	1006	Unsupported operation
UnknownError	1007	Unknown error

Error Handling Example:

```
lifecycleScope.launch {
    val result = connection.connect(ConnectionParams.USB(device))
    when (result) {
        is Result.Success -> {
            // Connected successfully, send data
            connection.write(printData)
        }
        is Result.Failure -> {
            when (result.error) {
                is PrinterError.ConnectionError -> {
```

```
Log.e("USB", "Connection error: ${result.error.message}")
// Try to reconnect
}
is PrinterError.CommunicationError -> {
    Log.e("USB", "Communication error:
${result.error.message}")
    // Check printer status
}
is PrinterError.InvalidParameterError -> {
    Log.e("USB", "Invalid parameter: ${result.error.message}")
    // Correct parameter values
}
is PrinterError.TimeoutError -> {
    Log.e("USB", "Timeout error: ${result.error.message}")
    // Increase timeout or check printer status
}
is PrinterError.NotConnectedError -> {
    Log.e("USB", "Not connected: ${result.error.message}")
    // Connect to printer first
}
is PrinterError.UnsupportedOperationError -> {
    Log.e("USB", "Unsupported operation:
${result.error.message}")
    // Use alternative solution
}
is PrinterError.UnknownError -> {
    Log.e("USB", "Unknown error: ${result.error.message}")
    // Log error and try to recover
}
}
}
}
```

9.2 Return Value Types

Type	Description
Result.Success	Success, contains return value
Result.Failure	Failure, contains error information