

Common 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 saveParameters interface documentation and save parameters note for setter methods	Qiu Xian	Lin Dechun	2026.04.25
3	v1.0.2	Add Vietnamese and Arabic text transform interfaces, add font download interface fontDownload	Qiu Xian	Lin Dechun	2026.06.15
4	v1.0.3	Add UTF-8 input mode set/get interfaces setUtf8InputMode, getUtf8InputMode	Qiu Xian	Lin Dechun	2026.07.16
5	v1.0.4	Add OTA firmware upgrade interface otaUpgrade; extract UXPErrCode error code table into a standalone document (Guide folder), PrinterError gains errorCode field	Qiu Xian	Lin Dechun	2026.08.04
6	v1.0.5	Add WiFi parameter read/write interfaces and WiFi hotspot scan interface	Qiu Xian	Lin Dechun	2026.09.16

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" />
```

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

Before using CommonPrinter, establish a printer connection:

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

2. API Reference

Common 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. Device Information

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

4. Paper Settings

- 4.1 getPaperType
- 4.2 setPaperType

5. Print Settings

- 5.1 setReprintSwitch
- 5.2 setPrintDensity
- 5.3 setPrintWidth
- 5.4 getPrintWidth

6. WiFi Settings

- 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 getWifiStaIp
 - 6.14 setWifiStaIp
 - 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. Language Settings
 - 7.1 setPrinterLanguage
 - 8. Code Page Settings
 - 9.1 setCodePage
 - 9. Character Set Settings
 - 9.1 setCharacterSet
 - 10. Text Transform Settings
 - 10.1 setVietnameseTextTransform
 - 10.2 getVietnameseTextTransform
 - 10.3 setArabicTextTransform
 - 10.4 getArabicTextTransform
 - 10.5 setUtf8InputMode
 - 10.6 getUtf8InputMode
 - 11. Printer Control
 - 11.1 resetPrinter
 - 12. Save Parameters
 - 12.1 saveParameters
 - 13. Font Upgrade
 - 13.1 fontDownload
 - 14. OTA Firmware Upgrade
 - 14.1 otaUpgrade
 - 15. Appendix
 - 15.1 Paper Type Enum (PaperType)
 - 15.2 WiFi Mode Enum (WifiMode)
 - 15.3 Printer Language Enum (PrinterLanguage)
 - 15.4 Error Types
 - 15.5 Error Code Enum (UXPErrorCode)
 - 15.6 WiFi Hotspot Info (WifiHotspotInfo)
 - 15.7 WiFi Encryption Enum (WifiEncryption)
-

3. Device Information

3.1 getPrinterModel

Description: Get the printer model.

Method Signature:

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

Return Value: `Result<String>` Printer model

Example Code:

```
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

Description: Get the 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}")
}
```

3.3 getPrinterVersion

Description: Get the printer firmware version.

Method Signature:

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

Return Value: `Result<String>` Firmware version

Example Code:

```
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

Description: Get the printer battery level.

Method Signature:

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

Return Value: `Result<Int>` Battery percentage (0-100)

Example Code:

```
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. Paper Settings

4.1 getPaperType

Description: Get the paper type.

Method Signature:

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

Return Value: `Result<PaperType>` Paper type

PaperType Enum Values:

Value	Description
CONTINUOUS	Continuous paper
LABEL	Label paper
BLACK_MARK_FRONT_2INCH	2-inch front black mark
BLACK_MARK_FRONT_3INCH	3-inch front black mark
BLACK_MARK_BACK_2INCH	2-inch back black mark
BLACK_MARK_BACK_3INCH	3-inch back black mark

Example Code:

```
val result = printer.getPaperType()
when (result) {
    is Result.Success -> {
        when (result.value) {
            PaperType.CONTINUOUS -> Log.d("Printer", "Continuous paper")
            PaperType.LABEL -> Log.d("Printer", "Label paper")
            PaperType.BLACK_MARK_FRONT_2INCH -> Log.d("Printer", "2-inch front
black mark")
            PaperType.BLACK_MARK_FRONT_3INCH -> Log.d("Printer", "3-inch front
black mark")
            PaperType.BLACK_MARK_BACK_2INCH -> Log.d("Printer", "2-inch back
black mark")
            PaperType.BLACK_MARK_BACK_3INCH -> Log.d("Printer", "3-inch back
black mark")
        }
    }
    is Result.Failure -> Log.e("Printer", "Error: ${result.error.message}")
}
```

4.2 setPaperType

Description: Set the paper type. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
type	PaperType	Yes	Paper type

PaperType Enum Values:

Value	Description
CONTINUOUS	Continuous paper
LABEL	Label paper
BLACK_MARK_FRONT_2INCH	2-inch front black mark
BLACK_MARK_FRONT_3INCH	3-inch front black mark
BLACK_MARK_BACK_2INCH	2-inch back black mark
BLACK_MARK_BACK_3INCH	3-inch back black mark

Return Value: `Result<Unit>` Setting result

Example Code:

```
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. Print Settings

5.1 setReprintSwitch

Description: Set the reprint switch. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
enable	Boolean	Yes	Enable reprint

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Set the print density. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
density	Int	Yes	Density value (0-15)

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Set the print width. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

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

Return Value: `Result<Unit>` Setting result

Example Code:

```
// Set print width to 80mm (576 dots)
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

Description: Get the print width.

Method Signature:

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

Return Value: `Result<Int>` Print width (dots)

Example Code:

```
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 Settings

6.1 getPrinterWifiMode

Description: Get the printer WiFi mode.

Method Signature:

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

Return Value: `Result<Int>` WiFi mode (1=AP mode, 2=Station mode)

Example Code:

```
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

Description: Set the printer WiFi mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
mode	WifiMode	Yes	WiFi mode

WifiMode Enum Values:

Value	Description
AP	Access Point mode
STATION	Station mode

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi switch state.

Method Signature:

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

Return Value: `Result<Boolean>` true: WiFi enabled, false: WiFi disabled

Example Code:

```
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

Description: Set the WiFi switch. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
enabled	Boolean	Yes	true: enable, false: disable

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the Ethernet switch state.

Method Signature:

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

Return Value: `Result<Boolean>` true: Ethernet enabled, false: Ethernet disabled

Example Code:

```
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

Description: Set the Ethernet switch. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
enabled	Boolean	Yes	true: enable, false: disable

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi name (SSID) in Station mode.

Method Signature:

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

Return Value: `Result<String>` WiFi name

Example Code:

```
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

Description: Set the WiFi name (SSID) in Station mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
ssid	String	Yes	WiFi name, UTF-8 encoded, up to 32 bytes, truncated when longer

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi encryption mode in Station mode.

Method Signature:

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

Return Value: `Result<WifiEncryption>` WiFi encryption mode

WifiEncryption Enum Values:

Value	Firmware Value	Description
OPEN	0	No encryption
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	Unknown

Example Code:

```
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

Description: Set the WiFi encryption mode in Station mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
encryption	WifiEncryption	Yes	WiFi encryption mode, see the table above

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi password in Station mode.

Method Signature:

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

Return Value: `Result<String>` WiFi password

Example Code:

```
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

Description: Set the WiFi password in Station mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
password	String	Yes	WiFi password, UTF-8 encoded, up to 32 bytes, truncated when longer

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the static IP address in Station mode.

Method Signature:

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

Return Value: `Result<String>` Dotted decimal IP address

Example Code:

```
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

Description: Set the static IP address in Station mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
ip	String	Yes	Dotted decimal IP address, e.g. 192.168.1.100; returns <code>InvalidParameterError</code> (code 1313) when malformed

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the DHCP (dynamic IP) switch state in Station mode.

Method Signature:

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

Return Value: `Result<Boolean>` true: dynamic IP, false: static IP

Example Code:

```
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

Description: Set the DHCP (dynamic IP) switch in Station mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
enabled	Boolean	Yes	true: dynamic IP, false: static IP

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi name (SSID) in AP mode.

Method Signature:

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

Return Value: `Result<String>` WiFi name

Example Code:

```
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

Description: Set the WiFi name (SSID) in AP mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
ssid	String	Yes	WiFi name, UTF-8 encoded, up to 32 bytes, truncated when longer

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the WiFi password in AP mode.

Method Signature:

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

Return Value: `Result<String>` WiFi password

Example Code:

```
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

Description: Set the WiFi password in AP mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
password	String	Yes	WiFi password, UTF-8 encoded, up to 32 bytes, truncated when longer

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get the IP address in AP mode.

Method Signature:

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

Return Value: `Result<String>` Dotted decimal IP address

Example Code:

```
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

Description: Set the IP address in AP mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
ip	String	Yes	Dotted decimal IP address, e.g. 192.168.1.1; returns <code>InvalidParameterError</code> (code 1313) when malformed

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Restart the printer WiFi so that the WiFi parameters take effect.

Method Signature:

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

Return Value: `Result<Unit>` Restart result

Example Code:

```
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

Description: Scan WiFi hotspots around the printer. The SDK sends the scan command, accumulates the multi-packet response, and returns the parsed hotspot list.

Method Signature:

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

Return Value: `Result<List<WifiHotspotInfo>>` WiFi hotspot list; returns `Result.Failure` when no data is received or nothing can be parsed

Response Format: The printer replies with text starting with `wifi_scan_ap:`, followed by one line per hotspot with tab-separated fields:

```
AP=n/m \t SSID \t security \t encryption(AES/NONE) \t signal strength \r\n
```

Where `n/m` means the n-th hotspot out of m in total.

Example Code:

```

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. Language Settings

7.1 setPrinterLanguage

Description: Set the printer language. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

```

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

```

Parameters:

Parameter	Type	Required	Description
language	PrinterLanguage	Yes	Printer language

PrinterLanguage Enum Values:

Value	Description
ENGLISH	English
CHINESE_SIMPLIFIED	Simplified Chinese
CHINESE_TRADITIONAL	Traditional Chinese

Return Value: `Result<Unit>` Setting result

Example Code:

```

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. Code Page Settings

9.1 setCodePage

Description: Set the printer code page. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

```

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

```

Parameters:

Parameter	Type	Required	Description
codePage	CommonCommands.CodePage	Yes	Code page enum

Return Value: `Result<Unit>` Setting result

Example Code:

```

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}")
}

```

Code Page Enum Values:

Code Page Name	Value	Description
PC437	0	US Standard ASCII
KATAKANA	1	Katakana
PC850	2	Multilingual
PC860	3	Portuguese
PC863	4	Canadian French

PC865	5	Nordic
PC857	13	Turkish
PC737	14	Greek
ISO8859_7	15	Greek
WPC1252	16	Latin 1
PC866	17	Cyrillic
PC852	18	Central European
PC858	19	Multilingual with euro
KU42	20	Thai
TIS11	21	Thai
TIS18	26	Thai
PC720	32	Arabic
WPC775	33	Baltic
PC855	33	Cyrillic
PC862	36	Hebrew
PC864	37	Arabic
ISO8859_2	39	Central European
ISO8859_15	40	Latin 9
WPC1250	45	Central European
WPC1251	46	Cyrillic
WPC1253	47	Greek
WPC1254	48	Turkish
WPC1255	49	Hebrew
WPC1256	50	Arabic
WPC1257	51	Baltic
WPC1258	52	Vietnamese
MIK	54	Bulgarian
CP755	55	Cyrillic
IRAN	56	Persian

IRAN_II	57	Persian
LATVIAN	58	Latvian
ISO_8859_1	59	Latin 1
ISO_8859_3	60	South European
ISO_8859_4	61	Nordic
ISO_8859_5	62	Cyrillic
ISO_8859_6	63	Arabic
ISO_8859_8	64	Hebrew
ISO_8859_9	65	Turkish
PC856	66	Hebrew
ABICOIM	67	Portuguese

| KHEMR | 68 | Khmer

Note: The KHEMR entry is appended at the end of the code page table.

9. Character Set Settings

9.1 setCharacterSet

Description: Set the printer character set. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
characterSet	CommonCommands.CharacterSet	Yes	Character set enum

Return Value: `Result<Unit>` Setting result

Example Code:

```

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}")
}

```

Character Set Enum Values:

Character Set Name	Value	Description
USA	0	USA
FRANCE	1	France
GERMANY	2	Germany
UK	3	UK
DENMARK_I	4	Denmark I
SWEDEN	5	Sweden
ITALY	6	Italy
SPAIN_I	7	Spain I
JAPAN	8	Japan
NORWAY	9	Norway
DENMARK_II	10	Denmark II
SPAIN_II	11	Spain II
LATIN_AMERICA	12	Latin America
KOREA	13	Korea
SLOVENIA_CROATIA	14	Slovenia Croatia
CHINA	15	China

10. Text Transform Settings

10.1 setVietnameseTextTransform

Description: Set Vietnamese text transform. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
mode	Int	Yes	Transform mode (0: Off, 1: Transform (ASCII input), 2: Transform (UTF-8 input))

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get Vietnamese text transform setting.

Method Signature:

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

Return Value: `Result<Int>` Transform mode (0: Off, 1: Transform (ASCII input), 2: Transform (UTF-8 input))

Example Code:

```
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

Description: Set Arabic text transform. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
mode	Int	Yes	Transform mode (0: Off, 1: Transform (by word order), 2: Transform (by phrase order), 3: Transform (by full rules))

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get Arabic text transform setting.

Method Signature:

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

Return Value: `Result<Int>` Transform mode (0: Off, 1: Transform (by word order), 2: Transform (by phrase order), 3: Transform (by full rules))

Example Code:

```
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

Description: Set UTF-8 input mode. After setting, you need to call `saveParameters()` to save the parameters.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
mode	Int	Yes	Input mode (0: Off, 1: On (UTF-8 Input))

Return Value: `Result<Unit>` Setting result

Example Code:

```
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

Description: Get UTF-8 input mode setting.

Method Signature:

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

Return Value: `Result<Int>` Input mode (0: Off, 1: On (UTF-8 Input))

Example Code:

```
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. Printer Control

11.1 resetPrinter

Description: Reset the printer.

Method Signature:

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

Return Value: `Result<Unit>` Reset result

Example Code:

```
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. Save Parameters

12.1 saveParameters

Description: Save parameters to the printer. After setting printer parameters, you need to call this method to save the parameters to the printer.

Method Signature:

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

Return Value: `Result<Unit>` Save result

Example Code:

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

13. Font Upgrade

13.1 fontDownload

Description: Upgrade printer font library. Data is sent in chunks with automatic single retry on failure. Progress is reported via callback.

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
fontData	ByteArray	Yes	Font file binary data
onProgress	(Int) -> Unit	Yes	Progress callback, percent=0~100

Return Value: `Result<Unit>` Upgrade result

Example Code:

```
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 Firmware Upgrade

14.1 otaUpgrade

Description: Upgrade printer firmware via OTA. The SDK enters OTA mode, queries the max package size, and sends firmware data in chunks. Progress is reported via callback. On failure, the error code is available in `PrinterError.errorCode` (see the `UXPErrorCode` Error Code Documentation in the Guide folder for the full `UXPErrorCode` enum).

Method Signature:

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

Parameters:

Parameter	Type	Required	Description
firmwareData	ByteArray	Yes	Firmware file binary data
progressBlock	(Int) -> Unit	Yes	Progress callback, percent=0~100

Return Value: `Result<UXPErrorCode>` Upgrade result.

`Result.Success (UXPErrorCode.Success)` on success; on failure the error code is in `PrinterError.errorCode` (corresponds to `UXPErrorCode.value`).

Upgrade Flow: 1) send the `enter_ota` command to enter OTA mode; 2) query `pack_length` to get the max bytes per packet; 3) send firmware data in chunks. Each packet contains offset, length, CRC32 and data. The printer replies with a status byte (0x00 OK / 0x01 LENGTH_ERROR / 0x02 CRC_ERROR / 0x03 TIME_OUT / 0x04 WRITE_FAIL / 0x05 FAIL), which is mapped to `OtaStatus*` error codes.

Example Code:

```
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 == UXPErrorCode.Success) {
            Log.d("Printer", "OTA upgrade completed")
        }
    }
    is Result.Failure -> {
        Log.e("Printer", "Error: ${result.error.message},
code=${result.error.errorCode}")
    }
}
```

15. Appendix

15.1 Paper Type Enum (PaperType)

Value	Description
CONTINUOUS	Continuous paper
LABEL	Label paper
BLACK_MARK_FRONT_2INCH	2-inch front black mark
BLACK_MARK_FRONT_3INCH	3-inch front black mark
BLACK_MARK_BACK_2INCH	2-inch back black mark
BLACK_MARK_BACK_3INCH	3-inch back black mark

15.2 WiFi Mode Enum (WifiMode)

Value	Description
AP	Access Point mode
STATION	Station mode

15.3 Printer Language Enum (PrinterLanguage)

Value	Description
ENGLISH	English
CHINESE_SIMPLIFIED	Simplified Chinese
CHINESE_TRADITIONAL	Traditional Chinese

15.4 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.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}")
                // Try to reconnect
            }
        }
    }
}
```

```

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

```

15.5 Error Code Enum (UXPErrorCode)

`UXPErrorCode` covers all `PrinterError` error types and OTA upgrade errors. When an API returns `Result.Failure`, the error code is available in `PrinterError.errorCode` (corresponds to `UXPErrorCode.value`). See the `UXPErrorCode Error Code Documentation` (Guide folder) for the full `UXPErrorCode` table.

15.6 WiFi Hotspot Info (WifiHotspotInfo)

Field	Type	Description	Example
index	String	Hotspot index, format n/m (n = current number, m = total)	1/5
name	String	Hotspot name (SSID)	Office_WiFi
security	String	Security protocol	WPA2-PSK
encryption	String	Encryption method	AES
signalStrength	String	Signal strength	-45

15.7 WiFi Encryption Enum (WifiEncryption)

Value	Firmware Value	Description
OPEN	0	No encryption
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	Unknown