

ZPL 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 UXPErrrorCode 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

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

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

```
// Create connection
val connection = BluetoothConnection()
connection.connect(ConnectionParams.Bluetooth(address = "00:11:22:33:44:55"))

// Create ZplPrinter instance
val printer = ZplPrinter(connection)

// Start label format
printer.start()
```

2. API Reference

ZPL SDK API Documentation

Document Revision History

1. Getting Started
 - 1.1 Project Configuration
 - 1.2 Permission Configuration
 - 1.3 ProGuard Configuration
 - 1.4 Initialization
2. API Reference
3. start
4. getStatus
5. clear
6. printText
7. printBarcode
8. printQRcode
9. printImage
10. setPrintWidth
11. setLabelLength
12. setPrintSpeed
13. setPrintDarkness
14. setLabelHome
15. setLabelShift
16. setFieldOrigin
17. setFieldOrientation
18. fieldData
19. fieldSeparator
20. comment
21. getSerialNumber
22. getFirmwareVersion
23. selfTest
24. setTaskId
25. print
26. cutPaper
27. printQuantity
28. cleanMissedHitsTask
29. writeRFID

- 30. readRFID
- 31. setLanguageEncode
- 32. setPrinterModel
- 33. printPRNTemplate
- 34. drawBox
- 35. drawCircle
- 36. drawDiagonalLine
- 37. Appendix
 - 37.1 Font Enum (ZplFont)
 - 37.2 Rotation Enum (Rotation)
 - 37.3 Barcode Type Enum (BarcodeType)
 - 37.4 QR Model Enum (QRModel)
 - 37.5 Dithering Algorithm Enum (DitheringAlgorithm)
 - 37.6 Error Types

3. start

Description: Start label format. **Prerequisite for label printing, must be called before printing.**

Method Signature:

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

Return Value: `Result<Unit>` Operation result

Example Code:

```
printer.start()
```

4. getStatus

Description: Get printer status.

Method Signature:

```
suspend fun getStatus(): Result<PrinterStatus>
```

Return Value: `Result<PrinterStatus>` Printer status

PrinterStatus Properties:

Property	Type	Description
isOnline	Boolean	Printer online status
isPaperOut	Boolean	Paper out status
isPaperNearEnd	Boolean	Paper near end status
isCoverOpen	Boolean	Cover open status
isError	Boolean	Error status
isCutterError	Boolean	Cutter error status
isVoltageError	Boolean	Voltage error status
isOverheating	Boolean	Overheating status
isPrinting	Boolean	Printing status
isRibbonOut	Boolean	Ribbon out status (ZPL protocol)
temperature	Int?	Temperature (Celsius), may be null
batteryLevel	Int?	Battery level (0-100%), may be null

Status Check Example:

```

val result = printer.getStatus()
when (result) {
    is Result.Success -> {
        val status = result.value
        when {
            status.isPaperOut -> Log.w("Printer", "No paper!")
            status.isCoverOpen -> Log.w("Printer", "Cover is open!")
            status.isOverheating -> Log.w("Printer", "Printer is
overheating!")
            status.isError -> Log.e("Printer", "Printer error!")
            status.isOnline -> Log.d("Printer", "Printer is online")
        }
    }
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}

```

5. clear

Description: Clear buffer.

Method Signature:

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

Return Value: `Result<Unit>` Operation result

Example Code:

```
printer.clear()
```

6. printText

Description: Print text.

Method Signature:

```
suspend fun printText(  
    x: Int,  
    y: Int,  
    type: Int,  
    orientation: Char,  
    textWidth: Int,  
    textHeight: Int,  
    data: String  
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
x	Int	Yes	X coordinate (dots)
y	Int	Yes	Y coordinate (dots)
type	Int	Yes	Font type (ZPL built-in font number)
orientation	Char	Yes	Orientation (N=Normal, R=Rotate 90°, I=Invert 180°, B=Bottom-up 270°)
textWidth	Int	Yes	Text width (dots)
textHeight	Int	Yes	Text height (dots)
data	String	Yes	Text to print

Example Code:

```
printer.printText(  
    x = 50,  
    y = 30,  
    type = 0,  
    orientation = 'N',  
    textWidth = 30,  
    textHeight = 30,  
    data = "ZPL Label"  
)
```

7. printBarcode

Description: Print barcode.

Method Signature:

```
suspend fun printBarcode(  
    x: Int,  
    y: Int,  
    type: Int,  
    ratio: Int,  
    orientation: Char,  
    height: Int,  
    printInterpretationLine: Boolean,  
    data: String  
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
x	Int	Yes	X coordinate (dots)
y	Int	Yes	Y coordinate (dots)
type	Int	Yes	Barcode type (ZPL barcode number)
ratio	Int	Yes	Ratio (narrow:wide bar ratio)
orientation	Char	Yes	Orientation (N/R/I/B)
height	Int	Yes	Barcode height (dots)
printInterpretationLine	Boolean	Yes	Print interpretation line
data	String	Yes	Barcode data

Example Code:

```
printer.printBarcode(  
    x = 50,  
    y = 80,  
    type = 128,  
    ratio = 3,  
    orientation = 'N',  
    height = 80,  
    printInterpretationLine = true,  
    data = "1234567890"  
)
```

8. printQRcode

Description: Print QR code.

Method Signature:

```
suspend fun printQRcode(  
    x: Int,  
    y: Int,  
    orientation: Char,  
    magnification: Int,  
    size: Int,  
    data: String  
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
x	Int	Yes	X coordinate (dots)
y	Int	Yes	Y coordinate (dots)
orientation	Char	Yes	Orientation (N/R/I/B)
magnification	Int	Yes	Magnification (1-10)
size	Int	Yes	Size (1-10)
data	String	Yes	QR code data

Example Code:

```
printer.printQRcode(
    x = 200,
    y = 150,
    orientation = 'N',
    magnification = 5,
    size = 5,
    data = "https://www.example.com"
)
```

9. printImage

Description: Print image.

Method Signature:

```
// Overload 1: Using byte array
suspend fun printImage(
    imageData: ByteArray,
    width: Int,
    height: Int,
    x: Int = 0,
    y: Int = 0,
    options: ImageOptions = ImageOptions()
): Result<Unit>

// Overload 2: Using Bitmap
suspend fun printImage(
    bitmap: Bitmap,
    x: Int = 0,
    y: Int = 0,
    options: ImageOptions = ImageOptions()
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
imageData	ByteArray	Yes (overload 1)	-	Image byte data
width	Int	Yes (overload 1)	-	Image width (dots)
height	Int	Yes (overload 1)	-	Image height (dots)
bitmap	Bitmap	Yes (overload 2)	-	Image to print
x	Int	No	0	X coordinate (dots)
y	Int	No	0	Y coordinate (dots)
options	ImageOptions	No	ImageOptions()	Image options

ImageOptions Properties:

Property	Type	Default	Description
threshold	Int	-1	Binarization threshold (-1 for auto-calculate)
algorithm	DitheringAlgorithm	DIFFUSION	Dithering algorithm
maxWidth	Int?	null	Maximum width (will auto-scale if exceeded, dots)

Example Code:

```
// Using Bitmap
val bitmap = BitmapFactory.decodeResource(resources, R.drawable.logo)
printer.printImage(
    bitmap = bitmap,
    x = 50,
    y = 200,
    options = ImageOptions(algorithm = DitheringAlgorithm.DIFFUSION)
)
```

10. setPrintWidth

Description: Set print width.

Method Signature:

```
suspend fun setPrintWidth(width: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
width	Int	Yes	Print width (dots)

Example Code:

```
printer.setPrintWidth(832) // 4 inches
```

11. setLabelLength

Description: Set label length.

Method Signature:

```
suspend fun setLabelLength(length: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
length	Int	Yes	Label length (dots)

Example Code:

```
printer.setLabelLength(600) // 3 inches
```

12. setPrintSpeed

Description: Set print speed.

Method Signature:

```
suspend fun setPrintSpeed(speed: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
speed	Int	Yes	Print speed (inches/second)

Example Code:

```
printer.setPrintSpeed(4)
```

13. setPrintDarkness

Description: Set print darkness.

Method Signature:

```
suspend fun setPrintDarkness(darkness: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
darkness	Int	Yes	Darkness value (0-30)

Example Code:

```
printer.setPrintDarkness(15)
```

14. setLabelHome

Description: Set label home.

Method Signature:

```
suspend fun setLabelHome(x: Int, y: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
x	Int	Yes	X coordinate (dots)
y	Int	Yes	Y coordinate (dots)

Example Code:

```
printer.setLabelHome(x = 0, y = 0)
```

15. setLabelShift

Description: Set label shift.

Method Signature:

```
suspend fun setLabelShift(shift: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
shift	Int	Yes	Shift value (dots)

Example Code:

```
printer.setLabelShift(0)
```

16. setFieldOrigin

Description: Set field origin.

Method Signature:

```
suspend fun setFieldOrigin(x: Int, y: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
x	Int	Yes	X coordinate (dots)
y	Int	Yes	Y coordinate (dots)

Example Code:

```
printer.setFieldOrigin(x = 50, y = 30)
```

17. setFieldOrientation

Description: Set field orientation.

Method Signature:

```
suspend fun setFieldOrientation(  
    rotation: Char = 'N',  
    justification: Char = 'L'  
) : Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
rotation	Char	No	'N'	Rotation (N=Normal, R=Rotate 90°, I=Invert 180°, B=Bottom-up 270°)
justification	Char	No	'L'	Justification (L=Left, C=Center, R=Right, J=Justified)

Example Code:

```
printer.setFieldOrientation(rotation = 'N', justification = 'L')
```

18. fieldData

Description: Send field data.

Method Signature:

```
suspend fun fieldData(data: String): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
data	String	Yes	Field data

Example Code:

```
printer.setFieldOrigin(50, 30)  
printer.fieldData("Field Data")
```

19. fieldSeparator

Description: Send field separator.

Method Signature:

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

Return Value: `Result<Unit>` Operation result

Example Code:

```
printer.fieldSeparator()
```

20. comment

Description: Add comment.

Method Signature:

```
suspend fun comment(text: String): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
text	String	Yes	Comment text

Example Code:

```
printer.comment("This is a comment")
```

21. getSerialNumber

Description: Get printer serial number.

Method Signature:

```
suspend fun getSerialNumber(): Result<String>
```

Return Value: `Result<String>` Serial number

Example Code:

```
val result = printer.getSerialNumber()
when (result) {
    is Result.Success -> Log.d("Printer", "SN: ${result.value}")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

22. getFirmwareVersion

Description: Get printer firmware version.

Method Signature:

```
suspend fun getFirmwareVersion(): Result<String>
```

Return Value: `Result<String>` Firmware version

Example Code:

```
val result = printer.getFirmwareVersion()
when (result) {
    is Result.Success -> Log.d("Printer", "Version: ${result.value}")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

23. selfTest

Description: Execute self test.

Method Signature:

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

Return Value: `Result<Unit>` Operation result

Example Code:

```
printer.selfTest()
```

24. setTaskId

Description: Set task ID.

Method Signature:

```
suspend fun setTaskId(id: Int): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
id	Int	Yes	Task ID

Example Code:

```
printer.setTaskId(1)
```

25. print

Description: Execute print. **Prerequisite for label printing, must be called after all print content.**

Method Signature:

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

Return Value: `Result<Unit>` Print result

Example Code:

```
// Start label format
printer.start()
// Add print content
printer.printText(50, 30, 0, 'N', 30, 30, "Label Content")
printer.printQRcode(200, 150, 'N', 5, 5, "QR Data")
// Execute print
printer.print()
```

26. cutPaper

Description: Cut paper.

Method Signature:

```
suspend fun cutPaper(partial: Boolean = false): Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
partial	Boolean	No	false	Partial cut

Return Value: `Result<Unit>` Operation result

Example Code:

```
// Full cut
printer.cutPaper()

// Partial cut
printer.cutPaper(partial = true)
```

27. printQuantity

Description: Set print quantity.

Method Signature:

```
suspend fun printQuantity(
    quantity: Int,
    pauseCount: Int = 0,
    replicateCount: Int = 0,
    overridePause: Char = 'N'
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
quantity	Int	Yes	-	Print quantity
pauseCount	Int	No	0	Pause count
replicateCount	Int	No	0	Replicate count
overridePause	Char	No	'N'	Override pause (Y=Yes, N=No)

Example Code:

```
printer.printQuantity(  
    quantity = 3,  
    pauseCount = 0,  
    replicateCount = 0,  
    overridePause = 'N'  
)  
printer.print()
```

28. cleanMissedHitsTask

Description: Clean missed hits task.

Method Signature:

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

Return Value: `Result<Unit>` Operation result

Example Code:

```
printer.cleanMissedHitsTask()
```

29. writeRFID

Description: Write RFID data.

Method Signature:

```
suspend fun writeRFID(address: Int, memory: Int, data: ByteArray):  
Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
address	Int	Yes	Start address (byte offset)
memory	Int	Yes	Memory area (0=EPC, 1=User Memory, 2=TID)
data	ByteArray	Yes	RFID data

Example Code:

```
val data = "RFID Data".toByteArray()
printer.writeRFID(address = 0, memory = 0, data = data)
```

30. readRFID

Description: Read RFID data.

Method Signature:

```
suspend fun readRFID(address: Int, length: Int, memory: Int):
Result<ByteArray>
```

Parameters:

Parameter	Type	Required	Description
address	Int	Yes	Start address (byte offset)
length	Int	Yes	Read length (bytes)
memory	Int	Yes	Memory area (0=EPC, 1=User Memory, 2=TID)

Return Value: `Result<ByteArray>` RFID data

Example Code:

```
val result = printer.readRFID(address = 0, length = 24, memory = 0)
when (result) {
    is Result.Success -> Log.d("Printer", "RFID: ${String(result.value)}")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

31. setLanguageEncode

Description: Set language encode.

Method Signature:

```
suspend fun setLanguageEncode(languageEncode: String): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
languageEncode	String	Yes	Language encode

Example Code:

```
printer.setLanguageEncode("GBK")
```

32. setPrinterModel

Description: Set printer model.

Method Signature:

```
suspend fun setPrinterModel(model: String): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
model	String	Yes	Printer model

Example Code:

```
printer.setPrinterModel("QL-800")
```

33. printPRNTemplate

Description: Print PRN template.

Method Signature:

```
suspend fun printPRNTemplate(prnContent: String): Result<Unit>
```

Parameters:

Parameter	Type	Required	Description
prnContent	String	Yes	PRN content

Example Code:

```
val prnContent = "^XA^FO50,50^A0N,50,50^FDHello World^FS^XZ"
printer.printPRNTemplate(prnContent)
```

34. drawBox

Description: Draw box.

Method Signature:

```
suspend fun drawBox(
    width: Int,
    height: Int,
    thickness: Int,
    color: Char = 'B',
    rounding: Int = 0
): Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
width	Int	Yes	-	Box width (dots)
height	Int	Yes	-	Box height (dots)
thickness	Int	Yes	-	Line width (dots)
color	Char	No	'B'	Color (B=Black, W=White)
rounding	Int	No	0	Rounding radius (Not supported by ZPL , fixed at 0)

Example Code:

```
printer.drawBox(width = 300, height = 200, thickness = 2, color = 'B',
    rounding = 0)
```

35. drawCircle

Description: Draw circle.

Method Signature:

```
suspend fun drawCircle(  
    diameter: Int,  
    thickness: Int,  
    color: Char = 'B'  
) : Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
diameter	Int	Yes	-	Diameter (dots)
thickness	Int	Yes	-	Line width (dots)
color	Char	No	'B'	Color (B=Black, W=White) (Not supported by ZPL , fixed at 'B')

Example Code:

```
printer.drawCircle(diameter = 100, thickness = 2, color = 'B')
```

36. drawDiagonalLine

Description: Draw diagonal line.

Method Signature:

```
suspend fun drawDiagonalLine(  
    width: Int,  
    height: Int,  
    thickness: Int,  
    color: Char = 'B',  
    orientation: Char = 'R'  
) : Result<Unit>
```

Parameters:

Parameter	Type	Required	Default	Description
width	Int	Yes	-	Width (dots)
height	Int	Yes	-	Height (dots)
thickness	Int	Yes	-	Line width (dots)
color	Char	No	'B'	Color (B=Black, W=White)
orientation	Char	No	'R'	Orientation (R=Right-slant, L=Left-slant)

Example Code:

```
printer.drawDiagonalLine(width = 100, height = 50, thickness = 2, color = 'B',
orientation = 'R')
```

37. Appendix

37.1 Font Enum (ZplFont)

Value	Description
FONT_0	Font 0 (9x5)
FONT_1	Font 1 (11x7)
FONT_2	Font 2 (13x9)
FONT_3	Font 3 (16x11)
FONT_4	Font 4 (21x13)
FONT_5	Font 5 (27x15)
FONT_6	Font 6 (33x17)
FONT_7	Font 7 (39x19)
FONT_A	Font A (9x5)
FONT_B	Font B (11x7)
FONT_C	Font C (18x10)
FONT_D	Font D (21x13)
FONT_E	Font E (27x15)
FONT_F	Font F (33x17)
FONT_G	Font G (60x40)
FONT_H	Font H (21x27)

37.2 Rotation Enum (Rotation)

Value	Description
NORMAL	Normal (0 degrees)
ROTATE_90	Rotate 90 degrees
ROTATE_180	Rotate 180 degrees
ROTATE_270	Rotate 270 degrees

37.3 Barcode Type Enum (BarcodeType)

Value	Description
CODE128	Code 128
CODE39	Code 39
CODE93	Code 93
EAN13	EAN-13
EAN8	EAN-8
UPC_A	UPC-A
UPC_E	UPC-E
CODABAR	Codabar
ITF	ITF
DATA_MATRIX	DataMatrix

37.4 QR Model Enum (QRModel)

Value	Description
MODEL_1	Model 1
MODEL_2	Model 2

37.5 Dithering Algorithm Enum (DitheringAlgorithm)

Value	Description
THRESHOLD	Simple threshold
DIFFUSION	Diffusion dithering
CLUSTER	Cluster dithering

37.6 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:

```
val result = printer.printText(50, 30, 0, 'N', 30, 30, "Hello, World!")
when (result) {
    is Result.Success -> {
        Log.d("Printer", "Print successful")
    }
    is Result.Failure -> {
        when (result.error) {
            is PrinterError.ConnectionError -> {
                Log.e("Printer", "Connection error: ${result.error.message}")
            }
        }
    }
}
```

```
        is PrinterError.CommunicationError -> {
            Log.e("Printer", "Communication error:
${result.error.message}")
        }
        is PrinterError.InvalidParameterError -> {
            Log.e("Printer", "Invalid parameter: ${result.error.message}")
        }
        is PrinterError.TimeoutError -> {
            Log.e("Printer", "Timeout error: ${result.error.message}")
        }
        is PrinterError.NotConnectedError -> {
            Log.e("Printer", "Not connected: ${result.error.message}")
        }
        is PrinterError.UnsupportedOperationError -> {
            Log.e("Printer", "Unsupported operation:
${result.error.message}")
        }
        is PrinterError.UnknownError -> {
            Log.e("Printer", "Unknown error: ${result.error.message}")
        }
    }
}
}
```