

Common 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 saveParameters 保存接口说明，设置接口添加保存参数注释	邱贤	林德春	2026.04.25
3	v1.0.2	新增越南文变形和阿拉伯语变形接口，新增字库升级接口 fontDownload	邱贤	林德春	2026.06.15
4	v1.0.3	新增UTF-8输入模式设置和读取接口 setUtf8InputMode, getUtf8InputMode	邱贤	林德春	2026.07.16
5	v1.0.4	新增 OTA 固件升级接口 otaUpgrade； UXPErrorCode 错误码表独立成文档（Guide 目录），PrinterError 增加 errorCode 字段	邱贤	林德春	2026.08.04
6	v1.0.5	新增 WiFi 参数读写接口与 WiFi 热点扫描接口	邱贤	林德春	2026.09.16

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" />
```

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

在使用 `CommonPrinter` 之前，需要先建立打印机连接：

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

// 创建 CommonPrinter 实例
val printer = CommonPrinter(connection)
```

2. 接口目录

Common 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. 设备信息

- 3.1 `getPrinterModel`
- 3.2 `getSerialNumber`
- 3.3 `getPrinterVersion`
- 3.4 `getBatteryLevel`

4. 纸张设置

- 4.1 `getPaperType`
- 4.2 `setPaperType`

5. 打印设置

- 5.1 `setReprintSwitch`
- 5.2 `setPrintDensity`
- 5.3 `setPrintWidth`

- 5.4 getPrintWidth
- 6. WiFi设置
 - 6.1 getPrinterWifiMode
 - 6.2 setWifiMode
 - 6.3 getWifiSwitch
 - 6.4 setWifiSwitch
 - 6.5 getEthernetSwitch
 - 6.6 setEthernetSwitch
 - 6.7 getWifiStaSsid
 - 6.8 setWifiStaSsid
 - 6.9 getWifiStaEncryption
 - 6.10 setWifiStaEncryption
 - 6.11 getWifiStaPassword
 - 6.12 setWifiStaPassword
 - 6.13 getWifiStalp
 - 6.14 setWifiStalp
 - 6.15 getWifiStaDhcp
 - 6.16 setWifiStaDhcp
 - 6.17 getWifiApSsid
 - 6.18 setWifiApSsid
 - 6.19 getWifiApPassword
 - 6.20 setWifiApPassword
 - 6.21 getWifiApIp
 - 6.22 setWifiApIp
 - 6.23 restartWifi
 - 6.24 getWifiHotspots
- 7. 语言设置
 - 7.1 setPrinterLanguage
- 8. 代码页设置
 - 8.1 setCodePage
- 9. 字符集设置
 - 9.1 setCharacterSet
- 10. 文本变形设置
 - 10.1 setVietnameseTextTransform
 - 10.2 getVietnameseTextTransform
 - 10.3 setArabicTextTransform
 - 10.4 getArabicTextTransform
 - 10.5 setUtf8InputMode
 - 10.6 getUtf8InputMode
- 11. 打印机控制
 - 11.1 resetPrinter
- 12. 保存参数
 - 12.1 saveParameters
- 13. 字库升级
 - 13.1 fontDownload
- 14. OTA 固件升级
 - 14.1 otaUpgrade
- 15. 附录
 - 15.1 纸张类型枚举 (PaperType)

- 15.2 WiFi模式枚举 (WifiMode)
- 15.3 打印机语言枚举 (PrinterLanguage)
- 15.4 错误类型说明
- 15.5 错误码枚举 (UXPErrorCode)
- 15.6 WiFi 热点信息 (WifiHotspotInfo)
- 15.7 WiFi 加密方式枚举 (WifiEncryption)

3. 设备信息

3.1 getPrinterModel

描述：获取打印机型号。

方法签名：

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

返回值： `Result<String>` 打印机型号

示例代码：

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

3.2 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}")
}
```

3.3 getPrinterVersion

描述：获取打印机固件版本。

方法签名：

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

返回值： `Result<String>` 固件版本

示例代码：

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

3.4 getBatteryLevel

描述：获取打印机电池电量。

方法签名：

```
suspend fun getBatteryLevel(): Result<Int>
```

返回值： `Result<Int>` 电量百分比 (0-100)

示例代码：

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

4. 纸张设置

4.1 getPaperType

描述：获取纸张类型。

方法签名：

```
suspend fun getPaperType(): Result<PaperType>
```

返回值： `Result<PaperType>` 纸张类型

PaperType 枚举值：

值	说明
CONTINUOUS	连续纸
LABEL	标签纸
BLACK_MARK_FRONT_2INCH	2寸前黑标
BLACK_MARK_FRONT_3INCH	3寸前黑标
BLACK_MARK_BACK_2INCH	2寸后黑标
BLACK_MARK_BACK_3INCH	3寸后黑标

示例代码：

```
val result = printer.getPaperType()
when (result) {
    is Result.Success -> {
        when (result.value) {
            PaperType.CONTINUOUS -> Log.d("Printer", "连续纸")
            PaperType.LABEL -> Log.d("Printer", "标签纸")
            PaperType.BLACK_MARK_FRONT_2INCH -> Log.d("Printer", "2寸前黑标")
            PaperType.BLACK_MARK_FRONT_3INCH -> Log.d("Printer", "3寸前黑标")
            PaperType.BLACK_MARK_BACK_2INCH -> Log.d("Printer", "2寸后黑标")
            PaperType.BLACK_MARK_BACK_3INCH -> Log.d("Printer", "3寸后黑标")
        }
    }
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

4.2 setPaperType

描述：设置纸张类型。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setPaperType(paperType: PaperType): Result<Unit>
```

参数：

参数名	类型	必填	说明
paperType	PaperType	是	纸张类型

PaperType 枚举值：

值	说明
CONTINUOUS	连续纸
LABEL	标签纸
BLACK_MARK_FRONT_2INCH	2寸前黑标
BLACK_MARK_FRONT_3INCH	3寸前黑标
BLACK_MARK_BACK_2INCH	2寸后黑标
BLACK_MARK_BACK_3INCH	3寸后黑标

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setPaperType(PaperType.LABEL)
when (result) {
    is Result.Success -> Log.d("Printer", "Paper type set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

5. 打印设置

5.1 setReprintSwitch

描述：设置重印开关。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setReprintSwitch(enable: Boolean): Result<Unit>
```

参数：

参数名	类型	必填	说明
enable	Boolean	是	是否启用重印

返回值： `Result<Unit>` 设置结果

示例代码：

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

5.2 setPrintDensity

描述：设置打印浓度。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setPrintDensity(density: Int): Result<Unit>
```

参数：

参数名	类型	必填	说明
density	Int	是	浓度值（0-15）

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setPrintDensity(10)
when (result) {
    is Result.Success -> Log.d("Printer", "Density set to 10")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

5.3 setPrintWidth

描述：设置打印宽度。设置完成后需要调用 `saveParameters()` 保存参数。

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（点数）

返回值： `Result<Unit>` 设置结果

示例代码：

```
// 设置打印宽度为 80mm (576点)
val result = printer.setPrintWidth(576)
when (result) {
    is Result.Success -> Log.d("Printer", "Width set to 576")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

5.4 getPrintWidth

描述：获取打印宽度。

方法签名：

```
suspend fun getPrintWidth(): Result<Int>
```

返回值： `Result<Int>` 打印宽度（点数）

示例代码：

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

6. WiFi设置

6.1 getPrinterWifiMode

描述：获取打印机 WiFi 模式。

方法签名：

```
suspend fun getPrinterWifiMode(): Result<Int>
```

返回值： `Result<Int>` WiFi 模式（1=AP模式，2=Station模式）

示例代码：

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

6.2 setWifiMode

描述：设置打印机 WiFi 模式。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiMode(mode: WifiMode): Result<Unit>
```

参数：

参数名	类型	必填	说明
mode	WifiMode	是	WiFi 模式

WifiMode 枚举值：

值	说明
AP	接入点模式
STATION	站点模式

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiMode(WifiMode.STATION)
when (result) {
    is Result.Success -> Log.d("Printer", "WiFi mode set to STATION")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.3 getWifiSwitch

描述：获取 WiFi 开关状态。

方法签名：

```
suspend fun getWifiSwitch(): Result<Boolean>
```

返回值: `Result<Boolean>` true: WiFi 已开启, false: WiFi 已关闭

示例代码:

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

6.4 setWifiSwitch

描述: 设置 WiFi 开关。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名:

```
suspend fun setWifiSwitch(enabled: Boolean): Result<Unit>
```

参数:

参数名	类型	必填	说明
enabled	Boolean	是	true: 开启, false: 关闭

返回值: `Result<Unit>` 设置结果

示例代码:

```
val result = printer.setWifiSwitch(true)
when (result) {
    is Result.Success -> Log.d("Printer", "WiFi switch set to on")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.5 getEthernetSwitch

描述: 获取以太网开关状态。

方法签名:

```
suspend fun getEthernetSwitch(): Result<Boolean>
```

返回值: `Result<Boolean>` true: 以太网已开启, false: 以太网已关闭

示例代码:

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

6.6 setEthernetSwitch

描述：设置以太网开关。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setEthernetSwitch(enabled: Boolean): Result<Unit>
```

参数：

参数名	类型	必填	说明
enabled	Boolean	是	true：开启，false：关闭

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setEthernetSwitch(false)
when (result) {
    is Result.Success -> Log.d("Printer", "Ethernet switch set to off")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.7 getWifiStaSsid

描述：获取 STA 模式下的 WiFi 名称。

方法签名：

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

返回值： `Result<String>` WiFi 名称

示例代码：

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

6.8 setWifiStaSsid

描述：设置 STA 模式下的 WiFi 名称。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiStaSsid(ssid: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
ssid	String	是	WiFi 名称，UTF-8 编码，最长 32 字节，超出部分截断

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiStaSsid("Office_WiFi")
when (result) {
    is Result.Success -> Log.d("Printer", "STA SSID set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.9 getWifiStaEncryption

描述：获取 STA 模式下的 WiFi 加密方式。

方法签名：

```
suspend fun getWifiStaEncryption(): Result<WifiEncryption>
```

返回值： `Result<WifiEncryption>` WiFi 加密方式

WifiEncryption 枚举值：

值	固件取值	说明
OPEN	0	不加密
WEP	1	WEP
WPA2_PSK	2	WPA2-PSK
WPA_WPA2_PSK	3	WPA/WPA2-PSK
WPA_PSK	4	WPA-PSK
WPA	5	WPA
WPA2	6	WPA2
SAE	7	SAE
WPA3_WPA2_PSK_MIX	8	WPA3-WPA2-PSK-MIX
UNKNOWN	9	未知

示例代码：

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

6.10 setWifiStaEncryption

描述：设置 STA 模式下的 WiFi 加密方式。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiStaEncryption(encryption: WifiEncryption): Result<Unit>
```

参数：

参数名	类型	必填	说明
encryption	WifiEncryption	是	WiFi 加密方式，取值见上表

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiStaEncryption(WifiEncryption.WPA2_PSK)
when (result) {
    is Result.Success -> Log.d("Printer", "STA encryption set to WPA2_PSK")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.11 getWifiStaPassword

描述：获取 STA 模式下的 WiFi 密码。

方法签名：

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

返回值： `Result<String>` WiFi 密码

示例代码：

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

6.12 setWifiStaPassword

描述：设置 STA 模式下的 WiFi 密码。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiStaPassword(password: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
password	String	是	WiFi 密码，UTF-8 编码，最长 32 字节，超出部分截断

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiStaPassword("12345678")
when (result) {
    is Result.Success -> Log.d("Printer", "STA password set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.13 getWifiStaIp

描述：获取 STA 模式下的静态 IP 地址。

方法签名：

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

返回值： `Result<String>` 点分十进制 IP 地址

示例代码：

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

6.14 setWifiStaIp

描述：设置 STA 模式下的静态 IP 地址。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiStaIp(ip: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
ip	String	是	点分十进制 IP 地址，如 192.168.1.100；格式非法返回 <code>InvalidParameterError</code> （错误码 1313）

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiStaIp("192.168.1.100")
when (result) {
    is Result.Success -> Log.d("Printer", "STA IP set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message},
code=${result.error.errorCode}")
}
```

6.15 getWifiStaDhcp

描述：获取 STA 模式下的动态 IP（DHCP）开关状态。

方法签名：

```
suspend fun getWifiStaDhcp(): Result<Boolean>
```

返回值： `Result<Boolean>` true: 动态 IP, false: 静态 IP

示例代码：

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

6.16 setWifiStaDhcp

描述：设置 STA 模式下的动态 IP（DHCP）开关。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiStaDhcp(enabled: Boolean): Result<Unit>
```

参数：

参数名	类型	必填	说明
enabled	Boolean	是	true: 动态 IP, false: 静态 IP

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiStaDhcp(true)
when (result) {
    is Result.Success -> Log.d("Printer", "STA DHCP enabled")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.17 getWifiApSsid

描述：获取 AP 模式下的 WiFi 名称。

方法签名：

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

返回值： `Result<String>` WiFi 名称

示例代码：

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

6.18 setWifiApSsid

描述：设置 AP 模式下的 WiFi 名称。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiApSsid(ssid: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
ssid	String	是	WiFi 名称，UTF-8 编码，最长 32 字节，超出部分截断

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiApSsid("POS_WIFI")
when (result) {
    is Result.Success -> Log.d("Printer", "AP SSID set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.19 getWifiApPassword

描述：获取 AP 模式下的 WiFi 密码。

方法签名：

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

返回值： `Result<String>` WiFi 密码

示例代码：

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

6.20 setWifiApPassword

描述：设置 AP 模式下的 WiFi 密码。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiApPassword(password: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
password	String	是	WiFi 密码，UTF-8 编码，最长 32 字节，超出部分截断

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiApPassword("12345678")
when (result) {
    is Result.Success -> Log.d("Printer", "AP password set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

6.21 getWifiApIp

描述：获取 AP 模式下的 IP 地址。

方法签名：

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

返回值： `Result<String>` 点分十进制 IP 地址

示例代码：

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

6.22 setWifiApIp

描述：设置 AP 模式下的 IP 地址。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setWifiApIp(ip: String): Result<Unit>
```

参数：

参数名	类型	必填	说明
ip	String	是	点分十进制 IP 地址，如 192.168.1.1；格式非法返回 <code>InvalidParameterError</code> （错误码 1313）

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setWifiApIp("192.168.1.1")
when (result) {
    is Result.Success -> Log.d("Printer", "AP IP set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message},
code=${result.error.errorCode}")
}
```

6.23 restartWifi

描述：重启打印机 WiFi，使 WiFi 参数生效。

方法签名：

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

返回值： `Result<Unit>` 重启结果

示例代码：

```
printer.setWifiStaSsid("Office_WiFi")
    .flatMap { printer.saveParameters() }
    .flatMap { printer.restartWifi() }
    .onSuccess { Log.d("Printer", "WiFi restarted") }
    .onFailure { Log.e("Printer", "Error: ${it.message}") }
```

6.24 getWifiHotspots

描述：扫描打印机周边的 WiFi 热点。SDK 发送扫描指令后累积读取多包响应，解析完成后返回热点列表。

方法签名：

```
suspend fun getWifiHotspots(): Result<List<WifiHotspotInfo>>
```

返回值： `Result<List<WifiHotspotInfo>>` WiFi 热点列表；未读取到数据或解析结果为空时返回 `Result.Failure`

响应格式：打印机返回文本以 `WiFi_scan_ap:` 开头，随后每行一个热点，字段以制表符分隔：

```
AP=n/m \t 热点名称 \t 加密协议 \t 加密方式(AES/NONE) \t 信号强度 \r\n
```

其中 `n/m` 表示当前第 `n` 个，共 `m` 个。

示例代码：

```

val result = printer.getWifiHotspots()
when (result) {
    is Result.Success -> {
        result.data.forEach { hotspot ->
            Log.d("Printer", "AP: ${hotspot.name}, security:
${hotspot.security}, " +
                "encryption: ${hotspot.encryption}, signal:
${hotspot.signalStrength}")
        }
    }
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}

```

7. 语言设置

7.1 setPrinterLanguage

描述：设置打印机语言。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setPrinterLanguage(language: PrinterLanguage): Result<Unit>
```

参数：

参数名	类型	必填	说明
language	PrinterLanguage	是	打印机语言

PrinterLanguage 枚举值：

值	说明
ENGLISH	英语
CHINESE_SIMPLIFIED	简体中文
CHINESE_TRADITIONAL	繁体中文

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setPrinterLanguage(PrinterLanguage.CHINESE_SIMPLIFIED)
when (result) {
    is Result.Success -> Log.d("Printer", "Language set to CHINESE_SIMPLIFIED")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

8. 代码页设置

8.1 setCodePage

描述：设置打印机代码页。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setCodePage(codePage: CommonCommands.CodePage): Result<Unit>
```

参数：

参数名	类型	必填	说明
codePage	CommonCommands.CodePage	是	代码页枚举

返回值：`Result<Unit>` 设置结果

示例代码：

```
val result = printer.setCodePage(CommonCommands.CodePage.PC437)
when (result) {
    is Result.Success -> Log.d("Printer", "Code page set to PC437")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

代码页枚举值：

代码页名称	设置值	说明
PC437	0	美国标准 ASCII
KATAKANA	1	片假名
PC850	2	多语言
PC860	3	葡萄牙语
PC863	4	加拿大法语
PC865	5	北欧语

PC857	13	土耳其语
PC737	14	希腊语
ISO8859_7	15	希腊语
WPC1252	16	拉丁 1
PC866	17	西里尔文
PC852	18	中欧
PC858	19	多语言带 euro
KU42	20	泰国语
TIS11	21	泰国语
TIS18	26	泰国语
PC720	32	阿拉伯语
WPC775	33	波罗的海语
PC855	33	西里尔文
PC862	36	希伯来语
PC864	37	阿拉伯语
ISO8859_2	39	中欧
ISO8859_15	40	拉丁 9
WPC1250	45	中欧
WPC1251	46	西里尔文
WPC1253	47	希腊语
WPC1254	48	土耳其语
WPC1255	49	希伯来语
WPC1256	50	阿拉伯语
WPC1257	51	波罗的海语
WPC1258	52	越南语
MIK	54	保加利亚语
CP755	55	西里尔文
IRAN	56	波斯语

IRAN_II	57	波斯语
LATVIAN	58	拉脱维亚语
ISO_8859_1	59	拉丁 1
ISO_8859_3	60	南欧
ISO_8859_4	61	北欧
ISO_8859_5	62	西里尔文
ISO_8859_6	63	阿拉伯语
ISO_8859_8	64	希伯来语
ISO_8859_9	65	土耳其语
PC856	66	希伯来语
ABICOIM	67	葡萄牙语
KHEMR	68	高棉语

说明：KHEMR 条目追加在代码页表的末尾。

9. 字符集设置

9.1 setCharacterSet

描述：设置打印机国际字符集。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setCharacterSet(characterSet: CommonCommands.CharacterSet):  
Result<Unit>
```

参数：

参数名	类型	必填	说明
characterSet	CommonCommands.CharacterSet	是	字符集枚举

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setCharacterSet(CommonCommands.CharacterSet.USA)
when (result) {
    is Result.Success -> Log.d("Printer", "Character set set to USA")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

字符集枚举值：

字符集名称	设置值	说明
USA	0	美国
FRANCE	1	法国
GERMANY	2	德国
UK	3	英国
DENMARK_I	4	丹麦 I
SWEDEN	5	瑞典
ITALY	6	意大利
SPAIN_I	7	西班牙 I
JAPAN	8	日本
NORWAY	9	挪威
DENMARK_II	10	丹麦 II
SPAIN_II	11	西班牙 II
LATIN_AMERICA	12	拉丁美洲
KOREA	13	韩国
SLOVENIA_CROATIA	14	斯洛文尼亚克罗地亚
CHINA	15	中国

10. 文本变形设置

10.1 setVietnameseTextTransform

描述：设置越南文变形。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setVietnameseTextTransform(mode: Int): Result<Unit>
```

参数：

参数名	类型	必填	说明
mode	Int	是	变形模式（0：关闭，1：变形（以ASCII输入），2：变形（以UTF-8输入））

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setVietnameseTextTransform(1)
when (result) {
    is Result.Success -> Log.d("Printer", "Vietnamese text transform set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

10.2 getVietnameseTextTransform

描述：读取越南文变形设置。

方法签名：

```
suspend fun getVietnameseTextTransform(): Result<Int>
```

返回值： `Result<Int>` 变形模式（0：关闭，1：变形（以ASCII输入），2：变形（以UTF-8输入））

示例代码：

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

10.3 setArabicTextTransform

描述：设置阿拉伯语变形。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setArabicTextTransform(mode: Int): Result<Unit>
```

参数：

参数名	类型	必填	说明
mode	Int	是	变形模式（0：关闭，1：变形（按单词排序），2：变形（按短语排序），3：变形（按完整规则排序））

返回值： `Result<Unit>` 设置结果

示例代码：

```
val result = printer.setArabicTextTransform(1)
when (result) {
    is Result.Success -> Log.d("Printer", "Arabic text transform set")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

10.4 getArabicTextTransform

描述：读取阿拉伯语变形设置。

方法签名：

```
suspend fun getArabicTextTransform(): Result<Int>
```

返回值： `Result<Int>` 变形模式（0：关闭，1：变形（按单词排序），2：变形（按短语排序），3：变形（按完整规则排序））

示例代码：

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

10.5 setUtf8InputMode

描述：设置 UTF-8 输入模式。设置完成后需要调用 `saveParameters()` 保存参数。

方法签名：

```
suspend fun setUtf8InputMode(mode: Int): Result<Unit>
```

参数:

参数名	类型	必填	说明
mode	Int	是	输入模式 (0: 关闭, 1: 开启 UTF-8 输入)

返回值: `Result<Unit>` 设置结果

示例代码:

```
val result = printer.setUtf8InputMode(1)
    .flatMap { printer.saveParameters() }
when (result) {
    is Result.Success -> Log.d("Printer", "UTF-8 input mode set to on")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

10.6 getUtf8InputMode

描述: 读取 UTF-8 输入模式设置。

方法签名:

```
suspend fun getUtf8InputMode(): Result<Int>
```

返回值: `Result<Int>` 输入模式 (0: 关闭, 1: 开启 UTF-8 输入)

示例代码:

```
val result = printer.getUtf8InputMode()
when (result) {
    is Result.Success -> Log.d("Printer", "UTF-8 input mode: ${result.value}")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

11. 打印机控制

11.1 resetPrinter

描述: 重启打印机。

方法签名:

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

返回值: `Result<Unit>` 重启结果

示例代码:

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

12. 保存参数

12.1 saveParameters

描述: 保存参数到打印机。设置完打印机参数后, 需要调用此方法将参数保存到打印机。

方法签名:

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

返回值: `Result<Unit>` 保存结果

示例代码:

```
printer.setPaperType(PaperType.LABEL)
    .flatMap { printer.saveParameters() }
    .onSuccess { Log.d("Printer", "Paper type saved") }
    .onFailure { Log.e("Printer", "Error: ${it.message}") }
```

13. 字库升级

13.1 fontDownload

描述: 升级打印机字库。分块发送字库数据, 每包失败自动重试一次, 通过回调通知进度。

方法签名:

```
suspend fun fontDownload(
    fontData: ByteArray,
    onProgress: (percent: Int) -> Unit
): Result<Unit>
```

参数:

参数名	类型	必填	说明
fontData	ByteArray	是	字库文件二进制数据
onProgress	(Int) -> Unit	是	进度回调, percent=0~100 百分比

返回值: `Result<Unit>` 升级结果

示例代码:

```
val fontFile = File("/sdcard/font.bin")
val fontData = fontFile.readBytes()
val result = printer.fontDownload(fontData) { percent ->
    Log.d("Printer", "Progress: $percent%")
}
when (result) {
    is Result.Success -> Log.d("Printer", "Font download completed")
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

14. OTA 固件升级

14.1 otaUpgrade

描述: 通过 OTA 升级打印机固件。SDK 会先进入 OTA 模式、获取每包最大字节数, 然后分块发送固件数据, 通过回调通知进度。失败时错误码见 `PrinterError.errorCode` (完整 `UXPErrorCode` 枚举见《UXPErrorCode 错误码说明》, Guide 目录)。

方法签名:

```
suspend fun otaUpgrade(
    firmwareData: ByteArray,
    progressBlock: (percent: Int) -> Unit
): Result<UXPErrorCode>
```

参数:

参数名	类型	必填	说明
firmwareData	ByteArray	是	固件文件二进制数据
progressBlock	(Int) -> Unit	是	进度回调, percent=0~100 百分比

返回值: `Result<UXPErrorCode>` 升级结果。成功为

`Result.Success(UXPErrorCode.Success)`; 失败时错误码见 `PrinterError.errorCode` (对应 `UXPErrorCode.value`)。

升级流程：1) 发送 `enter_ota` 命令进入 OTA 模式；2) 查询 `pack_length` 获取每包最大字节数；3) 分块发送固件数据。每包包含偏移、长度、CRC32 和数据，打印机回传状态字节（0x00 OK / 0x01 LENGTH_ERROR / 0x02 CRC_ERROR / 0x03 TIME_OUT / 0x04 WRITE_FAIL / 0x05 FAIL），映射为对应的 `OtaStatus*` 错误码。

示例代码：

```
val firmwareFile = File("/sdcard/firmware.bin")
val firmwareData = firmwareFile.readBytes()
val result = printer.otaUpgrade(firmwareData) { percent ->
    Log.d("Printer", "Progress: $percent%")
}
when (result) {
    is Result.Success -> {
        if (result.data == UXPErrCode.Success) {
            Log.d("Printer", "OTA upgrade completed")
        }
    }
    is Result.Failure -> {
        Log.e("Printer", "Error: ${result.error.message},
code=${result.error.errorCode}")
    }
}
```

15. 附录

15.1 纸张类型枚举 (PaperType)

值	说明
CONTINUOUS	连续纸
LABEL	标签纸
BLACK_MARK_FRONT_2INCH	2寸前黑标
BLACK_MARK_FRONT_3INCH	3寸前黑标
BLACK_MARK_BACK_2INCH	2寸后黑标
BLACK_MARK_BACK_3INCH	3寸后黑标

15.2 WiFi模式枚举 (WifiMode)

值	说明
AP	接入点模式
STATION	站点模式

15.3 打印机语言枚举 (PrinterLanguage)

值	说明
ENGLISH	英语
CHINESE_SIMPLIFIED	简体中文
CHINESE_TRADITIONAL	繁体中文

15.4 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 `UXPErrorCode.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《UXPErrorCode 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.setPrinterLanguage(PrinterLanguage.CHINESE_SIMPLIFIED)
when (result) {
    is Result.Success -> {
        Log.d("Printer", "Language set to CHINESE_SIMPLIFIED")
    }
    is Result.Failure -> {
        when (result.error) {
            is PrinterError.ConnectionError -> {
                Log.e("Printer", "Connection error: ${result.error.message}")
                // 尝试重新连接
            }
            is PrinterError.CommunicationError -> {
                Log.e("Printer", "Communication error:
${result.error.message}")
                // 检查打印机状态
            }
            is PrinterError.InvalidParameterError -> {
                Log.e("Printer", "Invalid parameter: ${result.error.message}")
                // 修正参数值
            }
            is PrinterError.TimeoutError -> {
                Log.e("Printer", "Timeout error: ${result.error.message}")
                // 增加超时时间或检查打印机状态
            }
            is PrinterError.NotConnectedError -> {
                Log.e("Printer", "Not connected: ${result.error.message}")
                // 先连接打印机
            }
            is PrinterError.UnsupportedOperationError -> {
                Log.e("Printer", "Unsupported operation:
${result.error.message}")
                // 使用替代方案
            }
        }
    }
}
```

```
    }
    is PrinterError.UnknownError -> {
        Log.e("Printer", "Unknown error: ${result.error.message}")
        // 记录错误并尝试恢复
    }
}
}
```

15.5 错误码枚举 (UXPErrorCode)

UXPErrorCode 覆盖全部 PrinterError 错误类型和 OTA 升级错误。接口返回 Result.Failure 时，错误码见 PrinterError.errorCode（对应 UXPErrorCode.value）。完整错误码表见《UXPErrorCode 错误码说明》（Guide 目录）。

15.6 WiFi 热点信息 (WifiHotspotInfo)

字段	类型	说明	示例
index	String	热点索引，格式 n/m（n=当前序号，m=总数）	1/5
name	String	热点名称(SSID)	Office_WiFi
security	String	加密协议	WPA2-PSK
encryption	String	加密方式	AES
signalStrength	String	信号强度	-45

15.7 WiFi 加密方式枚举 (WifiEncryption)

值	固件取值	说明
OPEN	0	不加密
WEP	1	WEP
WPA2_PSK	2	WPA2-PSK
WPA_WPA2_PSK	3	WPA/WPA2-PSK
WPA_PSK	4	WPA-PSK
WPA	5	WPA
WPA2	6	WPA2
SAE	7	SAE
WPA3_WPA2_PSK_MIX	8	WPA3-WPA2-PSK-MIX
UNKNOWN	9	未知