

ZPL SDK 接口文档

文档修改记录

序号	版本号	修改内容	修改者	评审者	修改日期
1	v1.0.0	初始版本	邱贤	林德春	2026.04.11
2	v1.0.1	新增错误码 UXErrorCode 说明, PrinterError 增加 errorCode 字段	邱贤	林德春	2026.08.04

1. 使用说明

1.1 工程配置说明

在项目的 `build.gradle` 中添加依赖:

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

1.2 权限配置

```
<!-- 基础权限 -->
<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 混淆配置

在 `proguard-rules.pro` 中添加:

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

1.4 初始化

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

// 创建 ZplPrinter 实例
val printer = ZplPrinter(connection)

// 开始标签格式
printer.start()
```

2. 接口目录

ZPL SDK 接口文档

文档修改记录

1. 使用说明

1.1 工程配置说明

1.2 权限配置

1.3 混淆配置

1.4 初始化

2. 接口目录

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. 附录
 - 37.1 字体枚举 (ZplFont)
 - 37.2 旋转角度枚举 (Rotation)
 - 37.3 条码类型枚举 (BarcodeType)
 - 37.4 二维码模型枚举 (QRModel)
 - 37.5 抖动算法枚举 (DitheringAlgorithm)
 - 37.6 错误类型说明

3. start

描述：开始标签格式。标签打印的前提条件，必须在打印前调用。

方法签名：

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

返回值： `Result<Unit>` 操作结果

示例代码：

```
printer.start()
```

4. getStatus

描述：获取打印机状态。

方法签名：

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

返回值： `Result<PrinterStatus>` 打印机状态

PrinterStatus 属性详细说明：

属性名	类型	说明
isOnline	Boolean	打印机是否在线
isPaperOut	Boolean	是否缺纸
isPaperNearEnd	Boolean	纸张是否即将用尽
isCoverOpen	Boolean	纸仓盖是否打开
isError	Boolean	是否有错误
isCutterError	Boolean	切刀是否有错误
isVoltageError	Boolean	电压是否有错误
isOverheating	Boolean	打印机是否过热
isPrinting	Boolean	打印机是否正在打印中
isRibbonOut	Boolean	是否缺碳带（ZPL协议）
temperature	Int?	温度（摄氏度），可能为null
batteryLevel	Int?	电池电量（0-100%），可能为null

状态检查示例：

```
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

描述：清除缓冲区。

方法签名：

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

返回值： `Result<Unit>` 操作结果

示例代码：

```
printer.clear()
```

6. printText

描述： 打印文本。

方法签名：

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

参数：

参数名	类型	必填	说明
x	Int	是	X坐标（点）
y	Int	是	Y坐标（点）
type	Int	是	字体类型（ZPL内置字体编号）
orientation	Char	是	方向（N=正常, R=旋转90°, I=反转180°, B=底部朝上270°）
textWidth	Int	是	文本宽度（点）
textHeight	Int	是	文本高度（点）
data	String	是	要打印的文本

示例代码：

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

7. printBarcode

描述：打印条码。

方法签名：

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

参数：

参数名	类型	必填	说明
x	Int	是	X坐标（点）
y	Int	是	Y坐标（点）
type	Int	是	条码类型（ZPL条码编号）
ratio	Int	是	比例（窄条:宽条比值）
orientation	Char	是	方向（N/R/I/B）
height	Int	是	条码高度（点）
printInterpretationLine	Boolean	是	是否打印解释线
data	String	是	条码数据

示例代码：

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

8. printQRcode

描述：打印二维码。

方法签名：

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

参数：

参数名	类型	必填	说明
x	Int	是	X坐标（点）
y	Int	是	Y坐标（点）
orientation	Char	是	方向（N/R/I/B）
magnification	Int	是	放大倍率（1-10）
size	Int	是	尺寸（1-10）
data	String	是	二维码数据

示例代码：

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

9. printImage

描述：打印图片。

方法签名：

```
// 重载1: 使用字节数组  
suspend fun printImage(  
    imageData: ByteArray,  
    width: Int,  
    height: Int,  
    x: Int = 0,  
    y: Int = 0,  
    options: ImageOptions = ImageOptions()  
) : Result<Unit>  
  
// 重载2: 使用Bitmap  
suspend fun printImage(  
    bitmap: Bitmap,  
    x: Int = 0,  
    y: Int = 0,  
    options: ImageOptions = ImageOptions()  
) : Result<Unit>
```

参数：

参数名	类型	必填	默认值	说明
imageData	ByteArray	是（重载1）	-	图片字节数据
width	Int	是（重载1）	-	图片宽度（点）
height	Int	是（重载1）	-	图片高度（点）
bitmap	Bitmap	是（重载2）	-	要打印的图片
x	Int	否	0	X坐标（点）
y	Int	否	0	Y坐标（点）
options	ImageOptions	否	ImageOptions()	图片选项

ImageOptions 属性：

属性名	类型	默认值	说明
threshold	Int	-1	二值化阈值（-1为自动计算）
algorithm	DitheringAlgorithm	DIFFUSION	抖动算法
maxWidth	Int?	null	最大宽度（超过将自动缩放，点）

示例代码：

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

10. setPrintWidth

描述：设置打印宽度。

方法签名：

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

参数：

参数名	类型	必填	说明
width	Int	是	打印宽度（点）

示例代码：

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

11. setLabelLength

描述：设置标签长度。

方法签名：

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

参数：

参数名	类型	必填	说明
length	Int	是	标签长度（点）

示例代码：

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

12. setPrintSpeed

描述：设置打印速度。

方法签名：

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

参数：

参数名	类型	必填	说明
speed	Int	是	打印速度（英寸/秒）

示例代码：

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

13. setPrintDarkness

描述：设置打印浓度。

方法签名：

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

参数：

参数名	类型	必填	说明
darkness	Int	是	浓度值（0-30）

示例代码：

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

14. setLabelHome

描述：设置标签原点。

方法签名：

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

参数：

参数名	类型	必填	说明
x	Int	是	X坐标（点）
y	Int	是	Y坐标（点）

示例代码：

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

15. setLabelShift

描述：设置标签偏移。

方法签名：

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

参数：

参数名	类型	必填	说明
shift	Int	是	偏移量（点）

示例代码：

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

16. setFieldOrigin

描述：设置字段原点。

方法签名：

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

参数：

参数名	类型	必填	说明
x	Int	是	X坐标（点）
y	Int	是	Y坐标（点）

示例代码：

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

17. setFieldOrientation

描述：设置字段方向。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
rotation	Char	否	'N'	旋转（N=正常, R=旋转90°, I=反转180°, B=底部朝上 270°）
justification	Char	否	'L'	对齐（L=左对齐, C=居中, R=右对齐, J=两端对齐）

示例代码：

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

18. fieldData

描述：发送字段数据。

方法签名：

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

参数：

参数名	类型	必填	说明
data	String	是	字段数据

示例代码：

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

19. fieldSeparator

描述：发送字段分隔符。

方法签名：

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

返回值：Result<Unit> 操作结果

示例代码：

```
printer.fieldSeparator()
```

20. comment

描述：添加注释。

方法签名：

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

参数：

参数名	类型	必填	说明
text	String	是	注释文本

示例代码：

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

21. getSerialNumber

描述：获取打印机序列号。

方法签名：

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

返回值：`Result<String>` 序列号

示例代码：

```
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

描述：获取打印机固件版本。

方法签名：

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

返回值: `Result<String>` 固件版本

示例代码:

```
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

描述: 执行自检。

方法签名:

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

返回值: `Result<Unit>` 操作结果

示例代码:

```
printer.selfTest()
```

24. setTaskId

描述: 设置任务ID。

方法签名:

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

参数:

参数名	类型	必填	说明
id	Int	是	任务ID

示例代码:

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

25. print

描述：执行打印。标签打印的前提条件，必须在所有打印内容之后调用。

方法签名：

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

返回值： `Result<Unit>` 打印结果

示例代码：

```
// 开始标签格式
printer.start()
// 添加打印内容
printer.printText(50, 30, 0, 'N', 30, 30, "Label Content")
printer.printQRcode(200, 150, 'N', 5, 5, "QR Data")
// 执行打印
printer.print()
```

26. cutPaper

描述：切纸。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
partial	Boolean	否	false	是否部分切纸

返回值： `Result<Unit>` 操作结果

示例代码：

```
// 完全切纸
printer.cutPaper()

// 部分切纸
printer.cutPaper(partial = true)
```

27. printQuantity

描述：设置打印份数。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
quantity	Int	是	-	打印份数
pauseCount	Int	否	0	暂停次数
replicateCount	Int	否	0	重复份数
overridePause	Char	否	'N'	覆盖暂停（Y=是, N=否）

示例代码：

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

28. cleanMissedHitsTask

描述：清理未命中任务。

方法签名：

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

返回值： `Result<Unit>` 操作结果

示例代码：

```
printer.cleanMissedHitsTask()
```

29. writeRFID

描述：写入 RFID 数据。

方法签名：

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

参数：

参数名	类型	必填	说明
address	Int	是	起始地址（字节偏移量）
memory	Int	是	内存区域（0=EPC, 1=用户内存, 2=TID）
data	ByteArray	是	RFID 数据

示例代码：

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

30. readRFID

描述：读取 RFID 数据。

方法签名：

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

参数：

参数名	类型	必填	说明
address	Int	是	起始地址（字节偏移量）
length	Int	是	读取长度（字节）
memory	Int	是	内存区域（0=EPC, 1=用户内存, 2=TID）

返回值：`Result<ByteArray>` RFID 数据

示例代码：

```
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

描述：设置语言编码。

方法签名：

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

参数：

参数名	类型	必填	说明
languageEncode	String	是	语言编码

示例代码：

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

32. setPrinterModel

描述：设置打印机型号。

方法签名：

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

参数：

参数名	类型	必填	说明
model	String	是	打印机型号

示例代码：

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

33. printPRNTemplate

描述：打印PRN模板。

方法签名：

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

参数：

参数名	类型	必填	说明
prnContent	String	是	PRN内容

示例代码：

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

34. drawBox

描述：绘制方框。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
width	Int	是	-	方框宽度（点）
height	Int	是	-	方框高度（点）
thickness	Int	是	-	线宽（点）
color	Char	否	'B'	颜色（B=黑色, W=白色）
rounding	Int	否	0	圆角半径（ZPL不支持，固定为0）

示例代码：

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

35. drawCircle

描述：绘制圆。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
diameter	Int	是	-	直径（点）
thickness	Int	是	-	线宽（点）
color	Char	否	'B'	颜色（B=黑色, W=白色）（ZPL不支持，固定为'B'）

示例代码：

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

36. drawDiagonalLine

描述：绘制对角线。

方法签名：

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

参数：

参数名	类型	必填	默认值	说明
width	Int	是	-	宽度（点）
height	Int	是	-	高度（点）
thickness	Int	是	-	线宽（点）
color	Char	否	'B'	颜色（B=黑色, W=白色）
orientation	Char	否	'R'	方向（R=右下斜, L=左下斜）

示例代码：

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

37. 附录

37.1 字体枚举 (ZplFont)

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

37.2 旋转角度枚举 (Rotation)

值	说明
NORMAL	正常 (0度)
ROTATE_90	旋转90度
ROTATE_180	旋转180度
ROTATE_270	旋转270度

37.3 条码类型枚举 (BarcodeType)

值	说明
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 二维码模型枚举 (QRModel)

值	说明
MODEL_1	模型1
MODEL_2	模型2

37.5 抖动算法枚举 (DitheringAlgorithm)

值	说明
THRESHOLD	简单阈值
DIFFUSION	扩散抖动
CLUSTER	聚集抖动

37.6 错误类型说明

错误类型	说明	可能的触发场景
ConnectionError	连接错误	打印机连接失败、网络连接问题、蓝牙配对失败
CommunicationError	通信错误	数据传输失败、打印机响应异常、通信超时
InvalidParameterError	参数无效错误	传入的参数值超出范围、参数格式不正确
TimeoutError	超时错误	操作执行超时、打印机无响应
NotConnectedError	未连接错误	尝试在未连接状态下执行操作
UnsupportedOperationError	不支持的操作错误	打印机型号不支持的功能、协议不支持的命令
UnknownError	未知错误	其他未分类的错误情况

错误码说明：接口返回 `Result.Failure` 时，`PrinterError` 携带 `errorCode` 字段（对应 `UXPErrCode.value`）。每个错误类型默认取下表兜底错误码，SDK 各调用点会传入更具体的错误码；完整错误码表见《UXPErrCode 错误码说明》（Guide 目录）。

兜底错误码	数值	说明
ConnectionError	1001	连接错误
CommunicationError	1002	通讯错误
InvalidParameterError	1003	参数无效
TimeoutError	1004	超时
NotConnectedError	1005	未连接
UnsupportedOperationError	1006	不支持的操作
UnknownError	1007	未知错误

错误处理示例：

```
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}")
        }
    }
}
}
```