EPSON
EXCEED YOUR VISION

ePOS-Print SDK for Android
ユーザーズマニュアル

概要

特徴および環境について説明します。

サンプルプログラム

サンプルプログラムの使い方について説明します。

プログラミングガイド

アプリケーション開発のプログラミング方法
について説明します。

API リファレンス

ePOS-Print SDK for Androidで提供しているAPI
について説明します。

コマンドの送受信

コマンドを送受信するためのAPIについて説明します。

付録

ePOS-Print SDK for Androidで使用するプリンター
仕様について説明します。

M00049011
Rev.L

ご注意

- **本書の内容の一部または全部を無断で転載、複写、複製、改ざんすることは固くお断りします。**
- **本書の内容については、予告なしに変更することがあります。最新の情報はお問い合わせください。**
- **本書の内容については、万全を期して作成いたしましたが、万一ご不審な点や誤り、記載もれなど、お気づきの点がありましたらご連絡ください。**
- **運用した結果の影響については、上項にかかわらず責任を負いかねますのでご了承ください。**
- **本製品がお客様により不適切に使用されたり、本書の内容に従わずに取り扱われたり、またはエプソンおよびエプソン指定の者以外の第三者により修理・変更されたことなどに起因して生じた損害などにつきましては、責任を負いかねますのでご了承ください。**
- **エプソン純正品およびエプソン品質認定品以外のオプションまたは消耗品を装着してトラブルが発生した場合には、責任を負いかねますのでご了承ください。**

商標について

EPSON、EXCEED YOUR VISION および ESC/POS はセイコーエプソン株式会社の登録商標です。

Android™ は、Google Inc. の商標または登録商標です。

Java™ は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。

Wi-Fi® は、Wi-Fi Alliance® の登録商標です。

Bluetooth® のワードマークおよびロゴは、Bluetooth SIG, Inc. が所有する登録商標であり、セイコーエプソン株式会社はこれらのマークをライセンスに基づいて使用しています。

Eclipse® は、Eclipse Foundation, Inc. の商標または登録商標です。

QR コードは株式会社デンソーウェーブの登録商標です。

その他の製品名および会社名は、各社の商標または登録商標です。

ESC/POS® コマンドシステム

エプソンは、独自の POS プリンターコマンドシステム、ESC/POS により業界のイニシアティブをとってきました。ESC/POS は特許取得済及び特許出願中のものを含む数多くの独自のコマンドを持ち、高い拡張性で多才な POS システムの構築を実現します。エプソン POS プリンターとディスプレイの全タイプに互換性を持つほか、この独自の制御システムにはフレキシビリティもあるため、将来アップグレードが行いやすくなります。その機能と利便性は世界中で評価されています。

安全のために

記号の意味

本書では以下の記号が使われています。それぞれの記号の意味をよく理解してから製品を取り扱ってください。

	ご使用上、必ずお守りいただきたいことを記載しています。この表示を無視して誤った取り扱いをすると、製品の故障や動作不良の原因になる可能性があります。
	補足説明や知っておいていただきたいことを記載しています。

使用制限

本製品を航空機・列車・船舶・自動車などの運行に直接関わる装置・防災防犯装置・各種安全装置など機能・精度などにおいて高い信頼性・安全性が必要とされる用途に使用される場合は、これらのシステム全体の信頼性および安全維持のためにフェールセーフ設計や冗長設計の措置を講じるなど、システム全体の安全設計にご配慮いただいた上で当社製品をご使用いただくようお願いいたします。

本製品は、航空宇宙機器、幹線通信機器、原子力制御機器、医療機器など、きわめて高い信頼性・安全性が必要とされる用途への使用を意図しておりませんので、これらの用途には本製品の適合性をお客様において十分ご確認の上、ご判断ください。

本書について

本書の目的

本書は、ePOS-Print SDK を使った、印刷システムの構築、設計またはプリンターアプリケーションの開発、設計に必要なすべての情報を開発技術者に提供することを、その目的としています。

本書の構成

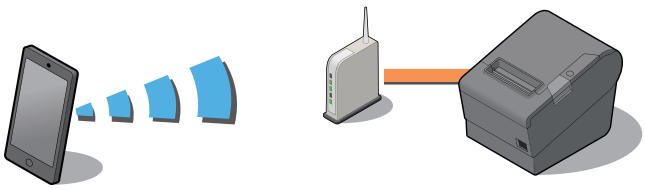
本書は次のように構成されています。

もくじ

概要

本章では、ePOS-Print SDK for Android の特徴および仕様について説明しています。

ePOS-Print SDK の概要



ePOS-Print SDK for Android は、エプソン TM プリンターに印刷するための Android アプリケーションを開発する、開発者向け SDK です。ePOS-Print SDK で提供する API を使用してアプリケーションを開発します。
ePOS-Print SDK には、iOS アプリケーション向けの ePOS-Print SDK for iOS も用意されています。

> TM プリンターにコマンドを送受信するための API も用意されています。
> コマンド送受信 API は、ePOS-Print API の Print クラスと同時に使用できません。コマンド送受信 API の詳細は 151 ページ「コマンドの送受信」を参照してください。

特徴

- ❏ Android アプリケーションから、TM プリンターに印刷できます。
- ❏ Android アプリケーションから、TM プリンターのステータスを取得できます。

機能

ePOS-Print API

- ❏ 印字の設定（位置揃え / 改行量 / 倒立印字 / ページモード）
- ❏ 文字データの設定（言語 / フォント（デバイスフォント）/ 倍角 / 倍率 / スムージング / 印字位置）
- ❏ 文字書式の設定（白黒反転 / アンダーライン / 太字）
- ❏ 紙送り設定（ドット単位 / 行単位）
- ❏ 画像の印字（ラスターイメージ /NV グラフィック）
- ❏ バーコードの印字
  （機種ごと印字できるバーコードは、162 ページ「プリンター別サポート情報」を参照してください。）
- ❏ 2 次元シンボルの印字
  （機種ごと印字できる 2 次元シンボルは、162 ページ「プリンター別サポート情報」を参照してください。）
- ❏ ドロアーキック機能
- ❏ ブザー機能
- ❏ 用紙レイアウトの設定
- ❏ ラベル / ブラックマーク紙の紙送り設定
- ❏ ESC/POS コマンドの送信
- ❏ プリンターからの応答を取得（印字結果 / プリンターの状態 / バッテリーの状態）

プリンター検索 API

- ❏ プリンターの検索

プリンター簡単選択 API

- ❏ プリンターの簡単選択
  (NFC と QR コードを使ってプリンターを簡単選択できます。)

ログ設定 API

- ❏ ログ出力の設定
  (Android 端末のストレージや、TCP 接続できるサーバーに出力できます。)

> Android 端末に出力したログは、USB 接続で他のコンピューターにも保存できます。

動作環境

Android バージョン

- ❏ Android Version 2.3.3 ～ 2.3.7
- ❏ Android Version 3.1 ～ 3.2.2
- ❏ Android Version 4.0 ～ 4.4
- ❏ Android Version 5.0 ～ 5.1

- USB のサポートは Android Version 3.1 以上です。
- 最新のバージョンは、README ファイルを参照してください。

Android デバイス

ARMv5TE 対応デバイス

プリンター

TM プリンター	インターフェイス			
	有線 LAN	無線 LAN	*Bluetooth*®	USB
TM-P20	-	○	○	○
TM-P60II	-	○	○	○
TM-T20II	-	-	○	○
TM-T70	○	○	-	○
TM-T70II	○	○	○	○
TM-T88V	○	○	○	○
TM-T90II	○	○	-	○

TM プリンター設定で、Busy となる条件は受信バッファーフルのみに設定してください。
設定についてはプリンターの詳細取扱説明書を参照してください。

開発環境

Android アプリケーションを開発するには、以下が必要です。

- Android SDK r15 以降
- Java Development Kit 6 以降

提供物

パッケージ

ファイル名	説明
ePOS-Print.jar	API を Java プログラムから利用するための jar 形式ファイルにアーカイブされたコンパイル済み Java のクラスファイルです。
ePOSEasySelect.jar	簡単にプリンターを選択するための Java のクラスファイルです。
libeposprint.so	機能実行用ライブラリーです。(ARMv5TE に対応)
ePOS-Print_Sample_Android.zip	サンプルプログラムファイルです。
README.jp.txt	README ファイルです。
README.en.txt	README ファイルです (英語版) です。
EULA.jp.txt	SOFTWARE LICENSE AGREEMENT が記載されています。
EULA.en.txt	SOFTWARE LICENSE AGREEMENT(英語版) が記載されています。
ePOS-Print_SDK_Android_ja_revx.pdf	本書です。
ePOS-Print_SDK_Android_en_revx.pdf	本書 (英語版) です。
ePOS-Print_SDK_Android_ AppDevGuide_ja_revx.pdf	開発環境を構築する手順が記載されています。
ePOS-Print_SDK_Android_ AppDevGuide_en_revx.pdf	開発環境を構築する手順が記載されています。(英語版)

マニュアル

ePOS-Print SDK for Android のマニュアルには以下のものがあります。

❏ ePOS-Print SDK for Android ユーザーズマニュアル (本書)

❏ ePOS-Print SDK for Android アプリケーション開発環境 - セットアップガイド

サンプルプログラム

ePOS-Print SDK for Android を使用した、TM プリンター用 Android アプリケーションのサンプルプログラムがあります。

❏ ePOS-Print_Sample_Android.zip

- 基本機能用サンプル (ePOSPrintSample)
- 簡単選択用サンプル (ePOSEasySelectSample)

ダウンロード

提供物は、下記エプソン販売ホームページからダウンロードできます。

http://www.epson.jp/support/sd/

制限事項

- ePOS-Print APIの通信API(49ページ)とコマンド送受信API(155ページ)は、同一デバイスに対して同時に使用することはできません。
- 同じアプリケーション内で同時にオープンできるデバイスポート数は 16 個です。
- 画面の回転時に Activity が破棄されることがあります。Print インスタンスを Activity で保持する場合、Activity が破棄される前に Print クラスの closePrinter を呼び出す必要があます。
- *Bluetooth* 接続でプリンターとの通信中に、端末がスリープ状態になると、通信が切断されてしまいます。

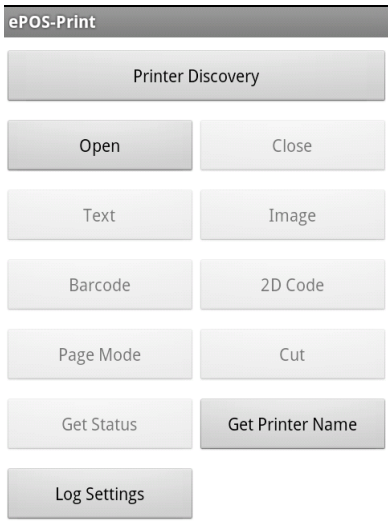
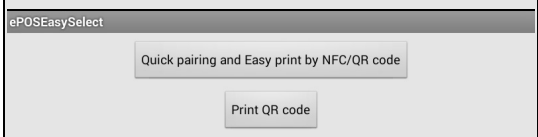
サンプルプログラム

本章では、サンプルプログラムの使い方について説明しています。

- サンプルプログラムは Android アプリケーション開発者向けの、ePOS-Print SDK for Android を使用した Android アプリケーションの実装サンプルとして提供します。
- サンプルプログラムは、Java ソースファイルを含む Eclipse 用 Android アプリケーション プロジェクトとして提供します。

概要

提供しているサンプルプログラムには、以下の機能があります。

サンプルプログラム	説明
<ePOSPrintSample> 	サンプルプログラムは、以下の機能を実装しています。 (サンプルプログラムでは、文字 / 画像 / バーコードなどを回転させる機能は、実装していません。) • プリンターの検索 • ポートオープン • ポートクローズ • テキスト印刷 • グラフィック印刷 (画像ファイルの印刷) • バーコード印刷 • 2 次元シンボル印刷 • ページモード印刷 • 用紙カット • プリンターのステータス取得 • プリンターの機種名 / 言語情報取得 • ログ出力設定 • ステータスイベントの表示 • バッテリーステータスイベントの表示
<ePOSEasySelectSample> 	NFC/ QR コードを使用して簡単にプリンターに接続します。 • NFC からプリンター情報の取得 • QR コードからプリンター情報の取得 • 取得したプリンター情報の解析 • 解析結果を用いたポートオープン • プリンターの検索結果からプリンター情報の QR コード作成

使用環境

開発環境

- Android SDK r16
- Java Development Kit 6
- Eclipse
- ADT Plugin for Eclipse

開発環境構築の詳細は、「ePOS-Print SDK for Android アプリケーション開発環境 - セットアップガイド」を参照してください。

プリンター

- ePOS-Print SDK でサポートしている TM プリンター

ターゲットデバイス

- コンピューターに USB 接続されているデバイス

環境構築

以下の手順でサンプルプログラムの環境を構築します。

1 サンプルプログラムの ZIP ファイルを任意のディレクトリーに展開します。

2 Eclipseの[File]-[Import] で[General]-[Existing Project into Workspace]を選択し、[Next] をクリックします。

3 「Import Projects」画面が表示されます。以下の設定をし、[Finish] をクリックします。

項目	設定
Select root directory	サンプルプログラムの ZIP ファイルを展開したディレクトリーを指定する
Copy projects into workspace	チェックを付ける

4 「Package Explorer」View で、″ePOSPrintSample″ プロジェクト、または ″ePOSEasySelectSample″ プロジェクトを右クリックし、[Properties] を選択します。

5 「Properties for ePOSPrintSample」、または「Properties for Properties for ePOSEasySelectSample」が表示されます。以下の設定をし、[OK] をクリックします。

項目	設定
Android	ターゲットデバイスの Android OS バージョンを選択する
Java Build Path	Libraries に、プロジェクトワークスペースにコピーしたプロジェクト内の ePOS-Print.jar、または ePOS-Pairing.jar へのパスが設定されていることを確認する

6 「Package Explorer」View で、″ePOSPrintSample ″、または ″ePOSEasySelectSample ″ プロジェクトを右クリックし、[Run As 、Android Application] を選択します。

7 ターゲットデバイスにサンプルプログラムがインストールされ、サンプルプログラムが起動します。

使用方法

ここでは、以下の使い方を説明します。

❏ ePOSPrintSample

- プリンターを検索して印刷する (16 ページ)
- プリンターの機種名を取得する (23 ページ)

❏ ePOSEasySelectSample

- NFC/ QR コードを使ってプリンターを選択する (24 ページ)
- QR コードを印刷する (24 ページ)

プリンターを検索して印刷する

以下の手順で使用します。

1 サンプルプログラムを起動します。詳細は、環境構築 (15 ページ)を参照してください。

2 プリンターを検索します。メイン画面で [Printer Discovery] をタップします。
[Device Type] を選択すると、[Printer List] に検索されたプリンターの IP アドレス / Mac アドレス / デバイスノード / プリンター名がリスト表示されます。

3 表示された [Printer List] の中から、使用するプリンターをタップします。

4 プリンターのポートをオープンします。メイン画面で [Open] をタップします。
手順 3 で選択したプリンターの ″Device Type″ と ″IP アドレス / Mac アドレス / デバイスノード ″ が表示されます。[Printer Name] と [Language] を選択します。

5 [Status Monitor] を設定します。

項目	説明
Enabled	• ON: ステータスモニターを有効にし、プリンターステータスの監視を行います。 • OFF:ステータスモニターを無効にします。
Interval	Enabled を ON に設定した場合、ステータスの監視間隔をミリ秒単位で設定します。

6 [Open] をタップします。

7 以下の処理を実行します。

処理	説明
テキスト印刷	メイン画面の (Text) をタップしてください。 詳細はテキスト印刷 (17 ページ) を参照してください。
グラフィック印刷	メイン画面の (Image) をタップしてください。 詳細はグラフィック印刷 (18 ページ) を参照してください。
バーコード印刷	メイン画面の (Barcode) をタップしてください。 詳細はバーコード印刷 (18 ページ) を参照してください。
2 次元シンボル印刷	メイン画面の (2D Code) をタップしてください。 詳細は 2 次元シンボル印刷 (19 ページ) を参照してください。
ページモード印刷	メイン画面の (Page Mode) をタップしてください。 詳細はページモード印刷 (19 ページ) を参照してください。
用紙カット	メイン画面の (Cut) をタップしてください。 詳細は用紙カット (19 ページ) を参照してください。
プリンターステータスの取得	メイン画面の (Get Status) をタップしてください。
ログ出力設定	メイン画面の (Log Settings) をタップしてください。 詳細はログ出力設定 (20 ページ) を参照してください。

8 以下の処理の実行結果が表示されます。

- 処理実行結果 (エラーステータス / プリンターステータス / バッテリーステータス)
  詳細は、処理実行結果 (21 ページ) を参照してください。
- メソッド (API) 実行エラー
  詳細は、メソッド (API) 実行エラー (22 ページ) を参照してください。

9 処理をすべて終えたら、メイン画面の [Close] をタップし、プリンターのポートをクローズします。

テキスト印刷

以下の手順で実行します。

1 [Print Characters] に印字文字列を入力します。

2 印字文字列に文字の属性を指定します。以下の属性を指定できます。

属性	説明
Font	文字フォントを設定します。
Align	位置揃えを設定します。
Line Spacing	改行量を設定します。
Language	言語を設定します。
Size	文字の倍率 (縦倍 / 横倍) を設定します。
Style	文字修飾 (ボールド / アンダーライン) を設定します。
X Position	横位置の開始位置を設定します。
Feed Unit	紙送り量を設定します。

3 [Print] をタップし、印刷します。

グラフィック印刷

以下の手順で実行します。

1 [Select Image] をタップし、印刷する画像ファイルを選択します。

2 [Color Mode] をタップし、階調を選択します。

> (Gray 16) (多階調印刷) を選択できる機種は、TM-T70II/ TM-T88V/ TM-T90II のみです。

3 [Halftone Method] をタップし、ハーフトーンの処理方法を選択します。

4 [Brightness] をタップし、数値を入力して明るさを指定します。

5 [Print] をタップし、印刷します。

バーコード印刷

以下の手順で実行します。

1 以下のバーコードの設定をします。

設定	説明
Type	バーコードの種類を選択します。
Data	バーコードデータを入力します。
HRI	HRI の位置を設定します。
Font	HRI フォントを設定します。
Module Size(Width, Height)	バーコードのモジュールサイズ (幅 / 高さ) を設定します。

2 [Print] をタップし、印刷します。

2 次元シンボル印刷

以下の手順で実行します。

1 [Type] から 2 次元シンボルの種類を選択します。

2 [Data] に 2 次元シンボルデータを入力します。

3 2 次元シンボルの種類ごとに、以下を設定します。

設定	説明
Error Correction Level (PDF417,QR Code, Aztec Code, DataMatrix)	エラー訂正レベルを設定します。
Module Size(Width, Height)	2 次元シンボルのモジュールサイズ (幅 / 高さ) を設定します。
Max Size	2 次元シンボルの最大サイズを設定します。

4 [Print] をタップし、印刷します。

ページモード印刷

以下の手順で実行します。

1 [Print Characters] に印字文字列を入力します。

2 [Print Area] で印字領域の設定をします。

設定	説明
X	横方向の原点を設定します。
Y	縦方向の原点を設定します。
Width	印字領域の幅を設定します。
Height	印字領域の高さを設定します。

3 [Print] をタップし、印刷します。

用紙カット

以下の手順で実行します。

1 [Type] で紙送りしてカットするか設定します。

2 [Print] をタップし、カットを実行します。

ログ出力設定

以下の手順で設定します。

1 [Enabled] に、ログ出力の有無と、ログの出力先を設定します。

2 ログの出力先に合わせて、以下を設定します。

設定	説明
IP Address	TCP 通信の IP アドレスを指定します。
Port	TCP 通信用のポート番号を指定します。
Log Size	端末のストレージに保存する、ログの最大容量を指定します。
Log Level	出力するログのレベルを設定します。

3 [Save Settings Permanently] に、設定の保存方法を設定します。

4 [Setting] をタップし、ログ出力の設定を有効にします。

5 印刷実行後、ログファイルを確認します。

詳細は、setLogSettings（147 ページ）を参照してください。

実行結果

処理実行結果

以下の表示がされます。

- Result:　　以下のエラーステータスが表示されます。

表示文字列	説明
SUCCESS	成功
ERR_PARAM	不正なパラメーターが渡された
ERR_ILLEGAL	不適切な方法で使用された
ERR_PROCESSING	処理を実行できなかった
ERR_TIMEOUT	処理がタイムアウトされた
ERR_CONNECT	デバイスとの通信に失敗した
ERR_MEMORY	処理に必要なメモリーが確保できなかった
ERR_OFF_LINE	オフライン状態
ERR_FAILURE	その他のエラーが発生した

- Status:　　以下のプリンターステータスが表示されます。

表示文字列	説明
NO_RESPONSE	プリンター無応答
PRINT_SUCCESS	印刷終了
DRAWER_KICK	ドロアーキックコネクター 3 番ピンの状態＝ "H" (TM-P60II 以外)
BATTERY_OFFLINE	バッテリーオフライン状態 (TM-P60II)
OFF_LINE	オフライン状態
COVER_OPEN	カバーが開いている
PAPER_FEED	紙送りスイッチによる紙送り中
WAIT_ON_LINE	オンライン復帰待ち中
PANEL_SWITCH	紙送りスイッチが押されている
MECHANICAL_ERR	メカニカルエラー発生
AUTOCUTTER_ERR	オートカッターエラー発生
UNRECOVER_ERR	復帰不可能エラー発生
AUTORECOVER_ERR	自動復帰エラー発生
RECEIPT_NEAR_END	ロール紙ニアエンド検出器に用紙なし
RECEIPT_END	ロール紙エンド検出器に用紙なし
BUZZER	ブザーが鳴っている (対応機器のみ)

- Battery Status:　　以下が表示されます。

表示文字列	説明
0xnnnn	バッテリーステータス値 詳細は、バッテリーステータス (46 ページ) を参照してください。

メソッド (API) 実行エラー

以下の表示がされます。

- Error Code:　以下のエラーステータスが表示されます。

表示文字列	説明
ERR_PARAM	不正なパラメーターが渡された
ERR_OPEN	オープン処理に失敗した
ERR_CONNECT	デバイスとの通信に失敗した
ERR_TIMEOUT	指定された時間内にすべてのデータを送信できなかった
ERR_MEMORY	処理に必要なメモリーが確保できなかった
ERR_ILLEGAL	不適切な方法で使用された
ERR_PROCESSING	処理を実行できなかった
ERR_UNSUPPORTED	サポートしていない機種名または言語使用が指定された
ERR_OFF_LINE	プリンターがオフライン状態だった
ERR_FAILURE	その他のエラーが発生した

- Method:　　メソッド実行エラーになった API が表示されます。

プリンターの機種名を取得する

プリンターの機種名の取得は、コマンド送受信 API を使用しています。詳細は、コマンドの送受信 (151 ページ) を参照してください。

以下の手順で使用します。

1. サンプルプログラムを起動します。詳細は、環境構築（15 ページ）を参照してください。
2. プリンターを検索します。メイン画面で [Printer Discovery] をタップします。
   [Device Type] を選択すると、[Printer List] に検索されたプリンターの IP アドレス / Mac アドレス / デバイスノード / プリンター名がリスト表示されます。
3. 表示された [Printer List] の中から、使用するプリンターをタップします。
4. メイン画面の [Get Printer Name] をタップします。
5. [Get Printer Name] をタップします。
6. 以下が表示されます。

表示内容	説明
Printer Name	プリンターの機種名が表示されます。
Language	プリンターの言語仕様が表示されます。

NFC/ QR コードを使ってプリンターを選択する

以下の手順で使用します。

1 サンプルプログラムを起動します。詳細は、環境構築（15 ページ）を参照してください。

2 メイン画面の [Quick pairing and Easy print by NFC/QR code] をタップします。

3 以下の方法でプリンターを選択します。
- NFC にスマートデバイスをかざします。
- QR コードをカメラで読み取ります。
  カメラプレビューの赤枠内に QR コードを入れてください。

4 [Print] をタップし、印刷します。

QR コードを印刷する

以下の手順で使用します。

1 サンプルプログラムを起動します。詳細は、環境構築（15 ページ）を参照してください。

2 メイン画面の [Print QR code] をタップします。

3 QR コード印刷用画面の [Find] をタップします。
[Printer List] に検索されたプリンターがリスト表示されます。

4 使用したいプリンターを選択します。

5 [Print] をタップし、印刷します。

プログラミングガイド

本章では、ePOS-Print SDK を使用したアプリケーション開発のプログラミング方法について説明します。

> ePOS-Print SDK for Android を使用した、Android アプリケーション開発環境の構築方法については、「ePOS-Print SDK for Android アプリケーション開発環境 – セットアップガイド」を参照してください。

ePOS-Print SDK for Android の組み込み方法

ePOS-Print SDK for Android の組み込み方法について説明します。

> Eclipse で説明しています。その他の環境で開発する場合、読み替えてください。

以下の手順で組み込んでください。

1. Eclipse で新しいプロジェクトを作成します。
2. 提供された jar 形式ファイル (ePOS-Print.jar, ePOSEasySelect.jar) を、次のパスにコピーします。

> ePOSEasySelect.jar は、プリンター簡単選択を使用しない場合、必要ありません。

3. 対象プロジェクトのプロパティの [Libraries] タグに、追加した Jar ファイル (ePOS-Print.jar) が [Java Build Path] に登録されていることを確認します。
追加されていない場合、[Add Jars...] で追加した Jar ファイルを、ビルドパスに追加します。
4. ライブラリーファイル (libeposprint.so) を、次のパスにコピーします。

5. Eclipse の [Package Explorer] でプロジェクトを選択し、右クリックで [Refresh] を押します。

6 使用したいアプリケーションの *.java ソースファイルで、パッケージのインポート定義を記載します。下記を参照してください。

```
import com.epson.eposprint.*;
import com.epson.epsonio.*;
import com.epson.epos.easyselect.*;
```

プリンター簡単選択を使用しない場合、easyselect パッケージを定義する必要はありません。

7 対象プロジェクトのプロパティの [Source] タグに、対象プロジェクト ″/libs″ があることを確認します。確認できない場合、[Add Folder...] から ″libs″ をビルドパスに追加します。

8 Eclipse の (Package Explorer) から、対象のプロジェクトが選択された状態で、[Window] メニューの [Preferences] を選択します。

9 [Preferences] 画面が表示されます。左のリストから [Java]-[Compiler] を選択します。

10 [Compiler] 画面が表示されます。[Compiler compliance level:] を ″1.6″ に設定し、[Apply] をクリックします。その後、[OK] をクリックします。

11 Eclipse の [Package Explorer] の ″AndroidManifest.xml″ をダブルクリックします。

12 [Permissions] タグを選択します。

13 [Android Manifest Permissions] 画面が表示されます。[Add] ボタンをクリックします。

14 [Uses Permission] を選択し、[OK] ボタンをクリックします。

15 [Permissions] に [Uses Permission] が追加されます。追加された [Uses Permission] に付加する機能の権限を [Attributes for Uses Permission] の [Name] から選択します。

機能	[Name] の設定
Wi-Fi®	android.permission.INTERNET
Bluetooth	android.permission.*BLUETOOTH*
	android.permission.*BLUETOOTH*_ADMIN
USB	android.permission.USB

付加できる機能の権限は、[Permissions] の [Uses Permission] に対して 1 つの設定です。*Bluetooth* 機能およびすべて機能を使用するためには、手順 13 ～手順 15 までを繰り返して設定する必要があります。

16 ″AndroidManifest.xml″ を保存します。

ePOS-Print SDK

印刷モードについて

印字モードにはスタンダードモードとページモードがあります。

スタンダードモード

スタンダードモードとは、1 行単位で印字する印字モードです。文字サイズ、画像、バーコードなどの高さに合わせて改行量が調整されます。レシートなど、印字量に合わせて用紙の長さが変化する印刷に向いています。

ページモード

ページモードとは、印字領域を設定して印字データを展開し、一括印字する印字モードです。印字位置（座標）に、文字、画像、バーコードなどを展開します。

プログラミングフロー

以下のフローでプログラミングします。

1. 事前準備 (USB 接続) (28 ページ) *

2. プリンターの選択 (29 ページ) *

< 検索機能を使ってプリンターを選択 >
- 検索機能を使ってプリンターを選択 (29 ページ)
- プリンターの検索結果を取得 (29 ページ)
- プリンターの検索を終了 (30 ページ)

<NFC を使ってプリンターを選択 >
- NFC を使ってプリンターを選択 (30 ページ)

<QR コードを使ってプリンターを選択 >
- QR コードを使ってプリンターを選択 (31 ページ)

3. 印刷ドキュメントの作成 (32 ページ)
- テキストの印刷ドキュメントを作成する場合 (32 ページ)
- グラフィックの印刷ドキュメントを作成する場合 (33 ページ)
- ページモードの印刷ドキュメントを作成する場合 (33 ページ)

4. 印刷ドキュメントの送信 (34 ページ)

*: 任意のプロセスです。

あらかじめプリンターの状態を確認して送信するプログラミングをすると、確実に印刷できます。方法については、プリンターの状態を確認してから印刷する (36 ページ) を参照してください。

事前準備（USB 接続）

インターフェイスが USB の場合、あらかじめにアプリケーションで USB デバイスへのアクセス許可を取得することを推奨します。

あらかじめ USB デバイスへのアクセス許可を取得せず、openPrinter メソッドでポートオープンする場合、以下の注意点があります。

- アクセス許可取得のダイアログで[OK]を押すと、ポート オープンまでに10秒前後の時間が掛かります。
- アクセス許可取得のダイアログで[キャンセル]を押すと、30 秒のタイムアウト 待ちになります。

アプリケーションでアクセス許可を取得する方法は以下のとおりです。

1 AndroidManifest.xml に以下のコードを追記します。

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

2 ソースファイルに res/xml/device_filter.xml を追加します。

3 device_filter.xml ファイルに以下のコードを記述します。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device vendor-id="1208" />
</resources>
```

4 アクセス許可を取得する際、ダイアログが表示されます。[OK] を押します。

プリンターの選択

検索機能を使ってプリンターを選択

Finder クラスの start（137 ページ）を使って、プリンターの検索を開始します。以下のプログラミングを参考にしてください。

```
int errStatus = IoStatus.SUCCESS;
int deviceType = Print.DEVTYPE_TCP;

// 検索開始
try {

//Wi-Fi/Ethernet デバイスの場合
    switch(deviceType){
    case DevType.TCP:
        Finder.start(getBaseContext(), DevType.TCP, "255.255.255.255");
        break;
//Bluetooth デバイスの場合
    case DevType.BLUETOOTH:
        Finder.start(getBaseContext(), DevType.BLUETOOTH, "null");
        break;
//USB デバイスの場合
    case DevType.USB:
        Finder.start(getBaseContext(), DevType.USB, "null");
        break;
    default:
        Finder.start(getBaseContext(), DevType.TCP, "255.255.255.255");
        break;
    }

// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

プリンターの検索結果を取得

Finder クラスの getDeviceinfolist（139 ページ）を使って、プリンターの検索結果を取得します。
以下のプログラミングを参考にしてください。取得した結果を openPrinter（107 ページ）で使用してください。

```
int errStatus = IoStatus.SUCCESS;
DeviceInfo[] mList = null;

// デバイス一覧の取得
try {
    mList = getDeviceInfoList(FilterOption.PARAM_DEFAULT);
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

プリンターの検索に時間がかかるため、Finder クラスの start を呼んだ直後に getDeviceInfoList を呼んだ場合、検索結果が無いこともあります。

プリンターの検索を終了

Finder クラスの stop（138 ページ）を使って、プリンターの検索を終了します。以下のプログラミングを参考にしてください。

```
int errStatus = IoStatus.SUCCESS;

// 検索終了
try {
    Finder.stop();
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

NFC を使ってプリンターを選択

EasySelect クラスの parseNFC（144 ページ）を使って、NFC タグを解析します。
以下のプログラミングを参考にしてください。取得した結果を openPrinter（107 ページ）で使用してください。

```
@Override
protected void onNewIntent(Intent intent) {

    //NFC タグは、onNewIntent で受け取る

    EasySelect easySelect = new EasySelect();

    Tag tag = (Tag)intent.getParcelableExtra( NfcAdapter.EXTRA_TAG );

    //NFC タグの解析
    EasySelectInfo easySelectInfo = easySelect.parseNFC( tag );
    if (null == easySelectInfo) {
        // 簡単選択用の NFC ではなかった場合
        return ;
    }

    try {
        Print printer = new Print();

        // 解析したデータを使って、プリンターをオープン
        printer.openPrinter(esaySelectInfo.deviceType, easySelect.printerName,
                            easySelectInfo.macAddress );

        //・・・印刷処理・・・

        // プリンターをクローズ
        printer.closePrinter();

    // 例外処理
    } catch (EposException e) {
        //・・・処理・・・
    }
}
```

本機能は、NFC 対応機種のみ使用できます。

QR コードを使ってプリンターを選択

EasySelect クラスの parseQR（144 ページ）を使って、QR コードを解析します。
以下のプログラミングを参考にしてください。取得した結果を openPrinter（107 ページ）で使用してください。

```
EasySelect easySelect = new EasySelect();
String data;

// カメラ画像から取得した QR コードデータを格納

//QR コードの解析
EasySelectInfo easySelectInfo = easySelect.parseQR(data);
if (null == easySelectInfo) {
    // 簡単選択用の QR コードでは無かった場合
    return ;
}

try {
    Print printer = new Print();

    // 解析したデータを使って、プリンターをオープン
    printer.openPrinter(easySelectInfo.deviceType, easySelectInfo.macAddress);

    // 解析したデータを使って印刷ドキュメントを作成
    Builder builder = new Builder(easySelectInfo.printerName, Builder.MODEL_JAPANESE);

    printer.closePrinter();

// 例外処理
} catch (EposException e) {
    // 例外処理
}
```

プリンター簡単選択用の QR コードを作成する方法

❏ プリンター簡単選択用の QR コードを自動印刷できる機種の場合

ダイナミックステータスシートの QR コードを使用してください。
ダイナミックステータスシートの印刷方法は機種ごとの Technical Reference Guide を参照してください。

❏ プリンター簡単選択用の QR コードを自動印刷できない機種の場合

createQR（145 ページ）を使用して QR コードを作成してください。
サンプルプログラムの QR コードを作成するを参照してください。

印刷ドキュメントの作成

印刷ドキュメントは、Builder クラス（47 ページ）で作成します。
コンストラクターで Builder クラスを作成し、Builder クラスの各 API で印刷ドキュメントを作成します。
以下のプログラミングを参考にしてください。

```
try {
    //Builder クラスのインスタンスを初期化
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    // 印刷ドキュメントの作成
    builder.addTextLang(Builder.LANG_JA);
    builder.addTextSmooth(Builder.TRUE);
    builder.addTextFont(Builder.FONT_A);
    builder.addTextSize(3, 3);
    builder.addText("Hello,\t");
    builder.addText("World!\n");
    builder.addCut(Builder.CUT_FEED);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

テキストの印刷ドキュメントを作成する場合

テキストの印刷ドキュメントを作成する場合、テキストの各 API でフォントの設定を命令バッファーに格納し、印刷ドキュメントを作成します。以下のプログラミングを参考にしてください。

文字列「Hello, World!」を以下の設定で印刷ドキュメントを作成する場合

- フォント：FontA
- 倍率：　幅４倍、高さ４倍
- スタイル：太字

```
try {
    //Builder クラスのインスタンスを初期化
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);

    // 印刷ドキュメントの作成
    //＜印字文字の設定＞
    builder.addTextLang(Builder.LANG_JA);
    builder.addTextSmooth(Builder.TRUE);
    builder.addTextFont(Builder.FONT_A);
    builder.addTextSize(4, 4);
    builder.addTextStyle(Builder.FALSE, Builder.FALSE, Builder.TRUE, Builder.PARAM_UNSPECIFIED);

    //＜印刷データを指定＞
    builder.addText("Hello,\t");
    builder.addText("World!\n");
    builder.addCut(Builder.CUT_FEED);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

グラフィックの印刷ドキュメントを作成する場合

グラフィックの印刷ドキュメントを作成する場合、グラフィックは、android.graphics.Bitmap クラスを Builder クラスの addImage (69 ページ) で命令バッファーに格納します。以下のプログラミングを参考にしてください。

```
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
try {
    //Builder クラスのインスタンスを初期化
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);

    // 印刷ドキュメントの作成
    Bitmap bmp = BitmapFactory.decodeResource(getResources(),R.drawable.background);
    builder.addImage(bmp, 0, 0, 8, 48, Builder.PARAM_DEFAULT);
    builder.addCut(Builder.CUT_FEED);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

グラフィック印字する方法には、プリンターの NV メモリーに登録されているグラフィックを印字する方法もあります。詳細は、addLogo (77 ページ) を参照してください。

ページモードの印刷ドキュメントを作成する場合

Builder クラスの addPageBegin (88 ページ) を命令バッファーに格納することで、ページモードが開始されます。印字領域 (addPageArea (90 ページ)) と印字開始位置 (addPagePosition (92 ページ)) を命令バッファーに格納します。
印字開始位置は、印字データに合わせて指定します。その後、各 API を命令バッファーに格納し印字データを作成します。ページモードの終了は addPageEnd (89 ページ) を命令バッファーに格納します。

文字列「Hello, World!」を以下の設定で印刷ドキュメントを作成する場合

- ページモードの印字領域 (ドット単位):
  横方向原点 : 100, 縦方向原点 : 50, 幅 :200, 高さ : 100
- ページモードの印字位置 (ドット単位):
  横方向の印字位置 : 0, 縦方の印字位置 : 42
- フォント : FontA
- 倍率 :　　幅 2 倍、高さ 2 倍
- スタイル : 太字

```
try {
    //Builder クラスのインスタンスを初期化
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);

    // 印刷ドキュメントの作成
    //< ページモード開始 >
    builder.addPageBegin();
    builder.addPageArea(100, 50, 200, 100);
    builder.addPagePosition(0, 42);
    //< 印字文字の設定 >
    builder.addTextLang(Builder.LANG_JA);
    builder.addTextSmooth(Builder.TRUE);
    builder.addTextFont(Builder.FONT_A);
    builder.addTextSize(2, 2);
    builder.addTextStyle(Builder.FALSE, Builder.FALSE, Builder.TRUE, Builder.PARAM_UNSPECIFIED);
    //< 印刷データを指定 >
    builder.addText("Hello,\t");
    builder.addText("World!\n");
    //< ページモード終了 >
    builder.addPageEnd();
    builder.addCut(Builder.CUT_FEED);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

印刷ドキュメントの送信

印刷ドキュメントは、Print クラス（49 ページ）で送信します。
コンストラクターで Print クラスを作成し、sendData で、印刷ドキュメントの命令バッファーを格納した Builder クラスのインスタンスを指定して送信します。
Builder に格納された命令バッファーは、clearCommandBuffer（54 ページ）を実行するまで保管されます。sendData（114 ページ）成功後に clearCommandBuffer を実行してください。

同じ印刷ドキュメントを繰り返し印刷したい場合、clearComanndBuffer を実行する必要はありません。

以下のプログラミングを参考にしてください。

```
//Print クラスのインスタンスを初期化
Print printer = new Print();

int[] status = new int[1];
status[0] = 0;

try {
    //Builder クラスのインスタンスを初期化
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);

    // 印刷ドキュメントの作成
    //＜印字文字の設定＞
    builder.addTextLang(Builder.LANG_JA);
    builder.addTextSmooth(Builder.TRUE);
    builder.addTextFont(Builder.FONT_A);
    builder.addTextSize(4, 4);
    builder.addTextStyle(Builder.FALSE, Builder.FALSE, Builder.TRUE, Builder.PARAM_UNSPECIFIED);

    //＜印刷データを指定＞
    builder.addText("Hello,\t");
    builder.addText("World!\n");
    builder.addCut(Builder.CUT_FEED);

    // 印刷ドキュメントを送信
    //＜プリンターとの通信を開始＞
    ////Wi-Fi/Ethernet デバイスの場合
    printer.openPrinter(mList.getDeviceType(), mList.getDeviceName());
    ////Bluetooth デバイスの場合
    printer.openPrinter(mList.getDeviceType(), mList.getDeviceName(), "null", Print.TRUE,
                        Print.PARAM_DEFAULT);
    ////USB デバイスの場合
    printer.openPrinter(mList.getDeviceType(), mList.getDeviceName(), getApplicationContext(),
                        Print.TRUE, Print.PARAM_DEFAULT);

    //＜データを送信＞
    printer.sendData(builder, 10000, status);
    //＜命令バッファーの削除＞
    if((status[0] & Print.ST_PRINT_SUCCESS) == Print.ST_PRINT_SUCCESS)
    {
        builder.clearCommandBuffer();
    }
    //＜プリンターとの通信を終了＞
    printer.closePrinter();

} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
    printer.closePrinter();
}
```

設定用の命令バッファーの有効範囲

設定用に使用される Builder クラスの addXXX の有効範囲は、addXXX 設定後、sendData が実行されるまでです。設定した値は、sendData の実行ごとに初期化されます。以下を参考にしてください。

例

```
Print printer = new Print();
Builder builder = null;

Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
builder.addText("Hello, World!\n");          addTextFont の設定が無効な文字列
builder.addTextFont(Builder.FONT_A);
builder.addText("Hello, World!\n");          addTextFont の設定が有効な文字列 (FONT_A)
printer.sendData(builder, 10000, status);
builder.addText("Hello, World!\n");          addTextFont の設定が無効な文字列
builder.addTextFont(Builder.FONT_B);
builder.addText("Hello, World!\n");          addTextFont の設定が有効な文字列 (FONT_B)
printer.sendData(builder, 10000, status);
```

プリンターの状態を確認してから印刷する

あらかじめプリンターの状態を確認してから印刷すると、確実に印刷できます。空の印刷データを送信し、プリンターがオンライン状態の場合に印刷します。

以下を参考にしてください。

```
//Print クラスのインスタンスを初期化
Print printer = new Print();

int[] status = new int[1];
status[0] = 0;
Builder builder = null;

try {
    // 印刷ドキュメントの作成                                                          ①
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addText("Hello, World!\n");
    builder.addCut(Builder.CUT_FEED);

    // 確認用 Builder クラスのインスタンスを初期化                                     ②
    Builder confirmBuilder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);

    //＜プリンターとの通信を開始＞
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");

    //＜確認用データを送信＞                                                          ③
    printer.sendData(confirmBuilder, 10000, status);

    if((status[0] & Print.ST_OFF_LINE) != Print.ST_OFF_LINE){                        ④
        //＜印刷データを送信＞
        printer.sendData(builder, 10000, status);

    } else if ((status[0] & Print.ST_OFF_LINE) == Print.ST_OFF_LINE){                ⑤
        ;
    } else {
        ;
    }

    //＜プリンターとの通信を終了＞
    printer.closePrinter();

} catch (EposException e1) {
    int errStatus1 = e1.getErrorStatus();
    status[0] = e1.getPrinterStatus();

    if ( errStatus1 == EposException.ERR_CONNECT ){                                  ⑥
        try{
            //＜プリンターとの通信を終了＞
            printer.closePrinter();

            //＜プリンターとの通信を開始＞
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");

            //＜印刷データを送信＞
            printer.sendData(builder, 10000, status);

            //＜プリンターとの通信を終了＞
            printer.closePrinter();

        } catch (EposException e2) {
            int errStatus2 = e2.getErrorStatus();
            status[0] = e2.getPrinterStatus();
        }
}
```

1 印刷データを作成します。

2 プリンターの状態を確認するために空の印刷データを作成します。

3 ②で作成した印刷データを送信します。

4 ②で作成した印刷データが正常に送信され、プリンターがオンライン状態の場合に、続けて①で作成した印刷データを送信します。

5 ②で作成した印刷データが正常に送信され、プリンターがオフライン状態の場合は、プリンターのオフラインとなる要因を取り除いてください。
（プリンターのカバーオープン、用紙切れなど）

6 ②で作成した印刷データが正常に送信されなかった場合、プリンターとの通信を終了して、再度プリンターとの通信を開始した後、印刷データを送信します。

プリンターステータスを自動で取得

ePOS-Print SDK では、リスナーを使用して、プリンターの状態の変化を自動でアプリケーションに通知できます。

以下を参考にしてください。

```
// プリンターステータスを通知する StatusChangeEventListener の登録                    ①
public class SampleActivity extends Activity implements OnClickListener,
                                                    StatusChangeEventListener {

    //・・・処理・・・

    //StatusChangeEventListener のメソッドを実装                                    ②/⑤
    private void onStatusChangeEvent(String deviceName, int status) {

        //・・・処理・・・
    }

    private void openPrinter() {
        //Print クラスのインスタンスを初期化
       Print printer = new Print();

        // プリンターの状態変化の通知先を登録                                        ③
        printer.setStatusChangeEventCallback(this);

        try {
          // プリンターとの通信、およびプリンターステータスのモニタリングを開始      ④
          printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                              Print.PARAM_DEFAULT);

          //・・・処理・・・

        } catch (EposException e) {
          int errStatus = e.getErrorStatus();
          printer.closePrinter();
        }
    }
}
```

1 プリンターステータスを取得するリスナーインターフェイスを定義します。

> 上記では、プリンターステータスを openPrinter（107 ページ）で指定した間隔で通知する StatusChangeEventListener を定義しています。
> ePOS-Print のリスナーインターフェイスには、カバーオープンやドロアーオープンのイベントといった、プリンターステータスごとのリスナーインターフェイスも用意されています。用途に応じて使い分けてください。ePOS-Print で使用できるリスナーインターフェイスは、リスナーインターフェイス一覧（39 ページ）を参照してください。

2 イベント発生時の通知先メソッドを実装します。

3 プリンターステータスの通知先を登録します。

4 openPrinter（107 ページ）を使って、プリンターステータスのモニタリングを開始します。

5 ②で実装したメソッドにプリンターステータスを通知します。

> プリンターステータスの通知を終了した場合、Print クラスの closePrinter（113 ページ）で終了します。

リスナーインターフェイス一覧

リスナーインターフェイスの詳細は、下記、通知先登録 API を説明している、API リファレンス (47 ページ) を参照してください。

機能	イベントリスナー 通知先メソッド 通知先登録 API
プリンターステータスの通知	public interface StatusChangeEventListener extends EventListener
	void onStatusChangeEvent(String deviceName, int status)
	setStatusChangeEventCallback (118 ページ)
オンラインの通知	public interface OnlineEventListener extends EventListener
	void onOnlineEvent(String deviceName)
	setOnlineEventCallback (119 ページ)
オフラインの通知	public interface OfflineEventListener extends EventListener
	void onOfflineEvent(String deviceName)
	setOfflineEventCallback (120 ページ)
無応答の通知	public interface PowerOffEventListener extends EventListener
	void onPowerOffEvent(String deviceName)
	setPowerOffEventCallback (121 ページ)
カバークローズの通知	public interface CoverOkEventListener extends EventListener
	void onCoverOkEvent(String deviceName)
	setCoverOkEventCallback (122 ページ)
カバーオープンの通知	public interface CoverOpenEventListener extends EventListener
	void onCoverOpenEvent(String deviceName)
	setCoverOpenEventCallback (123 ページ)
用紙ありの通知	public interface PaperOkEventListener extends EventListener
	void onPaperOkEvent(String deviceName)
	setPaperOkEventCallback (124 ページ)
用紙残量少の通知	public interface PaperNearEndEventListener extends EventListener
	void onPaperNearEndEvent(String deviceName)
	setPaperNearEndEventCallback (125 ページ)
用紙なしの通知	public interface PaperEndEventListener extends EventListener
	void onPaperEndEvent(String deviceName)
	setPaperEndEventCallback (126 ページ)
ドロアークローズの通知	public interface DrawerClosedEventListener extends EventListener
	void onDrawerClosedEvent(String deviceName)
	setDrawerClosedEventCallback (127 ページ)
ドロアーオープンの通知	public interface DrawerOpenEventListener extends EventListener
	void onDrawerOpenEvent(String deviceName)
	setDrawerOpenEventCallback (128 ページ)
バッテリー残量なしの通知	public interface BatteryLowEventListener extends EventListener
	void onBatteryLowEvent(String deviceName)
	setBatteryLowEventCallback (129 ページ)

機能	イベントリスナー 通知先メソッド 通知先登録 API
バッテリー残量ありの通知	public interface BatteryOkEventListener extends EventListener
	void onBatteryOkEvent(String deviceName)
	setBatteryOkEventCallback (130 ページ)
バッテリーステータスの通知	public interface BatteryStatusChangeEventListener extends EventListener
	void onBatteryStatusChangeEvent(String deviceName, int battery)
	setBatteryStatusChangeEventCallback (131 ページ)

例外処理

ePOS-Print SDK for Android では、エラー発生時に数値 (int) 型のパラメーターを持つ独自の例外を発生させ、呼び元にエラーを通知します。ePOS-Print API は EposException クラス (49 ページ)、プリンター検索 API は EpsonIoException クラス (136 ページ) で取得します。以下のエラーを通知します。

種類	説明
エラーステータス	各クラスの API 実行時のエラー要因です。 詳細は、エラーステータスと対処方法 (43 ページ) を参照してください。
プリンターステータス	印刷データ送信時のプリンターの状態です。 プリンターステータスは、sendData (114 ページ) を実行時にのみ取得できます。 詳細は、プリンターステータスと対処方法 (45 ページ) を参照してください。
バッテリーステータス	プリンターのバッテリー残量のステータスです。 詳細は、バッテリーステータス (46 ページ) を参照してください。

処理方法

ePOS-Print API

EposException クラスの getErrorStatus (133 ページ) でエラーステータス、getPrinterStatus (134 ページ) でプリンターステータス、getBatteryStatus (135 ページ) でバッテリーステータスを取得します。
以下のプログラミングを参考にしてください。

```
//Print クラスのインスタンスを初期化
Print printer = new Print();
import android.content.Context;

int[] status = new int[1];
int[] battery = new int[1]
status[0] = 0;
battery[0] = 0;

try {
    // 印刷ドキュメントの作成
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE, getApplicationContext());
    builder.addText("Hello,\t");
    builder.addText("World!\n");
    builder.addCut(Builder.CUT_FEED);

    // 印刷ドキュメントの送信
    ////Wi-Fi/Ethernet デバイスの場合
    printer.openPrinter(mList.getDeviceType(), mList.getDeviceName(), Print.TRUE,
                        Print.PARAM_DEFAULT);
    ////Bluetooth デバイスの場合
    printer.openPrinter(Print.DEVTYPE_BLUETOOTH, "00:00:12:34:56:78", Print.TRUE,
                        Print.PARAM_DEFAULT);
    ////USB デバイスの場合
    printer.openPrinter(Print.DEVTYPE_USB, "/dev/bus/usb/001/002", Print.TRUE,
                        Print.PARAM_DEFAULT);

    printer.sendData(builder, 10000, status, battery);
    printer.closePrinter();

} catch (EposException e) {
    // エラーステータスの取得
    int errStatus = e.getErrorStatus();
    // プリンターステータスの取得
    status[0] = e.getPrinterStatus();
    // バッテリーステータスの取得
    battery[0] = e.getBatteryStatus();
    printer.closePrinter();
}
```

プリンター検索 API

EpsonIoException クラスの getStatus（142 ページ）でエラーステータスを取得します。
以下のプログラミングを参考にしてください。

```
int errStatus = IoStatus.SUCCESS;
DeviceInfo[] mList = null;

// デバイス一覧の取得
try {

////Wi-Fi/Ethernet デバイスの場合
    Finder.start(getBaseContext(), DevType.TCP, "255.255.255.255");
////Bluetooth デバイスの場合
    Finder.start(getBaseContext(), DevType.BLUETOOTH, "null");
////USB デバイスの場合
    Finder.start(getBaseContext(), DevType.USB, "null");

    mList = getDeviceInfoList(FilterOption.PARAM_DEFAULT);
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

エラーステータスと対処方法

エラーステータスは、API を実行した各クラスに定義されています。
ここでエラーステータス内容と対処方法を説明しますので、お客様のアプリケーションでエラーハンドリング処理を行ってください。

エラーステータス	要因	対処方法
ERR_PARAM	不正なパラメーターが渡された。 < 例 > • null など、不正な引数を渡された。 • サポートしていない範囲の値が指定された。	パラメーターの指定が間違えています。 パラメーターを見直してください。
ERR_OPEN	オープン処理に失敗した。 < 例 > • 指定したプリンターに接続できなかった。 • USB 接続の場合、USB ケーブルが接続されていなかった。	Android デバイスとプリンターを確認してください。 （プリンターの電源、通信接続状態など）
ERR_CONNECT	デバイスとの通信に失敗した。 < 例 > プリンターへのデータ送信に失敗した。	closePrinter メソッドを実行後、openPrinter メソッドを実行して、Android デバイスとプリンターの通信を復帰させてください。 インターフェイスが *Bluetooth* の場合、Android OS が自動で再接続する可能性があります。 20 秒間 ステータス取得を試みて、"ERR_CONNECT" を受信し続ける場合は、Android デバイスとプリンターの通信を復帰させてください。
ERR_TIMEOUT	指定したタイムアウト時間を越えた。 < 例 > 指定された時間内に全データを送信できなかった。	タイムアウト時間を確認してください。 タイムアウト時間は、印刷所要時間以上に設定してください。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。	不要なアプリケーションを終了してください。
ERR_ILLEGAL	不適切な方法で使用された。 < 例 > • プリンターがオープンされていない状態で、プリンターにコマンドを送信する API が呼び出された。 • USB接続の場合、コンテキスト付のコンストラクターが使われなかった	API を適切な方法で使用してください。 27 ページ「プログラミングフロー」を参照してください。
ERR_PROCESSING	処理を実行できなかった。 < 例 > 同様の処理を他のスレッドで実行中のため、処理が実行できなかった。	アプリケーションの処理のタイミングを見直し、処理が重ならないようにしてください。
ERR_UNSUPPORTED	サポートしていない機種名または言語仕様が指定された。	サポートしていない機種では使用できません。

エラーステータス	要因	対処方法
ERR_OFF_LINE	プリンターがオフライン状態だった。	オフラインになる要因を取り除いてください。 （プリンターのカバーオープン、用紙切れなど）
ERR_FAILURE	その他のエラーが発生した。	• Android デバイスの通信設定を確認してください。(Wi-Fi の接続設定 / *Bluetooth* の接続設定 / USB の接続設定など） • 実行環境に問題がないか確認してください。

プリンターステータスと対処方法

TM プリンターの機種によってプリンターステータスは異なります。
ここではプリンターステータスの内容と対処方法を説明しますので、お客様のアプリケーションでエラーハンドリング処理を行ってください。

プリンターステータス	要因	対処方法
Print.ST_NO_RESPONSE (0x00000001)	• TM プリンターの電源が入っていない • 通信が確立されていいない • 通信ケーブルが抜かれている	電源、ケーブルなど プリンターの状態や通信状態を確認してください。
Print.ST_PRINT_SUCCESS (0x00000002)	印刷終了	-
<TM-P60II 以外 > Print.ST_DRAWER_KICK (0x00000004)	ドロアーキックコネクター 3 番ピンの状態＝ "H"	-
<TM-P60II> Print.ST_BATTERY_OFFLINE (0x00000004)	バッテリー残量によるオフライン状態	バッテリーを充電してください。
Print.ST_OFF_LINE (0x00000008)	オフライン状態	オフラインになる要因を取り除いてください。（カバーオープン、用紙切れなど）
Print.ST_COVER_OPEN (0x00000020)	カバーが開いている	プリンターのカバーを閉めてください。
Print.ST_PAPER_FEED (0x00000040)	紙送りスイッチによる紙送り中	-
Print.ST_PANEL_SWITCH (0x00000200)	プリンターのスイッチ、またはボタンが押されている	-
Print.ST_MECHANICAL_ERR (0x00000400)	メカニカルエラー発生	エラーの原因を取り除き、プリンターの電源を再投入してください。
Print.ST_AUTOCUTTER_ERR (0x00000800)	オートカッターエラー発生	エラーの原因を取り除き、プリンターの電源を再投入してください。
Print.ST_UNRECOVER_ERR (0x00002000)	プリンターに印刷できない復帰不可能エラー発生	ただちにプリンターの電源を切ってください。
Print.ST_AUTORECOVER_ERR (0x00004000)	ヘッドの温度が上昇し、自動復帰エラーが発生	時間の経過により、ヘッドの温度が下降すれば自動で解除されます。
Print.ST_RECEIPT_NEAR_END (0x00020000)	用紙残量が少なくなった	プリンターに用紙を入れてください。
Print.ST_RECEIPT_END (0x00080000)	用紙がなくなった	プリンターに用紙を入れてください。
Print.ST_BUZZER (0x01000000)	ブザーが鳴っている (対応機器のみ)	-
	ラベル除去待ち状態 (対応機器のみ)	ラベルを取り除いてください。

バッテリーステータス

バッテリーステータスは、以下の 16 ビット (0x0000) で構成されています。

ビット	説明
上位 8 ビット	共通のバッテリーステータス (詳細は、共通のバッテリーステータス (上位 8 ビット) (46 ページ) を参照してください。)
下位 8 ビット	機種専用のバッテリーステータス (詳細は、プリンター別サポート情報 (162 ページ) を参照してください。)

バッテリーステータス取得不可能状態、もしくは機種がバッテリーステータスに対応していない場合、″0x0000″ を返します。

共通のバッテリーステータス (上位 8 ビット)

バッテリーステータス値	要因
0x30	AC アダプターが接続されている
0x31	AC アダプターが接続されていない

API リファレンス

本章では、ePOS-Print SDK for Android で用意されている API について説明しています。

ePOS-Print API

ePOS-Print API は、印刷ドキュメントを作成し、印刷処理を行う API です。以下のクラスが用意されています。

- ❏ Builder クラス（47 ページ）
- ❏ Print クラス（49 ページ）
- ❏ EposException クラス（49 ページ）

プリンターによって、使用できる API や指定可能な設定値は異なります。
詳細は、プリンターごとのサポート API 一覧（161 ページ）およびプリンター別サポート情報（162 ページ）を参照してください。

Builder クラス

印字する文字列やグラフィックの印刷、用紙カットなどプリンターの制御命令の印刷ドキュメントを作成します。以下の API が用意されています。

API		説明	ページ
コンストラクター		Builder クラスのインスタンスを初期化	50
コンストラクター（旧フォーマット）		Builder クラスのインスタンスを初期化（ログ出力機能は使用できません）	52
命令バッファーのクリア	clearCommandBuffer	各 API で追加した命令バッファーをクリア	54
テキスト	addTextAlign	位置揃え設定を命令バッファーに追加	55
	addTextLineSpace	改行量設定を命令バッファーに追加	56
	addTextRotate	倒立印字設定を命令バッファーに追加	57
	addText	文字印字を命令バッファーに追加	58
	addTextLang	言語設定を命令バッファーに追加	59
	addTextFont	文字フォント設定を命令バッファーに追加	60
	addTextSmooth	文字スムージング設定を命令バッファーに追加	61
	addTextDouble	文字倍角設定を命令バッファーに追加	62
	addTextSize	文字倍率設定を命令バッファーに追加	63
	addTextStyle	文字装飾設定を命令バッファーに追加	64
	addTextPosition	文字印字位置設定を命令バッファーに追加	66
紙送り	addFeedUnit	ドット単位の紙送りを命令バッファーに追加	67
	addFeedLine	行単位の紙送りを命令バッファーに追加	68
	addFeedPosition	ラベル / ブラックマーク紙の紙送りを命令バッファーに追加	101

API		説明	ページ
グラフィック	addImage	ラスターイメージ印字を命令バッファーに追加 イメージデータを圧縮して命令バッファーに追加 (*Bluetooth* インターフェイス)	69
	addImage (旧フォーマット)	ラスターイメージ印字を命令バッファーに追加 (イメージデータの圧縮は使用できません (*Bluetooth* インターフェイス))	72
	addImage (旧フォーマット)	ラスターイメージ印字を命令バッファーに追加 (イメージデータの圧縮は使用できません (*Bluetooth* インターフェイス) 多階調は印刷できません)	75
	addLogo	NV ロゴ印字を命令バッファーに追加	77
バーコード	addBarcode	バーコード印字を命令バッファーに追加	78
	addSymbol	2 次元シンボル印字を命令バッファーに追加	83
ページモード	addPageBegin	ページモード開始を命令バッファーに追加	88
	addPageEnd	ページモード終了を命令バッファーに追加	89
	addPageArea	ページモード印字領域設定を命令バッファーに追加	90
	addPageDirection	ページモード印字方向設定を命令バッファーに追加	91
	addPagePosition	ページモード印字位置設定を命令バッファーに追加	92
	addPageLine	ページモード直線描画を命令バッファーに追加	93
	addPageRectangle	ページモード四角形描画を命令バッファーに追加	94
カット	addCut	用紙カットを命令バッファーに追加	95
ドロアーキック	addPulse	ドロアーキックを命令バッファーに追加	96
ブザー	addSound	ブザー鳴動を命令バッファーに追加	97
	addSound (旧フォーマット)	ブザー鳴動を命令バッファーに追加 (鳴動周期は設定できません)	99
用紙レイアウト	addLayout	用紙レイアウト情報を命令バッファーに追加	102
コマンド送信	addCommand	コマンドを命令バッファーに追加	104

Print クラス

Builder クラスで作成した印刷ドキュメントを送信してプリンターを制御したり、送信結果や通信状態を監視したりします。

API	説明	ページ
コンストラクター	Print クラスのインスタンスを初期化	105
コンストラクター (旧フォーマット)	Print クラスのインスタンスを初期化 (ログ出力機能と USB 接続での通信はできません)	106
openPrinter	プリンターとの通信を開始	107
openPrinter(旧フォーマット)	プリンターとの通信を開始 （タイムアウトが設定できません）	109
openPrinter (旧フォーマット)	プリンターとの通信を開始 （プリンターステータスの取得、およびタイムアウトが設定できません）	111
closePrinter	プリンターとの通信を終了	113
sendData	プリンターにコマンドを送信	114
sendData (旧フォーマット)	プリンターにコマンドを送信 (バッテリーステータスは取得できません)	116
setStatusChangeEventCallback	プリンターステータスの通知先を登録	118
setOnlineEventCallback	オンラインイベントの通知先を登録	119
setOfflineEventCallback	オフラインイベントの通知先を登録	120
setPowerOffEventCallback	無応答イベントの通知先を登録	121
setCoverOkEventCallback	カバークローズイベントの通知先を登録	122
setCoverOpenEventCallback	カバーオープンイベントの通知先を登録	123
setPaperOkEventCallback	用紙ありイベントの通知先を登録	124
setPaperNearEndEventCallback	用紙残量少イベントの通知先を登録	125
setPaperEndEventCallback	用紙なしイベントの通知先を登録	126
setDrawerClosedEventCallback	ドロアークローズイベントの通知先を登録	127
setDrawerOpenEventCallback	ドロアーオープンイベントの通知先を登録	128
setBatteryLowEventCallback	バッテリー残量なしイベントの通知先を登録	129
setBatteryOkEventCallback	バッテリー残量ありイベントの通知先を登録	130
setBatteryStatusChangeEventCallback	バッテリーステータスの通知先を登録	131

EposException クラス

API 実行エラーや、印刷エラーの例外発生のスロー時に、ステータスを取得します。

API	説明	ページ
getErrorStatus	エラーステータスを取得	133
getPrinterStatus	プリンターステータスを取得	134
getBatteryStatus	バッテリーステータスを取得	135

Builder クラス（コンストラクター）

Builder クラスのコンストラクターです。Builder クラスのインスタンスを初期化します。
本コンストラクターは、ログ出力機能を使用する場合に使用します。

構文

```
public Builder(String printerModel, int lang,
               Context context) throws EposException
```

パラメーター

- printerModel：対象のプリンターの機種名を指定します。

設定値	説明
"TM-P20"	TM-P20
"TM-P60II"	TM-P60II
"TM-T20II"	TM-T20II
"TM-T70"	TM-T70
"TM-T70II"	TM-T70II
"TM-T88V"	TM-T88V
"TM-T90II"	TM-T90II

- lang：　プリンターの言語仕様を指定します。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.MODEL_ANK	ANK モデル	○	○	○	○	○	○	○
Builder.MODEL_JAPANESE	日本語モデル	○	○	-	○	○	○	-

- context：　アプリケーションのコンテキストを指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_UNSUPPORTED	サポートしていない機種名または言語仕様が指定された。
ERR_FAILURE	その他のエラーが発生した。

例

TM-T88V 日本語モデル用の命令バッファーを初期化する場合

```
import android.content.Context;

try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE,
                                  getApplicationContext());
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

Builder クラス（コンストラクター）（旧フォーマット）

Builder クラスのコンストラクターです。Builder クラスのインスタンスを初期化します。

ログ出力機能は使用できません。ログ出力機能を使用する場合、Builder クラス（コンストラクター）（50 ページ）を使用してください。

構文

```
public Builder(String printerModel, int lang)
 throws EposException
```

パラメーター

- printerModel：対象のプリンターの機種名を指定します。

設定値	説明
"TM-P20"	TM-P20
"TM-P60II"	TM-P60II
"TM-T20II"	TM-T20II
"TM-T70"	TM-T70
"TM-T70II"	TM-T70II
"TM-T88V"	TM-T88V
"TM-T90II"	TM-T90II

- lang：　プリンターの言語仕様を指定します。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.MODEL_ANK	ANK モデル	○	○	○	○	○	○	○
Builder.MODEL_JAPANESE	日本語モデル	○	○	-	○	○	○	-

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_UNSUPPORTED	サポートしていない機種名または言語仕様が指定された。
ERR_FAILURE	その他のエラーが発生した。

例

TM-T88V 日本語モデル用の命令バッファーを初期化する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

clearCommandBuffer

Builder クラスの API で使用した命令バッファーをクリアします。
Builder クラスに格納された命令バッファーは、本 API を実行するまで保管されます。

構文

```
public void clearCommandBuffer()
```

例

命令バッファーをクリアする場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
   ・・・処理・・・
    builder.clearCommandBuffer();
   ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextAlign

位置揃え設定を命令バッファーに追加します。

- 本 API の設定は、バーコード /2 次元シンボルにも適用されます。
- ページモードで位置揃えを設定する場合、本 API ではなく、addPagePosition (92 ページ) で設定してください。

構文

```
public void addTextAlign(int align) throws EposException
```

パラメーター

- align：　位置揃えを指定します。

設定値	説明
Builder.ALIGN_LEFT (初期値)	左揃え
Builder.ALIGN_CENTER	中央揃え
Builder.ALIGN_RIGHT	右揃え

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

中央揃えに設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextAlign(Builder.ALIGN_CENTER);
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextLineSpace

改行量設定を命令バッファーに追加します。

構文

```
public void addTextLineSpace(int linespc)
 throws EposException
```

パラメーター

- linespc： 改行量（ドット単位）を指定します。0 ～ 255 の整数値で指定します。
（初期値：プリンター別サポート情報（162 ページ）を参照してください。）

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

改行量を 50 ドットに設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextLineSpace(50);
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextRotate

倒立印字設定を命令バッファーに追加します。

- 本 API の設定は、バーコード /2 次元シンボルにも適用されます。
- ページモードで倒立印字を設定する場合、本 API ではなく、addPageDirection (91 ページ) で設定してください。

構文

```
public void addTextRotate(int rotate)
 throws EposException
```

パラメーター

- rotate :　倒立印字の有無を指定します。

設定値	説明
Builder.TRUE	倒立印字を指定
Builder.FALSE (初期値)	倒立印字を解除

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

倒立印字を設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextRotate(Builder.TRUE);
   ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addText

文字の印字を命令バッファーに追加します。

テキストの印字後、テキスト以外を印字する場合、改行または紙送りを実行してください。
（例 : テキスト印字後、何もせずグラフィック印字を実行したが、印刷されない。）

構文

```
public void addText(String data) throws EposException
```

パラメーター

- data :　印字する文字列を指定します。
  水平タブ / 改行は、以下のエスケープシーケンスを使用します。

文字列	説明
\t	水平タブ (HT)
\n	改行 (LF)
\\	バックスラッシュ

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

文字列を追加する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addText("Hello,\t");
    builder.addText("World!\n");
   ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextLang

言語設定を命令バッファーに追加します。本 API で指定された言語情報に従って、addText (58 ページ) で指定された文字列をエンコードします。Builder クラス (コンストラクター) (50 ページ) で設定した言語仕様に合わせて指定してください。

構文

```
public void addTextLang(int lang) throws EposException
```

パラメーター

- lang :　　対象言語を指定します。

設定値	説明
Builder.LANG_EN(初期値)	英語 (ANK 仕様)
Builder.LANG_JA	日本語

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

日本語に設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextLang(Builder.LANG_JA);
   ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextFont

文字のフォント設定を命令バッファーに追加します。

構文

```
public void addTextFont(int font) throws EposException
```

パラメーター

- font： フォントを指定します。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.FONT_A (初期値)	フォント A	○	○	○	○	○	○	○
Builder.FONT_B	フォント B	○	○	○	○	○	○	○
Builder.FONT_C	フォント C	○	○	-	-	-	-	○
Builder.FONT_D	フォント D	○	-	-	-	-	-	-
Builder.FONT_E	フォント E	○	-	-	-	-	-	-

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

フォント B を設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextFont(Builder.FONT_B);
   ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextSmooth

スムージング設定を命令バッファーに追加します。

構文

```
public void addTextSmooth(int smooth)
 throws EposException
```

パラメーター

- smooth：　スムージングの有無を指定します。

設定値	説明
Builder.TRUE	スムージングを指定
Builder.FALSE (初期値)	スムージングを解除

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

スムージングを有効に設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextSmooth(Builder.TRUE);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextDouble

文字の倍角設定を命令バッファーに追加します。

構文

```
public void addTextDouble(int dw, int dh)
 throws EposException
```

パラメーター

- dw： 文字の横倍角を指定します。

設定値	説明
Builder.TRUE	横倍角を指定
Builder.FALSE (初期値)	横倍角を解除
Builder.PARAM_UNSPECIFIED	設定を変更しない

- dh： 文字の縦倍角を指定します。

設定値	説明
Builder.TRUE	縦倍角を指定
Builder.FALSE (初期値)	縦倍角を解除
Builder.PARAM_UNSPECIFIED	設定を変更しない

dw と dh のパラメーターの両方を Builder.TRUE にした場合、4 倍角の文字が印字されます。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

4 倍角を設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextDouble(Builder.TRUE, Builder.TRUE);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextSize

文字の倍率設定を命令バッファーに追加します。

構文

```
public void addTextSize(int width, int height)
 throws EposException
```

パラメーター

- width：　文字の横倍率を指定します。

設定値	説明
1 ～ 8 の整数	横方向の倍率を指定 (初期値 : 1)
Builder.PARAM_UNSPECIFIED	設定を変更しない

- height：　文字の縦倍率を指定します。

設定値	説明
1 ～ 8 の整数	縦方向の倍率を指定 (初期値 : 1)
Builder.PARAM_UNSPECIFIED	設定を変更しない

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

横倍率 4 倍、縦倍率 4 倍に設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextSize(4, 4);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextStyle

文字の装飾設定を命令バッファーに追加します。

構文

```
public void addTextStyle(int reverse, int ul, int em
, int color) throws EposException
```

パラメーター

- reverse : 白黒反転文字を指定します。

設定値	説明
Builder.TRUE	白黒反転文字を指定
Builder.FALSE (初期値)	白黒反転文字を解除
Builder.PARAM_UNSPECIFIED	設定を変更しない

- ul : アンダーラインを指定します。

設定値	説明
Builder.TRUE	アンダーラインを指定
Builder.FALSE (初期値)	アンダーラインを解除
Builder.PARAM_UNSPECIFIED	設定を変更しない

- em : 太字を指定します。

設定値	説明
Builder.TRUE	太字を指定
Builder.FALSE (初期値)	太字を解除
Builder.PARAM_UNSPECIFIED	設定を変更しない

- color : 色を指定します。

設定値	説明
Builder.COLOR_NONE	非印字 (印刷しない)
Builder.COLOR_1 (初期値)	第 1 色
Builder.PARAM_UNSPECIFIED	設定を変更しない

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

アンダーラインを設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextStyle(Builder.PARAM_UNSPECIFIED, Builder.TRUE,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addTextPosition

横方向の印字開始位置を命令バッファーに追加します。

本 API 実行後、addTextAlign (55 ページ)、addTextRotate (57 ページ) は使用できません。

構文

```
public void addTextPosition(int x) throws EposException
```

パラメーター

- x：　横方向の印字開始位置 (ドット単位) を指定します。
  0 ～ 65535 の整数値で指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

印字位置を左端から 120 ドットの位置に設定する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addTextPosition(120);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addFeedUnit

ドット単位の紙送りを命令バッファーに追加します。

構文

```
public void addFeedUnit(int unit) throws EposException
```

パラメーター

- unit：　　紙送り量（ドット単位）を指定します。0 ～ 255 の整数値で指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

紙送りを 30 ドットする場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addFeedUnit(30);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addFeedLine

行単位の紙送りを命令バッファーに追加します。

構文

```
public void addFeedLine(int line) throws EposException
```

パラメーター

- unit : 紙送り量（行単位）を指定します。0 ～ 255 の整数値で指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

紙送りを３行する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addFeedLine(3);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addImage

ラスターイメージの印字を命令バッファーに追加します。

android.graphics.Bitmap クラスのグラフィックを印字します。

android.graphics.Bitmap クラスのグラフィックのうち、指定範囲を本 API の設定に従って、ラスターイメージデータに変換します。また、画像を圧縮して送信することもできます。

画像の 1 ピクセルがプリンターの 1 ドットに相当します。透明色が含まれている場合、画像の背景を白とみなします。

- 画像圧縮は、*Bluetooth* インターフェイスの場合のみ設定してください。
- ラスターイメージを高速に印字する場合、addTextAlign (55 ページ) を Builder.ALIGN_LEFT に指定し、本 API の width パラメーターの値をプリンターの用紙幅を超えない 8 の倍数に指定してください。
- 透過画像を印字する場合、印字速度が遅くなる場合があります。
- ページモードでは多階調印字をサポートしていません。スタンダードモードでのみ多階調グラフィックスの印字が可能です。
- ページモードでは画像圧縮をサポートしていません。

構文

```
public void addImage(Bitmap data, int x, int y
                   , int width, int height, int color
                   , int mode, int halftone
                   , double brightness, int compress)
                    throws EposException
```

パラメーター

- data：　android.graphics.Bitmap クラスのインスタンスを指定します。
- x：　印字範囲の横方向の開始位置 (ピクセル単位) を指定します。
  0 ～ 65534 の整数値で指定します。
- y：　印字範囲の縦方向の開始位置 (ピクセル単位) を指定します。
  0 ～ 65534 の整数値で指定します。
- width：　印字範囲の幅 (ピクセル単位) を指定します。1 ～ 65535 の整数値で指定します。
- height：　印字範囲の高さ (ピクセル単位) を指定します。1 ～ 65535 の整数値で指定します。

x/y パラメーターと width/height パラメーターで指定された領域が data パラメーターで指定した画像のサイズに収まっていない場合、エラーステータスに ERR_PARAM を格納した EposException が発生します。

- color：　色を指定します。

設定値	説明
Builder.COLOR_NONE	非印字 (印刷しない)
Builder.COLOR_1	第 1 色
Builder.PARAM_DEFAULT	既定値 (第 1 色) を選択

- mode : カラーモードを指定します。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.MODE_MONO	モノクロ (2 階調)	○	○	○	○	○	○	○
Builder.MODE_GRAY16	多階調 (16 階調)	-	-	-	-	○	○	○
Builder.PARAM_DEFAULT	既定値を選択 (モノクロ (2 階調))	○	○	○	○	○	○	○

- halftone : ハーフトーン処理方法を指定します。

設定値	説明
Builder.HALFTONE_DITHER	ディザー (グラフィックの印刷に適しています。)
Builder.HALFTONE_ERROR_DIFFUSION	誤差拡散 (文字とグラフィックが混在する印刷に適しています。)
Builder.HALFTONE_THRESHOLD	しきい値 (文字の印刷に適しています。)
Builder.PARAM_DEFAULT	既定値 (ディザー) を選択

多階調 (16 階調) の場合、無視されます。

- brightness : 明るさの補正値を指定します。

設定値	説明
0.1 ～ 10.0 の実数	明るさ補正値 (ガンマー値)
Builder.PARAM_DEFAULT	既定値 (1.0) を選択

1.0 以外を指定した場合、印字速度が遅くなります。

- compress : 画像圧縮を指定します。Builder.COMPRESS_DEFLATE は、*Bluetooth* インターフェイスの場合のみ設定してください。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.COMPRESS_DEFLATE	画像圧縮します	○	-	○	-	○	○	-
Builder.COMPRESS_NONE	画像圧縮しません	○	○	○	○	○	○	○
Builder.PARAM_DEFAULT	既定値 (画像圧縮しません)	○	○	○	○	○	○	○

TCP/ USB 通信する場合、Builder.PARAM_DEFAULT を設定してください。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

```
try {
    Bitmap imageData = null;
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
    builder.addImage(imageData, 0, 0, 256, 256, Builder.PARAM_DEFAULT,
                     Builder.MODE_MONO, Builder.HALFTONE_DITHER, 1.0,
                     Builder.PARAM_DEFAULT);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addImage（旧フォーマット）

ラスターイメージの印字を命令バッファーに追加します。*Bluetooth* インターフェイス使用時、画像圧縮印刷ができないため、白筋が入る場合があります。

android.graphics.Bitmap クラスのグラフィックを印字します。

android.graphics.Bitmap クラスのグラフィックのうち、指定範囲を本 API の設定に従って、ラスターイメージデータに変換します。画像の 1 ピクセルがプリンターの 1 ドットに相当します。透明色が含まれている場合、画像の背景を白とみなします。

- ラスターイメージを高速に印字する場合、addTextAlign (55 ページ) を Builder.ALIGN_LEFT に指定し、本 API の width パラメーターの値をプリンターの用紙幅を超えない 8 の倍数に指定してください。
- 透過画像を印字する場合、印字速度が遅くなる場合があります。
- ページモードでは多階調印字をサポートしていません。スタンダードモードでのみ多階調グラフィックスの印字が可能です。

構文

```
public void addImage(Bitmap data, int x, int y
                   , int width, int height, int color
                   , int mode, int halftone
                   , double brightness)
                    throws EposException
```

パラメーター

- data：　android.graphics.Bitmap クラスのインスタンスを指定します。
- x：　印字範囲の横方向の開始位置（ピクセル単位）を指定します。
  0 ～ 65534 の整数値で指定します。
- y：　印字範囲の縦方向の開始位置（ピクセル単位）を指定します。
  0 ～ 65534 の整数値で指定します。
- width：　印字範囲の幅（ピクセル単位）を指定します。1 ～ 65535 の整数値で指定します。
- height：　印字範囲の高さ（ピクセル単位）を指定します。1 ～ 65535 の整数値で指定します。

x/y パラメーターと width/height パラメーターで指定された領域が data パラメーターで指定した画像のサイズに収まっていない場合、エラーステータスに ERR_PARAM を格納した EposException が発生します。

- color：　色を指定します。

設定値	説明
Builder.COLOR_NONE	非印字（印刷しない）
Builder.COLOR_1	第 1 色
Builder.PARAM_DEFAULT	既定値（第 1 色）を選択

- mode :　　カラーモードを指定します。

設定値	説明	TM プリンター別設定値						
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Builder.MODE_MONO	モノクロ (2 階調)	○	○	○	○	○	○	○
Builder.MODE_GRAY16	多階調 (16 階調)	-	-	-	-	○	○	○
Builder.PARAM_DEFAULT	既定値を選択 (モノクロ (2 階調))	○	○	○	○	○	○	○

- halftone :　　ハーフトーン処理方法を指定します。

設定値	説明
Builder.HALFTONE_DITHER	ディザー (グラフィックの印刷に適しています。)
Builder.HALFTONE_ERROR_DIFFUSION	誤差拡散 (文字とグラフィックが混在する印刷に適しています。)
Builder.HALFTONE_THRESHOLD	しきい値 (文字の印刷に適しています。)
Builder.PARAM_DEFAULT	既定値 (ディザー) を選択

多階調 (16 階調) の場合、無視されます。

- brightness :　　明るさの補正値を指定します。

設定値	説明
0.1 ～ 10.0 の実数	明るさ補正値 (ガンマー値)
Builder.PARAM_DEFAULT	既定値 (1.0) を選択

1.0 以外を指定した場合、印字速度が遅くなります。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

```
try {
    Bitmap imageData = null;
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
    builder.addImage(imageData, 0, 0, 256, 256, Builder.PARAM_DEFAULT,
                     Builder.MODE_MONO, Builder.HALFTONE_DITHER, 1.0);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

ページモードで幅 256 ドット、高さ 256 ドットの画像を印字する

```
try {
    Bitmap imageData = null;
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
    builder.addPageBegin();
    builder.addPagePosition(0, 255);
    builder.addImage(imageData, 0, 0, 256, 256, Builder.PARAM_DEFAULT,
                     Builder.MODE_MONO, Builder.HALFTONE_DITHER, 1.0);
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
```

addImage（旧フォーマット）

ラスターイメージの印字を命令バッファーに追加します。多階調印刷はできません。
Bluetooth 接続時、画像圧縮印刷ができないため、白筋が入る場合があります。

android.graphics.Bitmap クラスのグラフィックを印字します。

android.graphics.Bitmap クラスのグラフィックのうち、指定範囲をディザー処理で二値化し、ラスターイメージデータに変換します。画像の 1 ピクセルがプリンターの 1 ドットに相当します。透明色が含まれている場合、画像の背景を白とみなします。

- 多階調で印字する場合、addImage（69 ページ）を使用してください。
- ラスターイメージを高速に印字する場合、addTextAlign（55 ページ）を Builder.ALIGN_LEFT に指定し、本 API の width パラメーターの値をプリンターの用紙幅を超えない 8 の倍数に指定してください。
- 透過画像を印字する場合、印字速度が遅くなる場合があります。

構文

```
public void addImage(Bitmap data, int x, int y
                     , int width, int height, int color)
                      throws EposException
```

パラメーター

- data：　android.graphics.Bitmap クラスのインスタンスを指定します。
- x：　印字範囲の横方向の開始位置（ピクセル単位）を指定します。
  0 ～ 65534 の整数値で指定します。
- y：　印字範囲の縦方向の開始位置（ピクセル単位）を指定します。
  0 ～ 65534 の整数値で指定します。
- width：　印字範囲の幅（ピクセル単位）を指定します。1 ～ 65535 の整数値で指定します。
- height：　印字範囲の高さ（ピクセル単位）を指定します。1 ～ 65535 の整数値で指定します。

x/y パラメーターと width/height パラメーターで指定された領域が data パラメーターで指定した画像のサイズに収まっていない場合、エラーステータスに ERR_PARAM を格納した EposException が発生します。

- color：　色を指定します。

設定値	説明
Builder.COLOR_NONE	非印字（印刷しない）
Builder.COLOR_1	第 1 色
Builder.PARAM_DEFAULT	既定値（第 1 色）を選択

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

```
try {
    Bitmap imageData = null;
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
    builder.addImage(imageData, 0, 0, 256, 256, Builder.PARAM_DEFAULT);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

ページモードで幅 256 ドット、高さ 256 ドットの画像を印字する

```
try {
    Bitmap imageData = null;
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
     ・・・処理・・・
    builder.addPageBegin();
    builder.addPagePosition(0, 255);
    builder.addImage(imageData, 0, 0, 256, 256, Builder.PARAM_DEFAULT);
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addLogo

NV ロゴの印字を命令バッファーに追加します。プリンターの NV メモリーに登録されているロゴを印字します。

- ロゴは以下のユーティリティーを使って、あらかじめプリンターにロゴの登録します。
  * 機種専用ユーティリティー
  * ロゴ登録ユーティリティー(TMFLogo)
- ページモードでは多階調印字をサポートしていません。スタンダードモードでのみ多階調グラフィックスの印字が可能です。

構文

```
public void addLogo(int key1, int key2)
 throws EposException
```

パラメーター

- key1：　NV ロゴのキーコード 1 を指定します。32 ～ 126 の整数値で指定します。
- key2：　NV ロゴのキーコード 2 を指定します。32 ～ 126 の整数値で指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

キーコード 48,48 の NV ロゴを印字する

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addLogo(48, 48);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addBarcode

バーコード印字を命令バッファーに追加します。

構文

```
public void addBarcode
(String data, int type, int hri, int font, int width,
 int height) throws EposException
```

パラメーター

- data： バーコードデータを文字列で指定します。

type で指定するバーコードの規格に従った文字列を指定してください。規格に従っていない場合、バーコードは印刷されません。

種類	説明
UPC-A	11 桁の数字を指定した場合、チェックデジットを自動で付加します。 12 桁の数字を指定した場合、12 桁目をチェックデジットとして処理しますが、チェックデジットの検算は行いません。
UPC-E	最初の桁に 0 を指定してください。 2 ～ 6 桁目にメーカーコードを指定してください。 7 ～ 11 桁目にアイテムコードを右詰めで指定してください。アイテムコードの桁数はメーカーコードにより異なります。使用しない桁は 0 を指定してください。 11 桁の数字を指定した場合、チェックデジットを自動で付加します。 12 桁の数字を指定した場合、12 桁目をチェックデジットとして処理しますが、チェックデジットの検算は行いません。
EAN13 JAN13	12 桁の数字を指定した場合、チェックデジットを自動で付加します。 13 桁の数字を指定した場合、13 桁目をチェックデジットとして処理しますが、チェックデジットの検算は行いません。
EAN8 JAN8	7 桁の数字を指定した場合、チェックデジットを自動で付加します。 8 桁の数字を指定した場合、8 桁目をチェックデジットとして処理しますが、チェックデジットの検算は行いません。
CODE39	先頭の文字が * の場合、この文字をスタートキャラクターとして処理します。それ以外の場合、スタートキャラクターを自動で付加します。
ITF	スタートコードおよびストップコードを自動で付加します。 チェックデジットの付加および検算は行いません。

種類	説明
CODABAR	スタートキャラクター (A ～ D, a ～ d) を 指定してください。 ストップキャラクター (A ～ D, a ～ d) を 指定してください。 チェックデジットの付加および検算は行いません。
CODE93	スタートキャラクターおよびストップキャラクターを自動で付加します。 チェックデジットを計算して自動で付加します。
CODE128	スタートキャラクター (CODE A, CODE B, CODE C) を指定してください。 ストップキャラクターを自動で付加します。 チェックデジットを計算して自動で付加します。 以下の文字をエンコードするには、文字 { で始まる 2 文字を指定してください。 FNC1: {1 FNC2: {2 FNC3: {3 FNC4: {4 CODE A: {A CODE B: {B CODE C: {C SHIFT: {S {: {{
GS1-128	スタートキャラクター、FNC1、チェックデジット、ストップキャラクターを自動で付加します。 アプリケーション識別子 (AI) とそれに続くデータのチェックデジットを計算して自動で付加するには、チェックデジットの位置に文字 * を指定します。 アプリケーション識別子 (AI) を括弧で囲むことができます。括弧は HRI の印字文字として使用し、データとしてエンコードしません。 アプリケーション識別子 (AI) とデータの間に空白を挿入することができます。空白は HRI の印字文字として使用し、データとしてエンコードしません。 以下の文字をエンコードするには、文字 { で始まる 2 文字を指定してください。 FNC1: {1 FNC3: {3 (: {( ): {) *: {* {: {{
GS1 DataBar Omnidirectional GS1 DataBar Truncated GS1 DataBar Limited	アプリケーション識別子 (AI) とチェックデジットを除く 13 桁の商品識別番号 (GTIN) を指定してください。

種類	説明
GS1 DataBar Expanded	アプリケーション識別子 (AI) を括弧で囲むことができます。括弧は HRI の印字文字として使用し、データとしてエンコードしません。 以下の文字をエンコードするには、文字 { で始まる 2 文字を指定してください。 FNC1: {1 (: {( ): {)

文字列で表現できないバイナリーデータを指定する場合、
以下のエスケープシーケンスで指定します。

文字列	説明
\xnn	コントロールコード
\\	バックスラッシュ

- type： バーコードの種類を指定します。

設定値	説明
Builder.BARCODE_UPC_A	UPC-A
Builder.BARCODE_UPC_E	UPC-E
Builder.BARCODE_EAN13	EAN13
Builder.BARCODE_JAN13	JAN13
Builder.BARCODE_EAN8	EAN8
Builder.BARCODE_JAN8	JAN8
Builder.BARCODE_CODE39	CODE39
Builder.BARCODE_ITF	ITF
Builder.BARCODE_CODABAR	CODABAR
Builder.BARCODE_CODE93	CODE93
Builder.BARCODE_CODE128	CODE128
Builder.BARCODE_GS1_128	GS1-128
Builder.BARCODE_GS1_DATABAR_OMNIDIRECTIONAL	GS1 DataBar Omnidirectional
Builder.BARCODE_GS1_DATABAR_TRUNCATED	GS1 DataBar Truncated
Builder.BARCODE_GS1_DATABAR_LIMITED	GS1 DataBar Limited
Builder.BARCODE_GS1_DATABAR_EXPANDED	GS1 DataBar Expanded

- hri： HRI の位置を指定します。

設定値	説明
Builder.HRI_NONE(初期値)	印字しない
Builder.HRI_ABOVE	バーコードの上
Builder.HRI_BELOW	バーコードの下
Builder.HRI_BOTH	バーコードの上と下の両方
Builder.PARAM_UNSPECIFIED	設定を変更しない

- font：　HRI フォントを指定します。

設定値	説明
Builder.FONT_A(初期値)	フォント A
Builder.FONT_B	フォント B
Builder.FONT_C	フォント C
Builder.FONT_D	フォント D
Builder.FONT_E	フォント E
Builder.PARAM_UNSPECIFIED	設定を変更しない

- width：　1 モジュールの幅をドット単位で指定します。

設定値	説明
2 ～ 6 の整数値	1 モジュールの幅 (ドット単位)
Builder.PARAM_UNSPECIFIED	設定を変更しない

- height：　バーコードの高さをドット単位で指定します。1 ～ 255 の整数値で指定します。

設定値	説明
1 ～ 255 の整数値	バーコードの高さ (ドット単位)
Builder.PARAM_UNSPECIFIED	設定を変更しない

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

各種バーコードを印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addBarcode("01234567890", Builder.BARCODE_UPC_A,
            Builder.HRI_BELOW, Builder.PARAM_UNSPECIFIED, 2, 64);
    builder.addBarcode("01234500005", Builder.BARCODE_UPC_E,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("201234567890", Builder.BARCODE_EAN13,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("201234567890", Builder.BARCODE_JAN13,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("2012345", Builder.BARCODE_EAN8,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("2012345", Builder.BARCODE_JAN8,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("ABCDE", Builder.BARCODE_CODE39,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("012345", Builder.BARCODE_ITF,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("A012345A", Builder.BARCODE_CODABAR,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("ABCDE", Builder.BARCODE_CODE93,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("{Babcde", Builder.BARCODE_CODE128,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("(01)201234567890*", Builder.BARCODE_GS1_128,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("0201234567890",
            Builder.BARCODE_GS1_DATABAR_OMNIDIRECTIONAL,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("0201234567890",
            Builder.BARCODE_GS1_DATABAR_TRUNCATED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("0201234567890",
            Builder.BARCODE_GS1_DATABAR_LIMITED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
    builder.addBarcode("(01)2012345678903",
            Builder.BARCODE_GS1_DATABAR_EXPANDED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addSymbol

2 次元シンボル印字を命令バッファーに追加します。

構文

```
public void addSymbol
(String data, int type, int level, int width,
 int height, int size)
 throws EposException
```

パラメーター

- data：　　2 次元シンボルデータを文字列で指定します。

type で指定する 2 次元シンボルの規格に従った文字列を指定してください。規格に従っていない場合、2 次元シンボルは印刷されません。

文字列	説明
Standard PDF417	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 データ領域の最大コードワード数は 928 個、1 段あたりのデータ領域の最大コードワード数は 30 個、最大段数は 90 段です。
Truncated PDF417	
QR Code Model 1	文字列をシフト JIS に変換後、エスケープシーケンスの処理を行い、データの種類を以下の中から選択してエンコードします。 数字：　0 ～ 9 英数字：　0 ～ 9, A ～ Z, スペース , $, %, *, +, -, ., /, : 漢字：　シフト JIS 値 8 ビットバイトデータ： 0x00 ～ 0xff
QR Code Model 2	

文字列	説明
MaxiCode Mode 2 MaxiCode Mode 3 MaxiCode Mode 4 MaxiCode Mode 5 MaxiCode Mode 6	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 モード 2 およびモード 3 の場合、最初のデータが ()>\x1e01\x1dyy (yy は 2 桁の数字) の場合、これをメッセージヘッダーとして処理し、次のデータからプライマリメッセージとして処理します。それ以外の場合、最初のデータからプライマリメッセージとして処理します。 モード 2 の場合、プライマリメッセージを以下の形式で指定してください。 郵便コード (1 ～ 9 桁の数字) GS:(\x1d) ISO 国名コード (1 ～ 3 桁の数字) GS:(\x1d) サービスクラスコード (1 ～ 3 桁の数字) モード 3 の場合、プライマリメッセージを以下の形式で指定してください。 郵便コード (1 ～ 6 個のコードセット A で変換可能なデータ) GS(\x1d) ISO 国名コード (1 ～ 3 桁の数字) GS(\x1d) サービスクラスコード (1 ～ 3 桁の数字)
GS1 DataBar Stacked GS1 DataBar Stacked Omnidirectional	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 アプリケーション識別子 (AI) とチェックデジットを除く 13 桁の商品識別番号 (GTIN) を指定してください。
GS1 DataBar Expanded Stacked	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 アプリケーション識別子 (AI) を括弧で囲むことができます。括弧は HRI の印字文字として使用し、データとしてエンコードしません。 以下の文字をエンコードするには、文字 { で始まる 2 文字を指定してください。 FNC1: {1 (: {( ): {)
Aztec Code Full-Range モード	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 最大でテキスト 3067 文字、数字 3832 文字、バイナリーデータ 1914 バイトを指定できます。
Aztec Code Compact モード	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 最大でテキスト 89 文字、数字 110 文字、バイナリーデータ 53 バイトを指定できます。
DataMatrix 正方形 DataMatrix 長方形、行数 8 DataMatrix 長方形、行数 12 DataMatrix 長方形、行数 16	文字列を UTF-8 に変換後、エスケープシーケンスの処理を行い、エンコードします。 シンボルは 10 行 x10 列～ 144 行 x144 列の正方形、または行数 8、行数 12、行数 16 の長方形です。 最大で英数字 2335 文字、数字 3116 文字、バイナリーデータ 1556 バイトを指定できます。

文字列で表現できないバイナリーデータを指定する場合、以下のエスケープシーケンスで指定します。

文字列	説明
\xnn	コントロールコード
\\	バックスラッシュ

- type：　２次元シンボルの種類を指定します。

設定値	種類
Builder.SYMBOL_PDF417_STANDARD	Standard PDF417
Builder.SYMBOL_PDF417_TRUNCATED	Truncated PDF417
Builder.SYMBOL_QRCODE_MODEL_1	QR Code Model 1
Builder.SYMBOL_QRCODE_MODEL_2	QR Code Model 2
Builder.SYMBOL_MAXICODE_MODE_2	MaxiCode Mode 2
Builder.SYMBOL_MAXICODE_MODE_3	MaxiCode Mode 3
Builder.SYMBOL_MAXICODE_MODE_4	MaxiCode Mode 4
Builder.SYMBOL_MAXICODE_MODE_5	MaxiCode Mode 5
Builder.SYMBOL_MAXICODE_MODE_6	MaxiCode Mode 6
Builder.SYMBOL_GS1_DATABAR_STACKED	GS1 DataBar Stacked
Builder.SYMBOL_GS1_DATABAR_STACKED_OMNIDIRECTIONAL	GS1 DataBar Stacked Omnidirectional
Builder.SYMBOL_GS1_DATABAR_EXPANDED_STACKED	GS1 DataBar Expanded Stacked
Builder.SYMBOL_AZTECCODE_FULLRANGE	Aztec Code Full-Range モード
Builder.SYMBOL_AZTECCODE_COMPACT	Aztec Code Compact モード
Builder.SYMBOL_DATAMATRIX_SQUARE	DataMatrix 正方形
Builder.SYMBOL_DATAMATRIX_RECTANGLE_8	DataMatrix 長方形、行数 8
Builder.SYMBOL_DATAMATRIX_RECTANGLE_12	DataMatrix 長方形、行数 12
Builder.SYMBOL_DATAMATRIX_RECTANGLE_16	DataMatrix 長方形、行数 16

- level：　エラー訂正レベルを指定します。

設定値	説明
Builder.LEVEL_0	PDF417 エラー訂正レベル 0
Builder.LEVEL_1	PDF417 エラー訂正レベル 1
Builder.LEVEL_2	PDF417 エラー訂正レベル 2
Builder.LEVEL_3	PDF417 エラー訂正レベル 3
Builder.LEVEL_4	PDF417 エラー訂正レベル 4
Builder.LEVEL_5	PDF417 エラー訂正レベル 5
Builder.LEVEL_6	PDF417 エラー訂正レベル 6
Builder.LEVEL_7	PDF417 エラー訂正レベル 7
Builder.LEVEL_8	PDF417 エラー訂正レベル 8
Builder.LEVEL_L	QR Code エラー訂正レベル L
Builder.LEVEL_M	QR Code エラー訂正レベル M
Builder.LEVEL_Q	QR Code エラー訂正レベル Q
Builder.LEVEL_H	QR Code エラー訂正レベル H
5 ～ 95 の整数	Aztec Code エラー訂正レベル (パーセント単位)
Builder.LEVEL_DEFAULT	既定レベル
Builder.PARAM_UNSPECIFIED	設定を変更しない

- 2 次元シンボルの種類に合わせて選択してください。
- MaxiCode/2 次元 GS1 DataBar/DataMatrix の場合、Builder.LEVEL_DEFAULT を選択してください。

- width：　モジュールの幅を指定します。

設定値	説明
1 ～ 255 の整数値	モジュールの幅
Builder.PARAM_UNSPECIFIED	設定を変更しない

MaxiCode は無視さ れま す。

- height：　モジュールの高さを指定します。

設定値	説明
1 ～ 255 の整数値	モジュールの高さ
Builder.PARAM_UNSPECIFIED	設定を変更しない

QR Code/MaxiCode/ 2 次元 GS1 DataBar/Aztec Code/DataMatrix は無視されます。

- size：　2 次元シンボルの最大サイズを指定します。

設定値	説明
0 ～ 65535 の整数値	2 次元シンボルの最大サイズ
Builder.PARAM_UNSPECIFIED	設定を変更しない

QR Code/MaxiCode/Aztec Code/DataMatrix は無視されます。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

各種 2 次元シンボルを印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addSymbol("ABCDE", Builder.SYMBOL_PDF417_STANDARD,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addSymbol("ABCDE", Builder.SYMBOL_QRCODE_MODEL_2,
            Builder.LEVEL_Q, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addSymbol("908063840\\x1d850\\x1d001\\x1d\\x04",
            Builder.SYMBOL_MAXICODE_MODE_2, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
   builder.addSymbol("0201234567890", Builder.SYMBOL_GS1_DATABAR_STACKED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addSymbol("0201234567890",
            Builder.SYMBOL_GS1_DATABAR_STACKED_OMNIDIRECTIONAL,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.addSymbol("(01)02012345678903",
            Builder.SYMBOL_GS1_DATABAR_EXPANDED_STACKED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageBegin

ページモード開始を命令バッファーに追加します。ページモードの処理が開始します。

本 API は addPageEnd (89 ページ) と一緒にお使いください。

構文

```
public void addPageBegin() throws EposException
```

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

ページモードで文字「ABCDE」を印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPageBegin();
    builder.addText("ABCDE");
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageEnd

ページモード終了を命令バッファーに追加します。ページモードの処理が終了します。

本 API は、addPageBegin (88 ページ) と一緒にお使いください。

構文

```
public void addPageEnd() throws EposException
```

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

ページモードで文字「ABCDE」を印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPageBegin();
    builder.addText("ABCDE");
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageArea

ページモード印字領域を命令バッファーに追加します。

ページモード印字領域（座標）を指定します。本 API に続けて、addText など印刷データの API を指定します。

- 印字内容に合わせて印字領域を指定してください。印字データが印字領域をはみ出た場合、印字データが途中で切れた印字結果になります。
- 本 API は addPageBegin（88 ページ）と addPageEnd（89 ページ）に挟んでお使いください。

構文

```
public void addPageArea(int x, int y, int width
, int height) throws EposException
```

パラメーター

- x：　横方向の原点（ドット単位）を指定します。0 ～ 65535 の整数値で指定します。0 はプリンターの印字可能領域の左端になります。
- y：　縦方向の原点（ドット単位）を指定します。0 ～ 65535 の整数値で指定します。0 は紙送りをしていない位置です。
- width：　印字領域の幅（ドット単位）を指定します。1 ～ 65535 の整数値で指定します。
- height：　印字領域の高さ（ドット単位）を指定します。1 ～ 65535 の整数値で指定します。

印字領域の幅と高さは、印字方向の設定に合わせて確定してください。
印字データが切れてしまう場合があります。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

原点 (100, 50)、幅 200 ドット、高さ 30 ドットの印字領域を指定して、文字「ABCDE」を印字す

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPageBegin();
    builder.addPageArea(100, 50, 200, 30);
    builder.addText("ABCDE");
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageDirection

ページモード印字方向設定を命令バッファーに追加します。ページモードの印字方向を指定します。
回転させない場合は、省略できます。

本 API は addPageBegin (88 ページ) と addPageEnd (89 ページ) に挟んでお使いください。

構文

```
public void addPageDirection(int dir)
 throws EposException
```

パラメーター

- dir：　ページモードの印字方向を指定します。

設定値	説明
Builder.DIRECTION_LEFT_TO_RIGHT (初期値)	回転しない (左上を始点に右方向へ印字)
Builder.DIRECTION_BOTTOM_TO_TOP	反時計回り 90 度回転 (左下を始点に上方向へ印字)
Builder.DIRECTION_RIGHT_TO_LEFT	180 度回転 (右下を始点に左方向へ印字)
Builder.DIRECTION_TOP_TO_BOTTOM	時計回り 90 度回転 (右上を始点に下方向へ印字)

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

時計回りに 90 度回転させて、文字「ABCDE」を印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPageBegin();
    builder.addPageArea(100, 50, 30, 200);
    builder.addPageDirection(Builder.DIRECTION_TOP_TO_BOTTOM);
    builder.addText("ABCDE");
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPagePosition

ページモードの印字位置設定領域を命令バッファーに追加します。

addPageArea で指定したエリア内での、印字開始位置（座標）を指定します。

本 API は addPageBegin（88 ページ）と addPageEnd（89 ページ）に挟んでお使いください。

構文

```
public void addPagePosition(int x, int y)
 throws EposException
```

パラメーター

- x：　横方向の印字位置（ドット単位）を指定します。0 ～ 65535 の整数値で指定します。
- y：　縦方向の印字位置（ドット単位）を指定します。0 ～ 65535 の整数値で指定します。

印字開始位置（座標）は、印字内容に合わせて指定してください。以下を参考にしてください。

* 文字列を印字する場合
  最初の文字のベースライン左端を指定します。
  標準の大きさで左詰めの印字をする場合は省略可能です。高さが 2 倍の文字を印刷する場合は、y を 42 以上に指定します。
* バーコードを印字する場合
  シンボルの左下を指定します。y にバーコードの高さを指定してください。
* グラフィック / ロゴを印字する場合
  グラフィックデータの左下を指定します。y にグラフィックデータの高さを指定してください。
* 2 次元シンボルを印字する場合
  シンボルの左上を指定します。左上から印字する場合は、省略可能です。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

addPageArea で指定したエリア内の印字開始位置を (50, 30) に指定して、文字「ABCDE」を印字する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPageBegin();
    builder.addPageArea(100, 50, 200, 100);
    builder.addPagePosition(50, 30);
    builder.addText("ABCDE");
    builder.addPageEnd();
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageLine

ページモードの直線描画を命令バッファーに追加します。ページモードで直線を描画します。

- 斜線は描画できません。
- 本 API は addPageBegin (88 ページ) と addPageEnd (89 ページ) に挟んでお使いください。

構文

```
public void addPageLine
(int x1, int y1, int x2, int y2, int style)
 throws EposException
```

パラメーター

- x1:　横方向の描画開始位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- y1:　縦方向の描画開始位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- x2 :　横方向の描画終了位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- y2 :　縦方向の描画終了位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- style :　罫線の種類を指定します。

設定値	説明
Builder.LINE_THIN	実線 : 細
Builder.LINE_MEDIUM	実線 : 中太
Builder.LINE_THICK	実線 : 太
Builder.LINE_THIN_DOUBLE	二重線 : 細
Builder.LINE_MEDIUM_DOUBLE	二重線 : 中太
Builder.LINE_THICK_DOUBLE	二重線 : 太
Builder.PARAM_DEFAULT	実線 : 細

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

開始位置 (100, 0), 終了位置 (500, 0) を頂点とする直線を、細い実線で描画する場合

```
try {
    Builder builder = new Builder("TM-P60", Builder.MODEL_ANK);
    builder.addPageBegin();
    builder.addPageLine(100, 0, 500, 0, Builder.LINE_THIN);
    builder.addPageEnd();
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPageRectangle

ページモードの四角形描画を命令バッファーに追加します。ページモードで四角形を描画します。

本 API は addPageBegin (88 ページ) と addPageEnd (89 ページ) に挟んでお使いください。

構文

```
public void addPageRectangle
(int x1, int y1, int x2, int y2, int style)
 throws EposException
```

パラメーター

- x1: 横方向の描画開始位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- y1: 縦方向の描画開始位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- x2 : 横方向の描画終了位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- y2 : 縦方向の描画終了位置 (ドット単位) を指定します。0 ～ 65535 の整数値で指定します。
- style : 線の種類を指定します。

設定値	説明
Builder.LINE_THIN	実線 : 細
Builder.LINE_MEDIUM	実線 : 中太
Builder.LINE_THICK	実線 : 太
Builder.LINE_THIN_DOUBLE	二重線 : 細
Builder.LINE_MEDIUM_DOUBLE	二重線 : 中太
Builder.LINE_THICK_DOUBLE	二重線 : 太
Builder.PARAM_DEFAULT	実線 : 細

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

開始位置 (100, 0), 終了位置 (500, 200) を頂点とする四角形を、細い実線で描画する場合

```
try {
    Builder builder = new Builder("TM-P60", Builder.MODEL_ANK);
    builder.addPageBegin();
    builder.addPageRectangle(100, 0, 500, 200, Builder.LINE_THIN);
    builder.addPageEnd();
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addCut

用紙カットを命令バッファーに追加します。用紙カットを設定します。

ページモードでは使用できません。

構文

```
public void addCut(int type) throws EposException
```

パラメーター

- type：　用紙カット方法を指定します。

設定値	説明
Builder.CUT_NO_FEED	フィードなしカット （紙送りせずにカット）
Builder.CUT_FEED	フィードカット （紙送り後カット）
Builder.CUT_RESERVE	カット予約 （後に続く印字を実行後、カット位置でカット）
Builder.PARAM_DEFAULT	フィードカット （紙送り後カット）

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

フィードカットする場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addCut(Builder.CUT_FEED);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addPulse

ドロアーキックを命令バッファーに追加します。ドロアーキックを設定します。

- ページモードでは使用できません。
- ドロアーは、ブザーと一緒に使用できません。

構文

```
public void addPulse(int drawer, int time)
 throws EposException
```

パラメーター

- drawer： ドロアーキックコネクターを指定します。

設定値	説明
Builder.DRAWER_1	ドロアーキックコネクター 2 番ピン
Builder.DRAWER_2	ドロアーキックコネクター 5 番ピン
Builder.PARAM_DEFAULT	ドロアーキックコネクター 2 番ピン

- time： ドロアーキック信号のオン時間を指定します。

設定値	説明
Builder.PULSE_100	100 ミリ秒の信号
Builder.PULSE_200	200 ミリ秒の信号
Builder.PULSE_300	300 ミリ秒の信号
Builder.PULSE_400	400 ミリ秒の信号
Builder.PULSE_500	500 ミリ秒の信号
Builder.PARAM_DEFAULT	100 ミリ秒の信号

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

ドロアーキックコネクター 2 番ピンに 100 ミリ秒のパルス信号を出力する場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addPulse(Builder.DRAWER_1, Builder.PULSE_100);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addSound

ブザーの鳴動を命令バッファーに追加します。ブザーを設定します。

- ページモードでは使用できません。
- ブザーの機能は、ドロアーと一緒に使用できません。
- 本 API はプリンターにブザーが付いてなければ使用できません。

構文

```
public void addSound(int pattern, int repeat, int cycle)
 throws EposException
```

パラメーター

- pattern：　ブザーの音色を指定します。

設定値	説明
Builder.PATTERN_A	パターン A (外付けオプションブザー)
Builder.PATTERN_B	パターン B (外付けオプションブザー)
Builder.PATTERN_C	パターン C (外付けオプションブザー)
Builder.PATTERN_D	パターン D (外付けオプションブザー)
Builder.PATTERN_E	パターン E (外付けオプションブザー)
Builder.PATTERN_ERROR	エラー鳴動パターン (外付けオプションブザー)
Builder.PATTERN_PAPER_END	用紙なし鳴動パターン (外付けオプションブザー)
Builder.PATTERN_1	パターン 1 (内蔵ブザー)
Builder.PATTERN_2	パターン 2 (内蔵ブザー)
Builder.PATTERN_3	パターン 3 (内蔵ブザー)
Builder.PATTERN_4	パターン 4 (内蔵ブザー)
Builder.PATTERN_5	パターン 5 (内蔵ブザー)
Builder.PATTERN_6	パターン 6 (内蔵ブザー)
Builder.PATTERN_7	パターン 7 (内蔵ブザー)
Builder.PATTERN_8	パターン 8 (内蔵ブザー)
Builder.PATTERN_9	パターン 9 (内蔵ブザー)
Builder.PATTERN_10	パターン 10 (内蔵ブザー)
Builder.PARAM_DEFAULT	パターン A

- repeat：　繰り返し回数を指定します。

設定値	説明
1 ～ 255	1 ～ 255 回
Builder.PARAM_DEFAULT	1 回

- cycle：　ブザーを鳴らす周期 (ミリ秒単位) を指定します。

設定値	説明
1000 ～ 25500	1000 ～ 25500 ミリ秒
Builder.PARAM_DEFAULT	既定値 (1000 ミリ秒) を選択

パターン A ～ E/ エラー鳴動パターン / 用紙なし鳴動パターンは無視されます。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

パターン 1 を 1000 ミリ秒周期で 3 回鳴らす場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addSound(Builder.PATTERN_1, 3, 1000);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addSound（旧フォーマット）

ブザーの鳴動を命令バッファーに追加します。ブザーを設定します。

- ブザーを鳴らす周期は設定できません。ブザーを鳴らす周期（ミリ秒単位）を任意で設定したい場合、addSound（97 ページ）を使用してください。
- ページモードでは使用できません。
- ブザーの機能は、ドロアーと一緒に使用できません。
- 本 API はプリンターにブザーが付いてなければ使用できません。

構文

```
public void addSound(int pattern, int repeat)
 throws EposException
```

パラメーター

- pattern：　ブザーの音色を指定します。

設定値	説明
Builder.PATTERN_A	パターン A（外付けオプションブザー）
Builder.PATTERN_B	パターン B（外付けオプションブザー）
Builder.PATTERN_C	パターン C（外付けオプションブザー）
Builder.PATTERN_D	パターン D（外付けオプションブザー）
Builder.PATTERN_E	パターン E（外付けオプションブザー）
Builder.PATTERN_ERROR	エラー鳴動パターン（外付けオプションブザー）
Builder.PATTERN_PAPER_END	用紙なし鳴動パターン（外付けオプションブザー）
Builder.PATTERN_1	パターン 1（内蔵ブザー）
Builder.PATTERN_2	パターン 2（内蔵ブザー）
Builder.PATTERN_3	パターン 3（内蔵ブザー）
Builder.PATTERN_4	パターン 4（内蔵ブザー）
Builder.PATTERN_5	パターン 5（内蔵ブザー）
Builder.PATTERN_6	パターン 6（内蔵ブザー）
Builder.PATTERN_7	パターン 7（内蔵ブザー）
Builder.PATTERN_8	パターン 8（内蔵ブザー）
Builder.PATTERN_9	パターン 9（内蔵ブザー）
Builder.PATTERN_10	パターン 10（内蔵ブザー）
Builder.PARAM_DEFAULT	パターン A

- repeat：　繰り返し回数を指定します。

設定値	説明
1 ～ 255	1 ～ 255 回
Builder.PARAM_DEFAULT	1 回

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

パターンＡを３回鳴らす場合

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addSound(Builder.PATTERN_A, 3);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addFeedPosition

ラベル / ブラックマーク紙の紙送りを命令バッファーに追加します。

構文

```
public void addFeedPosition(int position)
 throws EposException
```

パラメーター

- position：　紙送りする位置を指定します。

設定値	説明
Builder.FEED_PEELING	剥離位置まで紙送り
Builder.FEED_CUTTING	カット位置まで紙送り
Builder.FEED_CURRENT_TOF	現在のラベル頭出し位置まで紙送り
Builder.FEED_NEXT_TOF	次のラベル頭出し位置まで紙送り

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

ラベル紙を剥離位置まで紙送りする場合

```
try {
    Builder builder = new Builder("TM-P60II", Builder.MODEL_JAPANESE);
    builder.addFeedPosition(Builder.FEED_PEELING);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

addLayout

ラベル / ブラックマーク紙の用紙レイアウト情報を命令バッファーに追加します。

構文

```
public void addLayout(int type, int width, int height,
                      int marginTop, int marginBottom,
                      int offsetCut, int offsetLabel)
                      throws EposException
```

パラメーター

- type : 用紙種類を指定します。

設定値	説明
Builder.LAYOUT_RECEIPT	レシート紙 (ブラックマークなし)
Builder.LAYOUT_LABEL	ラベル紙 (ブラックマークなし)
Builder.LAYOUT_LABEL_BM	ラベル紙 (ブラックマークあり)
Builder.LAYOUT_RECEIPT_BM	レシート紙 (ブラックマークあり)

- width : 用紙幅 (0.1 mm 単位) を指定します。1 ～ 10000 の整数値で指定します。
- height : 印字基準から次の印字基準までの距離 (0.1 mm 単位) を指定します。
  1 ～ 10000 の整数値で指定します。
  0 を指定した場合、印字基準位置から次の印字基準位置までの距離を自動検出します。
- marginTop : 印字基準から頭出し位置までの距離 (0.1 mm 単位) を指定します。
  -9999 ～ 10000 の整数値で指定します。
- marginBottom :排出基準から印刷可能領域の下端までの距離 (0.1 mm 単位) を指定します。
  -9999 ～ 10000 の整数値で指定します。
- offsetCut : 排出基準からカット位置までの距離 (0.1 mm 単位) を指定します。
  -9999 ～ 10000 の整数値で指定します。
- offsetLabel : 排出基準からラベル下端までの距離 (0.1 mm 単位) を指定します。
  0 ～ 10000 の整数値で指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

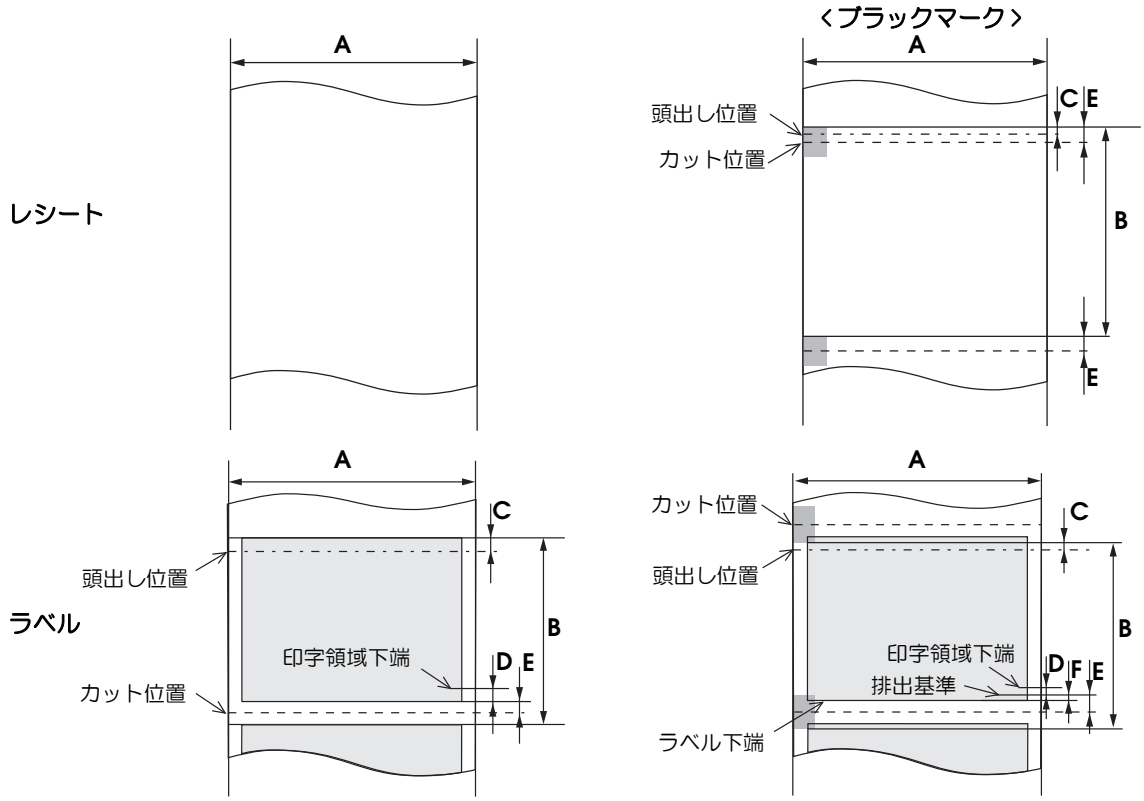
例

60 mm ラベル紙（ブラックマークあり）に設定する場合

```
try {
    Builder builder = new Builder("TM-P60II", Builder.MODEL_JAPANESE);
    builder.addLayout(Builder.PAPER_TYPE_LABEL_BM, 600, 0,
                      15, -15, 15, 0);
     ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

詳細説明

❏ 用紙ごと指定可能なパラメーターと、パラメーターの位置は以下を参照してください。



記号	パラメーター	設定値			
		レシート	レシート（ブラックマーク）	ラベル	ラベル（ブラックマーク）
A	width	1 ～ 10000	1 ～ 10000	1 ～ 10000	1 ～ 10000
B	height	0	0 ～ 10000	0 ～ 10000	0 ～ 10000
C	marginTop	0	-9999 ～ 10000	0 ～ 10000	-9999 ～ 10000
D	marginBottom	0	0	-9999 ～ 0	-9999 ～ 10000
E	offsetCut	0	-9999 ～ 10000	0 ～ 10000	0 ～ 10000
F	offsetLabel	0	0	0	0 ～ 10000

addCommand

コマンドを命令バッファーに追加します。ESC/POS コマンドを送信します。

コマンドの詳細は、ESC/POS コマンドリファレンスを参照してください。
https://reference.epson-biz.com/modules/ref_escpos_ja/

構文

```
public void addCommand(byte[] data) throws EposException
```

パラメーター

- data： ESC/POS コマンドをバイナリーデータで指定します。

例外

処理に失敗した場合、以下のエラーステータスの EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

```
try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    byte[] data = null;
     ・・・処理・・・
    builder.addCommand(data);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

Print クラス（コンストラクター）

Print クラスのコンストラクターです。Print クラスのインスタンスを初期化します。
本コンストラクターは、ログ出力機能を使用する場合や USB 接続で通信する場合に使用します。

構文

```
public Print(Context context)
```

パラメーター

- context：　アプリケーションのコンテキストを指定します。

例

```
import android.content.Context;

Print printer = new Print(getApplicationContext());

    ・・・処理・・・
```

Print クラス（コンストラクター）（旧フォーマット）

Print クラスのコンストラクターです。Print クラスのインスタンスを初期化します。

ログ出力機能や USB 接続での通信はできません。ログ出力機能を使用できません。ログ出力機能を使用する場合や USB 接続で通信する場合、Print クラス（コンストラクター）（105 ページ）を使用してください。

構文

```
public Print()
```

例

```
Print printer = new Print();

    ・・・処理・・・
```

openPrinter

プリンターとの通信・プリンターステータスのモニタリングを開始します。

プリンターとの通信が不要になった場合、必ず closePrinter（113 ページ）を呼び出し、プリンターとの通信を終了してください。

- プリンターステータスは、Print クラスで登録したイベントに通知されます。
  詳細は、プリンターステータスを自動で取得（38 ページ）を参照してください。
- プリンターステータスのモニタリングをやめたい場合、closePrinter（113 ページ）を呼び出してください。
- 一台のプリンターを複数のモバイル端末から使用する場合、注意事項（171 ページ）を参照してください。

構文

```
public void openPrinter
(int deviceType, String deviceName, int enabled,
 int interval, int timeout) throws EposException
```

パラメーター

- deviceType：　通信を開始するデバイスの種別を指定します。

設定値	説明
Print.DEVTYPE_TCP	Wi-Fi/Ethernet デバイス
Print.DEVTYPE_*BLUETOOTH*	*Bluetooth* デバイス
Print.DEVTYPE_USB	USB デバイス

- deviceName：対象デバイスを特定するための識別子を指定します。deviceType ごとに以下を指定します。

deviceType	設定値
Print.DEVTYPE_TCP	以下のいずれかを指定できます。 • IPv4 形式の IP アドレス (例 :"192.168.192.168") • MAC アドレス (例 :"01:23:45:67:89:AB") • プリンターホスト名 (任意の文字列)
Print.DEVTYPE_*BLUETOOTH*	BD アドレス (例 :"01:23:45:67:89:AB")
Print.DEVTYPE_USB	USB デバイスノード

- プリンターの IP アドレスを DHCP に設定している場合、deviceName に Mac アドレスまたはプリンターホスト名を指定してください。
- deviceType が Print.DEVTYPE_TCP で、deviceName にプリンターホスト名を指定する場合、DNS サーバーからプリンターホスト名が検索可能な環境で使用してください。

- enabled： プリンターステータスのモニタリングの有効・無効を指定します。

設定値	設定値
Print.TRUE	有効
Print.FALSE	無効
Print.PARAM_DEFAULT	既定値 (無効) を選択

- interval： プリンターステータスを更新する間隔 (ミリ秒単位) を指定します。

設定値	設定値
1000 ～ 60000 の整数	プリンターステータスを更新する間隔 (ミリ秒単位)
Print.PARAM_DEFAULT	既定値 (1000) を指定

- timeout： プリンターと通信確立するための最大待ち時間 (ミリ秒単位) を指定します。

設定値	設定値
1000 ～ 300000 の整数	エラーを返すまでの最大待ち時間 (ミリ秒単位)
Print.PARAM_DEFAULT	既定値 (15000) を指定

- 指定したデバイスが存在しない場合、ただちにエラーを返します。
- deviceType が Print.DEVTYPE_TCP で、指定したデバイスがすでに使用されている場合、タイムアウト時間まで本 API を再試行します。
- *Bluetooth*・USB 通信する場合、Print.PARAM_DEFAULT を設定してください。

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_OPEN	ポートオープン処理に失敗した。
ERR_TIMEOUT	指定したデバイスがすでに使用されていて、タイムアウト時間内にプリンターと通信確立できなかった。
ERR_ILLEGAL	すでに通信が開始されているデバイスを再度通信開始しようとした。
ERR_PROCESSING	処理を実行できなかった。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

IP アドレスが″192.168.192.168″のプリンターと Wi-Fi/Ethernet でプリンターステータスのモニタリングを有効にして通信を開始する場合

```
Print printer = new Print();
try {
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                        Print.PARAM_DEFAULT, Print.PARAM_DEFAULT);
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

openPrinter (旧フォーマット)

プリンターとの通信・プリンターステータスのモニタリングを開始します。

プリンターとの通信が不要になった場合、必ず closePrinter (113 ページ) を呼び出し、プリンターとの通信を終了してください。

- 本 API のタイムアウト時間は設定できません。本 API のタイムアウト時間を設定したい場合、openPrinter (107 ページ) を使用してください。
- プリンターステータスは、Print クラスで登録したイベントに通知されます。
  詳細は、プリンターステータスを自動で取得 (38 ページ) を参照してください。
- プリンターステータスのモニタリングをやめたい場合、closePrinter (113 ページ) を呼び出してください。
- 他のアプリケーションがプリンターをオープンしている場合、接続形式によって下記の注意が必要です。
  * TCP 接続 : 本 API を 15 秒間再試行します。15 秒後に ERR_OPEN が返されます。
  * *Bluetooth* 接続 : 本 API で通信を開始しようとすると、処理が戻ってこない場合があります。
- 一台のプリンターを複数のモバイル端末から使用する場合、注意事項 (171 ページ) を参照してください。

構文

```
public void openPrinter
(int deviceType, String deviceName, int enabled,
 int interval) throws EposException
```

パラメーター

- deviceType :　通信を開始するデバイスの種別を指定します。

設定値	説明
Print.DEVTYPE_TCP	Wi-Fi/Ethernet デバイス
Print.DEVTYPE_*BLUETOOTH*	*Bluetooth* デバイス
Print.DEVTYPE_USB	USB デバイス

- deviceName : 対象デバイスを特定するための識別子を指定します。deviceType ごとに以下を指定します。

deviceType	設定値
Print.DEVTYPE_TCP	以下のいずれかを指定できます。 • IPv4 形式の IP アドレス (例 :"192.168.192.168") • MAC アドレス (例 :"01:23:45:67:89:AB") • プリンターホスト名 (任意の文字列)
Print.DEVTYPE_*BLUETOOTH*	BD アドレス (例 :"01:23:45:67:89:AB")
Print.DEVTYPE_USB	デバイスノード

- プリンターの IP アドレスを DHCP に設定している場合、deviceName に Mac アドレスまたはプリンターホスト名を指定してください。
- deviceType が Print.DEVTYPE_TCP で、deviceName にプリンターホスト名を指定する場合、DNS サーバーからプリンターホスト名が検索可能な環境で使用してください。

- enabled： プリンターステータスのモニタリングの有効・無効を指定します。

設定値	設定値
Print.TRUE	有効
Print.FALSE	無効
Print.PARAM_DEFAULT	既定値 (無効) を選択

- interval： プリンターステータスを更新する間隔 (ミリ秒単位) を指定します。

設定値	設定値
1000 ～ 60000 の整数	プリンターステータスを更新する間隔 (ミリ秒単位)
Print.PARAM_DEFAULT	既定値 (1000) を指定

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_OPEN	• ポートオープン処理に失敗した。 • プリンターがすでに使われていた。
ERR_ILLEGAL	すでに通信が開始されているデバイスを再度通信開始しようとした。
ERR_PROCESSING	処理を実行できなかった。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

IP アドレスが″192.168.192.168″ のプリンターと Wi-Fi/Ethernet でプリンターステータスのモニタリングを有効にして通信を開始する場合

```
Print printer = new Print();
try {
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                        Print.PARAM_DEFAULT);
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

openPrinter (旧フォーマット)

プリンターとの通信を開始します。プリンターステータスの取得はできません。

プリンターとの通信が不要になった場合、必ず closePrinter (113 ページ) を呼び出し、プリンターとの通信を終了してください。

- 本 API のタイムアウト時間は設定できません。本 API のタイムアウト時間を設定したい場合、openPrinter (107 ページ) を使用してください。
- プリンターステータスを自動で取得したい場合、openPrinter (107 ページ) を使用してください。
- 他のアプリケーションがプリンターをオープンしている場合、接続形式によって下記の注意が必要です。
  * TCP 接続 : 本 API を 15 秒間再試行します。15 秒後に ERR_OPEN が返されます。
  * *Bluetooth* 接続 : 本 API で通信を開始しようとすると、処理が戻ってこない場合があります。
- 一台のプリンターを複数のモバイル端末から使用する場合、注意事項 (171 ページ) を参照してください。

構文

```
public void openPrinter
(int deviceType, String deviceName) throws EposException
```

パラメーター

- deviceType :　通信を開始するデバイスの種別を指定します。

設定値	説明
Print.DEVTYPE_TCP	Wi-Fi/Ethernet デバイス
Print.DEVTYPE_*BLUETOOTH*	*Bluetooth* デバイス
Print.DEVTYPE_USB	USB デバイス

- deviceName : 対象デバイスを特定するための識別子を指定します。deviceType ごとに以下を指定します。

deviceType	設定値
Print.DEVTYPE_TCP	以下のいずれかを指定できます。 • IPv4 形式の IP アドレス (例 :"192.168.192.168") • MAC アドレス (例 :"01:23:45:67:89:AB") • プリンターホスト名 (任意の文字列)
Print.DEVTYPE_*BLUETOOTH*	BD アドレス (例 :"01:23:45:67:89:AB")
Print.DEVTYPE_USB	デバイスノード

- プリンターの IP アドレスを DHCP に設定している場合、deviceName に Mac アドレスまたはプリンターホスト名を指定してください。
- deviceType が Print.DEVTYPE_TCP で、deviceName にプリンターホスト名を指定する場合、DNS サーバーからプリンターホスト名が検索可能な環境で使用してください。

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_OPEN	• ポートオープン処理に失敗した。 • プリンターがすでに使われていた。
ERR_ILLEGAL	すでに通信が開始されているデバイスを再度通信開始しようとした。
ERR_PROCESSING	処理を実行できなかった。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

IP アドレスが〝192.168.192.168〟のプリンターと Wi-Fi/Ethernet で通信を開始する場合

```
Print printer = new Print();
try {
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");
    ・・・処理・・・
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

closePrinter

プリンターとの通信、およびプリンターステータスのモニタリングを終了します。

構文

```
public void closePrinter() throws EposException
```

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_ILLEGAL	通信が開始されていない状態で、本 API が呼び出された。
ERR_PROCESSING	処理を実行できなかった。
ERR_FAILURE	その他のエラーが発生した。

例

```
Print printer = new Print();
try {
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");
    ・・・処理・・・
    printer.closePrinter();
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
}
```

sendData

Builder クラスで作成した印刷ドキュメントを送信します。

- *Bluetooth* 接続の場合、オフライン状態が検出できずタイムアウトエラーになることがあります。
- 一台のプリンターを複数のモバイル端末から使用する場合、注意事項（171 ページ）を参照してください。

構文

```
public void sendData(Builder builder, int timeout
, int[] status, int[] battery) throws EposException
```

パラメーター

- builder：　Builderクラスのインスタンスを指定します。Builderクラスの詳細は、Builderクラス（47ページ）を参照してください。
- timeout：　送受信待ちのタイムアウト時間を指定します。機種の仕様、通信インターフェイス、送信データサイズによって、timeout 時間を調整してください。
  0 ～ 600000（ミリ秒単位）の整数値を指定します。
- status：　コマンド送信終了時のプリンターステータスがセットされます。プリンターステータスの設定値の組み合わせがセットされます。詳細は、プリンターステータスと対処方法（45 ページ）を参照してください。
- battery：　コマンド送信終了時のバッテリーステータスがセットされます。
  詳細は、プリンター別サポート情報（162 ページ）のバッテリーステータスを参照してください。

例外が発生した場合、プリンターステータスは例外処理で getPrinterStatus（134 ページ）、バッテリーステータスは getBatteryStatus（135 ページ）を使って取得します。

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_ILLEGAL	通信が開始していない状態で本 API が呼び出された。
ERR_PROCESSING	処理を実行できなかった。
ERR_TIMEOUT	指定された時間内に全データを送信できなかった。
ERR_CONNECT	通信エラーが発生した。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_OFF_LINE	プリンターがオフライン状態だった。
ERR_FAILURE	その他のエラーが発生した。

例

タイムアウトに 10 秒を指定し、プリンターにコマンドを送信する場合

```
Print printer = new Print();
int[] status = new int[1];
int[] battery = new int[1];
status[0] = 0;
battery[0] = 0;

try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addText("ABCDE");

    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");
    printer.sendData(builder, 10000, status, battery);
    printer.closePrinter();
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
    battery[0] = e.getBatteryStatus();
}
```

sendData（旧フォーマット）

Builder クラスで作成した印刷ドキュメントを送信します。バッテリーステータスは、取得できません。

- *Bluetooth* 接続の場合、オフライン状態が検出できずタイムアウトエラーになることがあります。
- バッテリーステータスを取得できません。印刷ドキュメント送信時に、バッテリーステータスを取得したい場合、sendData（114 ページ）を使用してください。
- 一台のプリンターを複数のモバイル端末から使用する場合、注意事項（171 ページ）を参照してください。

構文

```
public void sendData(Builder builder, int timeout
, int[] status) throws EposException
```

パラメーター

- builder：　Builderクラスのインスタンスを指定します。Builderクラスの詳細は、Builderクラス（47ページ）を参照してください。
- timeout：　送受信待ちのタイムアウト時間を指定します。機種の仕様、通信インターフェイス、送信データサイズによって、timeout 時間を調整してください。
  0 ～ 600000（ミリ秒単位）の整数値を指定します。
- status：　コマンド送信終了時のプリンターステータスがセットされます。プリンターステータスの設定値の組み合わせがセットされます。詳細は、プリンターステータスと対処方法（45 ページ）を参照してください。

例外が発生した場合、プリンターステータスは例外処理で getPrinterStatus（134 ページ）を使って取得します。

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_ILLEGAL	通信が開始していない状態で本 API が呼び出された。
ERR_PROCESSING	処理を実行できなかった。
ERR_TIMEOUT	指定された時間内に全データを送信できなかった。
ERR_CONNECT	通信エラーが発生した。
ERR_MEMORY	処理に必要なメモリーが確保できなかった。
ERR_OFF_LINE	プリンターがオフライン状態だった。
ERR_FAILURE	その他のエラーが発生した。

例

タイムアウトに 10 秒を指定し、プリンターにコマンドを送信する場合

```
Print printer = new Print();
int[] status = new int[1];
status[0] = 0;

try {
    Builder builder = new Builder("TM-T88V", Builder.MODEL_JAPANESE);
    builder.addText("ABCDE");

    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");
    printer.sendData(builder, 10000, status);
    printer.closePrinter();
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
}
```

setStatusChangeEventCallback

プリンターステータスのイベントの通知先を登録します。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setStatusChangeEventCallback
            (StatusChangeEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface StatusChangeEventListener
                 extends EventListener
```

リスナー登録メソッド

```
void onStatusChangeEvent(String deviceName, int status)
```

パラメーター

- deviceName： プリンターステータスを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。
- status： プリンターステータスがセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, StatusChangeEventListener {
    ・・・処理・・・
    private void onStatusChangeEvent(String deviceName, int status) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setStatusChangeEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setOnlineEventCallback

オンラインイベントの通知先を登録します。プリンターステータスがオンライン時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setOnlineEventCallback
            (OnlineEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface OnlineEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onOnlineEvent(String deviceName)
```

パラメーター

- deviceName：オンラインイベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, OnlineEventListener {
    ・・・処理・・・
    private void onOnlineEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setOnlineEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setOfflineEventCallback

オフラインイベントの通知先を登録します。プリンターステータスがオフライン時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setOfflineEventCallback
            (OfflineEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface OfflineEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onOfflineEvent(String deviceName)
```

パラメーター

- deviceName： オフラインイベントを通知した、デバイスの識別子（IPv4形式のIPアドレス / BDアドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, OfflineEventListener {
    ・・・処理・・・
    private void onOfflineEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setOfflineEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setPowerOffEventCallback

無応答イベントの通知先を登録します。プリンターステータスが無応答時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setPowerOffEventCallback
            (PowerOffEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface PowerOffEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onPowerOffEvent(String deviceName)
```

パラメーター

- deviceName：無応答イベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, PowerOffEventListener {
    ・・・処理・・・
    private void onPowerOffEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setPowerOffEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setCoverOkEventCallback

カバークローズイベントの通知先を登録します。プリンターステータスがカバークローズ時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setCoverOkEventCallback
            (CoverOkEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface CoverOkEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onCoverOkEvent(String deviceName)
```

パラメーター

- deviceName：カバークローズイベントを通知した、デバイスの識別子（IPv4形式のIPアドレス / BDアドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                  OnClickListener, CoverOkEventListener {
    ・・・処理・・・
    private void onCoverOkEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setCoverOkEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                        Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setCoverOpenEventCallback

カバーオープンイベントの通知先を登録します。プリンターステータスがカバーオープン時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setCoverOpenEventCallback
            (CoverOpenEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface CoverOpenEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onCoverOpenEvent(String deviceName)
```

パラメーター

- deviceName：カバーオープンイベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                  OnClickListener, CoverOpenEventListener {
    ・・・処理・・・
    private void onCoverOpenEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setCoverOpenEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setPaperOkEventCallback

用紙ありイベントの通知先を登録します。プリンターステータスが用紙あり時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setPaperOkEventCallback
            (PaperOkEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface PaperOkEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onPaperOkEvent(String deviceName)
```

パラメーター

- deviceName： 用紙ありイベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, PaperOkEventListener {
    ・・・処理・・・
    private void onPaperOkEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setPaperOkEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setPaperNearEndEventCallback

用紙残量少イベントの通知先を登録します。プリンターステータスが用紙残量少時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setPaperNearEndEventCallback
            (PaperNearEndEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface PaperNearEndEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onPaperNearEndEvent(String deviceName)
```

パラメーター

- deviceName：用紙残量少イベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, PaperNearEndEventListener {
    ・・・処理・・・
    private void onPaperNearEndEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setPaperNearEndEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setPaperEndEventCallback

用紙なしイベントの通知先を登録します。プリンターステータスが用紙なし時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setPaperEndEventCallback
            (PaperEndEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface PaperEndEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onPaperEndEvent(String deviceName)
```

パラメーター

- deviceName： 用紙なしイベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                    OnClickListener, PaperEndEventListener {
    ・・・処理・・・
    private void onPaperEndEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setPaperEndEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setDrawerClosedEventCallback

ドロアークローズイベントの通知先を登録します。プリンターステータスがドロアークローズ時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setDrawerClosedEventCallback
            (DrawerClosedEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface DrawerClosedEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onDrawerClosedEvent(String deviceName)
```

パラメーター

- deviceName：ドロアークローズイベントを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                  OnClickListener, DrawerClosedEventListener {
    ・・・処理・・・
    private void onDrawerClosedEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setDrawerClosedEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setDrawerOpenEventCallback

ドロアーオープンイベントの通知先を登録します。プリンターステータスがドロアーオープン時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setDrawerOpenEventCallback
            (DrawerOpenEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface DrawerOpenEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onDrawerOpenEvent(String deviceName)
```

パラメーター

- deviceName： ドロアーオープンイベントを通知した、デバイスの識別子（IPv4形式のIPアドレス / BDアドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                  OnClickListener, DrawerOpenEventListener {
    ・・・処理・・・
    private void onDrawerOpenEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setDrawerOpenEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setBatteryLowEventCallback

バッテリー残量なしイベントの通知先を登録します。プリンターステータスがバッテリー残量によるオフライン時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setBatteryLowEventCallback
            (BatteryLowEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface BatteryLowEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void onBatteryLowEvent(String deviceName)
```

パラメーター

- deviceName：バッテリー残量なしイベントを通知した、デバイスの識別子（IPv4形式のIPアドレス / BDアドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                                  OnClickListener, BatteryLowEventListener {
    ・・・処理・・・
    private void onBatteryLowEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setBatteryLowEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setBatteryOkEventCallback

バッテリー残量ありイベントの通知先を登録します。プリンターステータスがバッテリー残量によるオフラインから復帰した時に通知されるイベントです。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setBatteryOkEventCallback
            (BatteryOkEventListener target)
```

パラメーター

- target： 通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface BatteryOkEventListener extends
                 EventListener
```

リスナー登録メソッド

```
void BatteryOkEventListener(String deviceName)
```

パラメーター

- deviceName：バッテリー残量ありイベントを通知した、デバイスの識別子（IPv4形式のIPアドレス / BDアドレス / デバイスノード / プリンターホスト名）がセットされます。

例

```
public class SampleActivity extends Activity implements
                          OnClickListener, BatteryOkEventListener {
    ・・・処理・・・
    private void onBatteryOkEvent(String deviceName) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setBatteryOkEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                    Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

setBatteryStatusChangeEventCallback

バッテリーステータスのイベントの通知先を登録します。

- 本 API は、openPrinter（107 ページ）の実行後でも実行できます。
- 本 API を複数回実行した場合、後に指定された通知先で上書きされます。

構文

```
public void setBatteryStatusChangeEventCallback
            (BatteryStatusChangeEventListener target)
```

パラメーター

- target：　通知先メソッド（リスナー登録メソッド）を持つオブジェクト（リスナーインターフェイス）を指定します。null を指定した場合、通知先の登録が解除されます。

リスナーインターフェイス

```
public interface BatteryStatusChangeEventListener
                 extends EventListener
```

リスナー登録メソッド

```
void onBatteryStatusChangeEvent
                (String deviceName, int battery)
```

パラメーター

- deviceName：バッテリーステータスを通知した、デバイスの識別子（IPv4 形式の IP アドレス / BD アドレス / デバイスノード / プリンターホスト名）がセットされます。
- battery：　バッテリーステータスがセットされます。

例

```
public class SampleActivity extends Activity implements
                          OnClickListener, BatteryStatusChangeEventListener {
    ・・・処理・・・
    private void onBatteryStatusChangeEvent(String deviceName, int battery) {
        ・・・処理・・・
    }
    private void openPrinter() {
        Print printer = new Print();
        printer.setBatteryStatusChangeEventCallback(this);
        try {
            printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168", Print.TRUE,
                      Print.PARAM_DEFAULT);
            ・・・処理・・・
        } catch (EposException e) {
            int errStatus = e.getErrorStatus();
        }
    }
}
```

getErrorStatus

例外からエラーステータスを取得します。

構文

```
public int getErrorStatus()
```

戻り値

例外発生元の API で設定されたエラーステータスが返ります。

例

EposException からエラーステータスを取得する場合

```
try {
    printer.openPrinter(Print.DEVTYPE_TCP, "192.168.192.168");
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    if ( errStatus == EposException.ERR_OPEN) {
        ・・・処理・・・
    }
}
```

getPrinterStatus

sendData（114 ページ）で発生した例外からプリンターステータスを取得します。

構文

```
public int getPrinterStatus()
```

戻り値

プリンターステータスが返ります。プリンターステータスの設定値の組み合わせが返ります。
詳細は、プリンターステータスと対処方法（45 ページ）を参照してください。

例

EposException からプリンターステータスを取得する場合

```
int[] printerStatus = new int[1];
printerStatus[0] = 0;
int timeout = 1000;

try {
    printer.sendData(builder, timeout, printerStatus);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    if (errStatus == EposException.ERR_TIMEOUT) {
        printerStatus[0] = e.getPrinterStatus();
    }
}

if ((printerStatus[0] & Print.ST_PRINT_SUCCESS) == Print.ST_PRINT_SUCCESS)
{
    ・・・処理・・・
}
```

getBatteryStatus

sendData（114 ページ）で発生した例外からバッテリーステータスを取得します。

構文

```
public int getBatteryStatus()
```

戻り値

バッテリーステータスが返ります。詳細は、プリンター別サポート情報（162 ページ）を参照してください。

例

EposException からバッテリーステータスを取得する場合

```
int[] printerStatus = new int[1];
int[] batteryStatus = new int[1];
printerStatus[0] = 0;
batteryStatus[0] = 0;
int timeout = 1000;

try {
    printer.sendData(builder, timeout, printerStatus, batteryStatus);
} catch (EposException e) {
    int errStatus = e.getErrorStatus();
    if (errStatus == EposException.ERR_TIMEOUT) {
        printerStatus[0] = e.getPrinterStatus();
        batteryStatus[0] = e.getBatteryStatus();
    }
}

if ((printerStatus[0] & Print.ST_PRINT_SUCCESS) == Print.ST_PRINT_SUCCESS)
{
    ・・・処理・・・
}
```

プリンター検索 API

プリンターを検索するための API です。以下のクラスが用意されています。

❏ Finder クラス（136 ページ）

❏ EpsonIoException クラス（49 ページ）

Finder クラス

プリンターを検索するクラスです。以下の API が用意されています。

API	説明	ページ
start	プリンター検索を開始	137
stop	プリンターとの通信を終了	138
getDeviceInfoList	プリンターの検索結果を取得	139
getResult (旧フォーマット)		141

EpsonIoException クラス

Finder クラスおよび Epsonlo クラス（155 ページ）の API 呼び出し時に発生した、例外のエラー値を通知するクラスです。以下の API が用意されています。

API	説明	ページ
getStatus	例外のエラー値を取得	142

start

指定されたデバイス種別のプリンター検索を開始します。

本 API を使用したら、必ず stop（138 ページ）で検索終了してください。

すでにプリンター検索を開始している状態で、本 API を呼び出すことはできません。

構文

```
public static synchronized void start
            (Context context, int deviceType,
             String findOption)
             throws EpsonIoException
```

パラメーター

- context：　呼び元の Context クラスのインスタンスを設定します。
  （例：Activity 内で getBaseContext() で得た Context を設定）
- deviceType：　検索するデバイス種別を指定します。以下の値を指定します。

deviceType	説明
DevType.TCP	ネットワークに接続された TM デバイスを検索します
DevType.*BLUETOOTH*	Printer または Uncategorized のデバイスクラスを持つ、*Bluetooth* デバイスを検索します。
DevType.USB	VID と PID から USB デバイスを検索します。 検索条件 USB に接続された TM デバイスを検索します。

- findOption：　対象デバイスを検索する際の設定値を指定します。

deviceType	指定する値
DevType.TCP	検索範囲のブロードキャストアドレス
DevType.*BLUETOOTH*	"null"
DevType.USB	"null"

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	すでに検索を開始した状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

stop

プリンター検索を終了します。

構文

```
public static synchronized void stop()
                throws EpsonIoException
```

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	検索を開始していない状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

getDeviceinfolist

本 API を呼び出した時点までの、デバイスの検索結果を取得します。

オープン済みの *Bluetooth* デバイスや USB デバイスは、取得できません。

構文

```
public static synchronized final DeviceInfo[]
              getDeviceInfoList(int filterOption)
              throws EpsonIoException
```

パラメーター

- filterOption：　エプソン製プリンターのフィルタリング方法を指定します。以下の値を指定します。

設定値	説明
FilterOption.FILTER_NONE	フィルタリングしない
FilterOption.FILTER_NAME	プリンター名でフィルタリングする
FilterOption.PARAM_DEFAULT	既定値 (プリンター名でフィルタリングする)

TCP /USB 接続の場合、FilterOption の設定に関係なく、エプソン製プリンターのみ検索されます。

戻り値

検索されたデバイスのデバイス情報リスト (DeviceInfo[]) が返されます。
リスト内には、デバイス情報が DeviceInfo 型の配列で格納されています。
デバイス種別 (deviceType) によって、格納される情報が異なります。

deviceType	DeviceInfo	取得する情報
DevType.TCP	getDeviceType()	DevType.TCP(固定)
	getPrinterName()	プリンターモデル名
	getDeviceName()	• DHCP 無効の場合 : IP アドレス • DHCP 有効の場合 : MAC アドレス
	getIpAddress()	IP アドレス
	getMacAddress()	MAC アドレス
DevType.*BLUETOOTH*	getDeviceType()	DevType.*BLUETOOTH*(固定)
	getPrinterName()	*Bluetooth* デバイス名
	getDeviceName()	BD アドレス (MAC アドレスと同じ形式)
	getIpAddress()	"" (空文字)
	getMacAddress()	"" (空文字)
DevType.USB	getDeviceType()	DevType.USB(固定)
	getPrinterName()	USB デバイス名
	getDeviceName()	デバイスノード
	getIpAddress()	"" (空文字)
	getMacAddress()	"" (空文字)

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	検索を開始していない状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

getResult（旧フォーマット）

本 API を呼び出した時点までの、プリンターの検索結果を取得します。

オープン済みの *Bluetooth* デバイスや USB デバイスは、取得できません。

構文

```
public static synchronized final String[] getResult()
                throws EpsonIoException
```

戻り値

検索したデバイスのリストが返されます。

リスト内には、検索したデバイスの識別情報が、文字列 (String 型) で格納されています。

デバイス種別 (deviceType) によって、格納される結果が異なります。

deviceType	取得するリスト
DevType.TCP	プリンターの IP アドレスのリスト
DevType.*BLUETOOTH*	*Bluetooth* デバイスの BD アドレスのリスト
DevType.USB	USB デバイスのデバイスノードのリスト

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	検索を開始していない状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

getStatus

例外のエラー値を取得します。

構文

```
public int getStatus();
```

戻り値

例外で発生したエラー値が返ります。エラー値は、IoStatus クラスに定義されています。

エラー値	説明
IoStatus.ERR_PARAM	不正なパラメーターが渡された。
IoStatus.ERR_MEMORY	処理に必要なメモリーが確保できなかった。
IoStatus.ERR_ILLEGAL	不適切な方法で使用された。
IoStatus.ERR_PROCESSING	処理を実行できなかった。
IoStatus.ERR_FAILURE	その他のエラーが発生した。

プリンター簡単選択 API

NFC と QR コードを使ってプリンターを選択する際に使う API です。NFC と QR コードから取得したデータを openPrinter に渡せる形に変換します。以下のクラスが用意されています。

❏ EasySelect クラス（143 ページ）
❏ EasySelectInfo クラス（143 ページ）

EasySelect クラス

NFC データと QR コードデータを解析します。以下の API が用意されています。

API	説明	ページ
parseNFC	NFC タグデータの解析	144
parseQR	QR コードデータの解析	144
createQR	簡単選択用 QR コード印刷データの作成	145

EasySelectInfo クラス

EasySelect クラスで解析したデータを格納し、openPrinter に渡す変数に変換するクラスです。
以下のメンバー変数が用意されています。

メンバー変数	説明	ページ
deviceType	解析結果のデバイス種別	146
printerName	解析結果のプリンター名	146
macAddress	解析結果の MAC アドレス / BD アドレス	146

parseNFC

NFC タグのデータを解析します。

構文

```
public EasySelectInfo parseNFC(Tag tag)
```

パラメーター

- tag：　NFC タグデータを指定します。

解析タグ	説明
Wi-FI データ	独自定義データ (Wi-Fi 用)
BTSSP	NFC 標準規格 (*Bluetooth* 用)

戻り値

NFC の解析結果が返されます。EasySelectInfo クラスに格納します。
解析に失敗した場合、null が返されます。

parseQR

文字列の QR コードのデータを解析します。

構文

```
public EasySelectInfo parseQR(String data)
```

パラメーター

- data：　QR コードの文字列データを指定します。

戻り値

QR コードの文字列データの解析結果が返されます。EasySelectInfo クラスに格納します。
解析に失敗した場合、null が返されます。

createQR

簡単選択用 QR コードの印刷データを作成します。

構文

```
public String createQR(String printerName,
                       int deviceType,
                       String macAddress)
```

パラメーター

- printerName：プリンター名を指定します。
- deviceType：デバイスの種別を指定します。以下を指定します。

設定値	説明
Print.DEVTYPE_TCP	Wi-Fi/ Ethernet デバイス
Print.DEVTYPE_*BLUETOOTH*	*Bluetooth* デバイス

- macAddress：BD アドレスを指定します。
  BD アドレスは、以下のフォーマットに対応しています。

フォーマット	説明
00:11:22:33:44:55	":" コロン区切り
00-11-22-33-44-55	"-" ハイフン区切り
001122334455	区切りなし

戻り値

簡単選択用 QR コードの印刷データが返されます。印刷データの作成に失敗した場合、null が返されます。

deviceType

解析結果のデバイス種別を格納します。

格納されるデータ	説明
Print.DEVTYPE_TCP	Wi-Fi/ Ethernet デバイス
Print.DEVTYPE_*BLUETOOTH*	NFC 標準規格 (*Bluetooth* 用)

書式

```
int deviceType;
```

printerName

解析結果のプリンター名を格納します。

書式

```
String printerName;
```

macAddress

解析結果の BD アドレスを格納します。

書式

```
String macAddress;
```

ログ設定 API

ログ出力の設定をします。以下のクラスが用意されています。

❏ Log クラス（147 ページ）

Log クラス

ログの出力機能を設定します。

API	説明	ページ
setLogSettings	ログ出力機能の設定	147

setLogSettings

ログ出力機能を設定します。

構文

```
public static void setLogSettings(Context context,
            int period, int enabled, String ipAddress,
            int port, int logSize, int logLevel)
             throws EposException
```

パラメーター

- context：　アプリケーションのコンテキストを指定します。
- period：　ログ出力機能の設定方法を指定します。

設定値	説明
Log.LOG_TEMPORARY	アプリケーションを終了すると、本 API の設定は無効になります。
Log.LOG_PERMANENT	アプリケーションを終了させても、本 API の設定を有効にします。

period を Log.LOG_PERMANENT に指定する場合、アプリケーションにストレージアクセスのパーミッションを設定してください。

- enabled：　ログ出力機能の有効 / 無効、およびログの出力先を指定します。

設定値	説明
Log.LOG_DISABLE	ログ出力機能を無効にする。
Log.LOG_STORAGE	端末のストレージに出力する。
Log.LOG_TCP	TCP で出力する。

- enabled を Log.LOG_STORAGE に指定する場合、アプリケーションにストレージアクセスのパーミッションを設定してください。
- enabled を Log.LOG_TCP に指定する場合、アプリケーションにネットワークアクセスのパーミッションを設定してください。

- ipAddress： TCP 通信の IP アドレス (IPv4 形式) を指定します。

> enabled が以下の値の場合、"null" も指定できます。
> * Log.LOG_DISABLE
> * Log.LOG_STORAGE

- port： TCP 通信用のポート番号を指定します。0 ～ 65535 の整数値を指定します。

> enabled に以下の値を指定した場合も、範囲内の任意の値を指定してください。
> * Log.LOG_DISABLE
> * Log.LOG_STORAGE

- logSize： 端末のストレージへ保存する、ログの最大容量を指定します。
  1 ～ 50(MB 単位) の整数値を指定します。

> enabled に以下の値を指定した場合も、範囲内の任意の値を指定してください。
> * Log.LOG_DISABLE
> * Log.LOG_TCP

- logLevel： ログの出力レベルを指定します。

設定値	説明
Log.LOG_LOW	低レベル

例外

処理に失敗した場合、以下のエラー値の EposException が発生します。

エラーステータス	説明
ERR_PARAM	不正なパラメーターが渡された。
ERR_FAILURE	その他のエラーが発生した。

例

TCP で、IP アドレス 192.168.192.168 の 8080 番ポートにログを出力する場合

```
try {
  Log.setLogSettings(getApplicationContext(), Log.LOG_PERMANENT,
                     Log.LOG_TCP, "192.168.192.168", 8080, 10, Log.LOG_LOW);
} catch (EposException e) {
int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
}
```

端末のストレージにログを出力する場合

```
try {
  Log.setLogSettings(getApplicationContext(), Log.LOG_PERMANENT,
                     Log.LOG_STORAGE, null,0, 10, Log.LOG_LOW);
} catch (EposException e) {
int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
}
```

ログ出力機能を無効にする場合

```
try {
  Log.setLogSettings(getApplicationContext(), Log.LOG_PERMANENT,
                     Log.LOG_DISABLE, null, 0, 10, Log.LOG_LOW);
} catch (EposException e) {
int errStatus = e.getErrorStatus();
    status[0] = e.getPrinterStatus();
}
```

ログファイルの取り出し方法

保存先

Android バージョン	保存先
Android Version 4.1 以前	/(storage パス)/Android/data/(アプリケーションのパッケージ名)/files/EposLog < 例 > /storage/sdcard0/Android/data/com.example/files/EposLog
Android Version 4.2 以降	/storage/emulated/(ユーザーを指すインデックス)/Android/data/(アプリケーションのパッケージ名)/files/EposLog < 例 > /storage/emulated/0/Android/data/com.example/files/EposLog

ファイル名

❏ EposLog.xx

ログの見方

ログのフォーマット

ログのレコードは、以下の形式で構成されています。

≪ 日時 , プロセス ID: スレッド ID, 入出力階層 , 入出力方向 , 入出力データ ≫

項目	説明
日時	yyyy/mm/dd,h:mm:ss.000 の形式です。
プロセス ID: スレッド ID	各処理の ID です。
入出力階層	データを入出力している階層です。 • APIIO: アプリケーションから呼び出されるインターフェイス層 • IOCM/DEVIO: デバイスとの通信層
入出力方向	データの入出力方向です。 • ->: 階層からの入力です。 • <-: 階層からの出力です。
入出力データ	呼び出された API、パラメーターおよび通信データです。

各項目は、コンマ (,) で区切られてます。

出力例

アプリケーションから addCut メソッドを呼び出す場合 :

```
2014/07/28,20:12:35.836,00002ae9:00006008,APIIO,->,0x687bc5d8,,addCut,1
2014/07/28,20:12:35.836,00002ae9:00006008,APIIO,<-,0x687bc5d8,0,addCut}
```

コマンドの送受信

本章では、コマンド (ESC/POS コマンドなど) を送受信するための API について説明しています。

⚠	本章で説明している、コマンドを送受信するための API は、ESC/POS コマンドを熟知しているお客様向けの API です。
✎	コマンド送受信 API は、ePOS-Print API の Print クラス (49 ページ) と同時に使用できません。

プログラミング

プログラミングフロー

以下のフローでプログラミングします。

1. 事前準備 (USB 接続) (28 ページ) *

2. プリンターの選択 (29 ページ) *

< 検索機能を使ってプリンターを選択 >
- プリンターの検索結果を取得 (29 ページ)
- プリンターの検索結果を取得 (29 ページ)
- プリンターの検索を終了 (30 ページ)

<NFC を使ってプリンターを選択 >
- NFC を使ってプリンターを選択 (30 ページ)

<QR コードを使ってプリンターを選択 >
- QR コードを使ってプリンターを選択 (31 ページ)

3. デバイスポート のオープン (152 ページ)

4. データの送信 (152 ページ)

5. データの受信 (153 ページ)

6. デバイスポート のクローズ (153 ページ)

*: 任意のプロセスです。

デバイスポートのオープン

EpsonIo クラスの open（155 ページ）を使って、デバイスポートをオープンします。以下のプログラミングを参考にしてください。

```
//EpsonIo クラスの生成
EpsonIo mPort = new EpsonIo();
int errStatus = IoStatus.SUCCESS;

// デバイスポートのオープン
try {
////Wi-Fi/Ethernet デバイスの場合
    mPort.open(DevType.TCP, "192.168.192.168", null, null);
////Bluetooth デバイスの場合
    mPort.open(DevType.BLUETOOTH, "00:00:12:34:56:78", null, null);
////USB デバイスの場合
    mPort.open(DevType.USB, "/dev/bus/usb/001/002", null, getApplicationContext());

// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

データの送信

EpsonIo クラスの write（159 ページ）を使って、プリンターにデータを送信します。以下のプログラミングを参考にしてください。

文字列「Hello, World!」を印刷する場合

```
// 送信設定
String str = "Hello, World!\r\n";
byte[] data = str.getBytes();
int offset = 0;
int size = data.length;
int timeout = 5000;
int sizeWritten = 0;
int errStatus = IoStatus.SUCCESS;

try {
// データの送信
    sizeWritten = mPort.write(data, offset, size, timeout);
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

データの受信

EpsonIo クラスの read（160 ページ）を使って、プリンターからのデータを受信します。
以下のプログラミングを参考にしてください。

```
// 受信設定
byte[] data = new byte[256];
int offset = 0;
int size = 256;
int timeout = 5000;
int sizeRead = 0;
int errStatus = IoStatus.SUCCESS;

// データの受信
try {
    sizeRead = mPort.read(data, offset, size, timeout);
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

デバイスポートのクローズ

EpsonIo クラスの close（158 ページ）を使って、デバイスポートをクローズします。以下のプログラミングを参考にしてください。

```
int errStatus = IoStatus.SUCCESS;

// デバイスのクローズ
try {
    mPort.close();
// 例外処理
} catch ( EpsonIoException e ) {
    errStatus = e.getStatus();
}
```

例外処理

コマンド送受信 API は、エラー発生時に数値 (int) 型のパラメーターを持つ独自の例外を発生させ、呼び元にエラーを通知します。

処理方法

EpsonIoException クラス（136 ページ）クラスの getStatus でエラー値を取得します。以下のプログラミングを参考にしてください。

```
String str = "Hello, World!\r\n";
byte[] data = str.getBytes();
int offset = 0;
int size = data.length;
int timeout = 5000;
int sizeWritten = 0;
int errStatus = IoStatus.SUCCESS;

try {
    sizeWritten = mPort.write(data, offset, size, timeout);
} catch ( EpsonIoException e ) {
// エラー値の取得
    errStatus = e.getStatus();
}
```

エラー値一覧

エラー値は、IoStatus クラスに定義されています。

エラー値	要因
IoStatus.ERR_PARAM	不正なパラメーターが渡された。 < 例 > • null など、不正な引数を渡された。 • サポートしていない範囲の値が指定された。
IoStatus.ERR_OPEN	オープン処理に失敗した
IoStatus.ERR_CONNECT	デバイスとの通信に失敗した。 < 例 > • タイムアウト以外の要因で、対象デバイスへのデータ送信に失敗した。 • タイムアウト以外の要因で、対象デバイスからのデータ受信に失敗した。
IoStatus.ERR_MEMORY	処理に必要なメモリーが確保できなかった。
IoStatus.ERR_ILLEGAL	不適切な方法で使用された。 < 例 > • デバイスポートがオープンされていない状態でデータ送受信用 API が呼び出された。 • すでにプリンター検索が開始されている状態で、再度検索開始 API が呼び出された。
IoStatus.ERR_PROCESSING	処理を実行できなかった。 < 例 > 同様の処理を他のスレッドで実行中のため、共有リソースのロック権限を取得できなかった。
IoStatus.ERR_FAILURE	その他のエラーが発生した。

コマンド送受信 API リファレンス

コマンド送受信 API には以下のクラスが用意されています。

EpsonIo クラス

データ送受信用のクラスです。以下の API が用意されています。

API	説明	ページ
open	デバイスポートのオープン	155
open(旧フォーマット)	デバイスポートのオープン (USB 接続での通信はできません。)	157
close	デバイスポートのクローズ	158
write	データ送信	159
read	データ受信	160

open

指定されたデバイスポートをオープンします。

構文

```
public void open
            (int deviceType, String deviceName,
             String deviceSettings, Context context)
             throws EpsonIoException
```

パラメーター

- deviceType：　オープンするデバイス種別を指定します。以下の値を指定します。

deviceType	説明
DevType.TCP	オープンするプリンターが Wi-Fi/Ethernet の時に指定します。
DevType.*BLUETOOTH*	オープンするプリンターが *Bluetooth* の時に指定します。
DevType.USB	オープンするプリンターが USB の時に指定します。

- deviceName：対象デバイスを特定するための識別子を指定します。以下の値を指定します。

deviceType	指定する値
DevType.TCP	以下のいずれかを指定できます。 • IPv4 形式の IP アドレス (例 :"192.168.192.168") • MAC アドレス (例 :"01:23:45:67:89:AB") • プリンターホスト名 (任意の文字列)
DevType.*BLUETOOTH*	BD アドレス (例 :"01:23:45:67:89:AB")
DevType.USB	デバイスノード

- deviceSettings：
  ″null″ を指定します。
- context：　アプリケーションのコンテキストを指定します。

deviceType	指定する値
DevType.TCP	″null″
DevType.*BLUETOOTH*	″null″
DevType.USB	アプリケーションのコンテキスト

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_OPEN	オープン処理に失敗した
IoStatus.ERR_ILLEGAL	すでにオープンされているデバイスを再度オープンしようとした
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

open (旧フォーマット)

指定されたデバイスポートをオープンします。USB 接続での通信はできません。

USB 接続で通信する場合、open (155 ページ) を使用してください。

構文

```
public void open
              (int deviceType, String deviceName,
               String deviceSettings)
               throws EpsonIoException
```

パラメーター

- deviceType :　オープンするデバイス種別を指定します。以下の値を指定します。

deviceType	説明
DevType.TCP	オープンするプリンターが Wi-Fi/Ethernet の時に指定します。
DevType.*BLUETOOTH*	オープンするプリンターが *Bluetooth* の時に指定します。

- deviceName : 対象デバイスを特定するための識別子を指定します。以下の値を指定します。

deviceType	指定する値
DevType.TCP	以下のいずれかを指定できます。 • IPv4 形式の IP アドレス (例 :"192.168.192.168") • MAC アドレス (例 :"01:23:45:67:89:AB") • プリンターホスト名 (任意の文字列)
DevType.*BLUETOOTH*	BD アドレス (例 :"01:23:45:67:89:AB")

- deviceSettings :
  ″null″ を指定します。

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_OPEN	オープン処理に失敗した
IoStatus.ERR_ILLEGAL	すでにオープンされているデバイスを再度オープンしようとした
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

close

指定されたデバイスポートをクローズします。

構文

```
public void close() throws EpsonIoException
```

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	オープンしていない状態で本 API が呼び出された。
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

write

データをデバイスポートへ送信します。

構文

```
public int write
            (byte[] data, int offset, int size,
             int timeout)
             throws EpsonIoException
```

パラメーター

- data：　送信データのバッファーです。送信するデータを格納します。
- offset：　送信開始位置を指定します。
  送信データバッファーの先頭からのオフセット値を指定してください。
- size：　送信したいデータのバイト数を指定します。

size に 0 が指定された場合、送信されません。この場合、戻り値に 0 が返ります。

- timeout：　送信待ちのタイムアウト時間を、msec 単位で指定します。
  指定可能な最長時間は 600000 msec(10 分) です。

- timeout は、伝送速度、送信データ量などを考慮して指定してください。
- timeout が短すぎる場合、正常にデータが送信できている間、全データを送信し終えるまで timeout を超えても送信処理を継続します。
- *Bluetooth* のデバイスのとき、送信処理がブロックされる可能性があります。その場合、タイムアウト指定時間が経過しても処理は終了しません。

戻り値

送信を終了したデータのバイト数が返されます。

- 戻り値で返されるサイズのデータが、実際にプリンターが受信しているとは限りません。
- timeout で指定した時間を過ぎた場合、その時点までに送信を終了したバイト数を戻り値に返します。

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	オープンしていない状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_CONNECT	通信エラーが発生した
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

read

データをデバイスポートから受信します。

本 API は、受信エラーが発生するまでデータを受信し続けますが、timeout で指定された時間内に 1 バイトもデータが受信できなかった場合、処理が終了します。

構文

```
public int read
            (byte[] data, int offset, int size,
             int timeout)
             throws EpsonIoException
```

パラメーター

- data : 受信データの格納先バッファーです。
- offset : 格納先バッファーの格納開始位置を指定します。
  受信データバッファーの先頭からのオフセット値を指定します。
- size : 受信可能なバイト数を指定します。

size に 0 が指定された場合、受信されません。この場合、戻り値に 0 が返ります。

- timeout : データ受信する時間を、msec 単位で指定します。
  指定可能な最長時間は 600000 msec(10 分) です。

戻り値

受信したデータのバイト数が返されます。

例外

処理に失敗した場合、以下のエラー値の EpsonIoException が発生します。

エラー値	説明
IoStatus.ERR_ILLEGAL	オープンしていない状態で本 API が呼び出された
IoStatus.ERR_PROCESSING	処理を実行できなかった
IoStatus.ERR_PARAM	不正なパラメーターが渡された
IoStatus.ERR_CONNECT	通信エラーが発生した
IoStatus.ERR_MEMORY	メモリーを確保できなかった
IoStatus.ERR_FAILURE	その他のエラーが発生した

付録

プリンターごとのサポート API一覧

API	TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
addTextAlign （55 ページ）	○	○	○	○	○	○	○
addTextLineSpace （56 ページ）	○	○	○	○	○	○	○
addTextRotate （57 ページ）	○	○	○	○	○	○	○
addText （58 ページ）	○	○	○	○	○	○	○
addTextLang （59 ページ）	○	○	○	○	○	○	○
addTextFont （60 ページ）	○	○	○	○	○	○	○
addTextSmooth （61 ページ）	○	○	○	○	○	○	○
addTextDouble （62 ページ）	○	○	○	○	○	○	○
addTextSize （63 ページ）	○	○	○	○	○	○	○
addTextStyle （64 ページ）	○	○	○	○	○	○	○
addTextPosition （66 ページ）	○	○	○	○	○	○	○
addFeedUnit （67 ページ）	○	○	○	○	○	○	○
addFeedLine （68 ページ）	○	○	○	○	○	○	○
addImage （69 ページ）	○	○	○	○	○	○	○
addImage (旧フォーマット) （72 ページ）	○	○	○	○	○	○	○
addImage (旧フォーマット) （75 ページ）	○	○	○	○	○	○	○
addLogo （77 ページ）	○	○	○	○	○	○	○
addBarcode （78 ページ）	○	○	○	○	○	○	○
addSymbol （83 ページ）	○	○	○	○	○	○	○
addPageBegin （88 ページ）	○	○	○	○	○	○	○
addPageEnd （89 ページ）	○	○	○	○	○	○	○
addPageArea （90 ページ）	○	○	○	○	○	○	○
addPageDirection （91 ページ）	○	○	○	○	○	○	○
addPagePosition （92 ページ）	○	○	○	○	○	○	○
addPageLine （93 ページ）	○	○					
addPageRectangle （94 ページ）	○	○					
addCut （95 ページ）	○	○	○	○	○	○	○
addPulse （96 ページ）			○	○	○	○	○
addSound （97 ページ）	○	○	○		○	○	
addSound (旧フォーマット) （99 ページ）	○	○	○		○	○	
addFeedPosition （101 ページ）	○	○					
addLayout （102 ページ）	○	○					
addCommand （104 ページ）	○	○	○	○	○	○	○

プリンター別サポート情報

TM-P20

		58 mm 仕様
解像度		203 x 203 dpi
言語		• ANK モデル • 日本語モデル
印字幅		384 ドット
印字桁数	フォント A	ANK 32 桁 / 漢字 16 桁
	フォント B	ANK 42/ 漢字 19 桁
	フォント C	ANK 42 桁 / 漢字 24 桁
	フォント D	ANK 38 桁
	フォント E	ANK 48 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット
	フォント B	ANK 9x 24 ドット / 漢字 20 x 24 ドット
	フォント C	ANK 9 x 17 ドット / 漢字 16x 16 ドット
	フォント D	ANK 10 x 24 ドット
	フォント E	ANK 8 x 16 ドット
文字のベースライン	フォント A	文字の上端から 21 ドット目
	フォント B	文字の上端から 21 ドット目
	フォント C	文字の上端から 16 ドット目
	フォント D	文字の上端から 21 ドット目
	フォント E	文字の上端から 15 ドット目
初期改行量		30 ドット
色指定		第 1 色
ページモード初期領域		384 x 2400 ドット
ページモード最大領域		384 x 2400 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded

	58 mm 仕様
2 次元シンボル	PDF417, QR Code, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional, GS1 DataBar Expanded Stacked, Composite Symbology
用紙のカット	カット位置まで紙送り
ドロアーキック	非サポート
ブザー	オプション
バッテリー	サポート

バッテリーステータス

上位 8 ビット

バッテリーステータス	要因
0x30	AC アダプターが接続されている
0x31	AC アダプターが接続されていない

下位 8 ビット

バッテリーステータス	要因
0x30	バッテリー残量 0(リアルエンド)
0x31	バッテリー残量 1(ニアエンド)
0x32	バッテリー残量 2
0x33	バッテリー残量 3
0x34	バッテリー残量 4
0x35	バッテリー残量 5
0x36	バッテリー残量 6

バッテリーステータス取得不可能状態の場合、"0x0000" を返します。

TM-P60II

		58 mm 仕様	60 mm 仕様
解像度		203 x 203 dpi	
言語		• ANK モデル • 日本語モデル	
印字幅		420 ドット	432 ドット
印字桁数	フォント A	ANK 35 桁 / 漢字 17 桁	ANK 36 桁 / 漢字 18 桁
	フォント B	ANK 42 桁	ANK 43 桁
	フォント C	ANK 52 桁	ANK 54 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 10 x 24 ドット	
	フォント C	ANK 8 x 16 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 21 ドット目	
	フォント C	文字の上端から 15 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		420 x 1624 ドット	432 x 1624 ドット
ページモード最大領域		420 x 1624 ドット	432 x 1624 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded	
2 次元シンボル		PDF417, QR Code, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional, GS1 DataBar Expanded Stacked, Composite Symbology	
用紙のカット		カット / フィードカット	
ドロアーキック		非サポート	
ブザー		オプション	
バッテリー		サポート	

バッテリーステータス

上位 8 ビット

バッテリーステータス	要因
0x30	AC アダプターが接続されている
0x31	AC アダプターが接続されていない

下位 8 ビット

バッテリーステータス	要因
0x30	バッテリー残量 0(リアルエンド)
0x31	バッテリー残量 1(ニアエンド)
0x32	バッテリー残量 2
0x33	バッテリー残量 3
0x34	バッテリー残量 4
0x35	バッテリー残量 5
0x36	バッテリー残量 6

バッテリーステータス取得不可能状態の場合、"0x0000" を返します。

TM-T20II

		58 mm 仕様	80 mm 仕様
解像度		203 x 203 dpi	
言語		日本語モデル	
印字幅		420 ドット	576 ドット
印字桁数	フォント A	ANK 35 桁 / 漢字 17 桁	ANK 48 桁 / 漢字 24 桁
	フォント B	ANK 46 桁 / 漢字 23 桁	ANK 64 桁 / 漢字 32 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 9 x 17 ドット / 漢字 16 x 16 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 16 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		420 x 831 ドット	576 x 831 ドット
ページモード最大領域		420 x 1662 ドット	576 x 1662 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF,CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded	
2 次元シンボル		PDF417, QR Code, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional, GS1 DataBar Expanded Stacked, Composite Symbology	
用紙のカット		カット / フィードカット	
ドロアーキック		サポート	
ブザー		オプション	
バッテリー		非サポート	

TM-T70

		58 mm 仕様	80 mm 仕様
解像度		203 x 203 dpi	
言語		• ANK モデル • 日本語モデル	
印字幅		416 ドット	576 ドット
印字桁数	フォント A	ANK 34 桁 / 漢字 17 桁	ANK 48 桁 / 漢字 24 桁
	フォント B	ANK 52 桁 / 漢字 26 桁	ANK 72 桁 / 漢字 36 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 8 x 16 ドット / 漢字 16 x 16 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 15 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		416 x 1662 ドット	576 x 1662 ドット
ページモード最大領域		416 x 1662 ドット	576 x 1662 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128	
2 次元シンボル		QR Code	
用紙のカット		カット / フィードカット	
ドロアーキック		サポート	
ブザー		非サポート	
バッテリー		非サポート	

TM-T70II

		58 mm 仕様	80 mm 仕様
解像度		203 x 203 dpi	
言語		• ANK モデル • 日本語モデル	
印字幅		416 ドット	576 ドット
印字桁数	フォント A	ANK 34 桁 / 漢字 17 桁	ANK 48 桁 / 漢字 24 桁
	フォント B	ANK 52 桁 / 漢字 26 桁	ANK 72 桁 / 漢字 36 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 9 x 17 ドット / 漢字 16 x 16 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 15 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		416 x 1662 ドット	576 x 1662 ドット
ページモード最大領域		416 x 1662 ドット	576 x 1662 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded	
2 次元シンボル		PDF417, QR Code, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional, GS1 DataBar Expanded Stacked	
用紙のカット		カット / フィードカット	
ドロアーキック		サポート	
ブザー		サポート	
バッテリー		非サポート	

TM-T88V

		58 mm 仕様	80 mm 仕様
解像度		180 x 180 dpi	
言語		• ANK モデル • 日本語モデル	
印字幅		360 ドット	512 ドット
印字桁数	フォント A	ANK 30 桁 / 漢字 15 桁	ANK 42 桁 / 漢字 21 桁
	フォント B	ANK 40 桁	ANK 56 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 9 x 17 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 16 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		360 x 831 ドット	512 x 831 ドット
ページモード最大領域		360 x 1662 ドット	512 x 1662 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded	
2 次元シンボル		PDF417, QR Code, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omnidirectional, GS1 DataBar Expanded Stacked (Composite Symbology 非サポート)	
用紙のカット		カット / フィードカット	
ドロアーキック		サポート	
ブザー		オプション	
バッテリー		非サポート	

TM-T90II

		58 mm 仕様	80 mm 仕様
解像度		203 x 203 dpi	
言語		日本語モデル	
印字幅		420 ドット	576 ドット
印字桁数	フォント A	ANK 35 桁 / 漢字 17 桁	ANK 48 桁 / 漢字 24 桁
	フォント B	ANK 42 桁 / 漢字 21 桁	ANK 57 桁 / 漢字 28 桁
	フォント C	ANK 52 桁 / 漢字 26 桁	ANK 72 桁 / 漢字 36 桁
文字サイズ	フォント A	ANK 12 x 24 ドット / 漢字 24 x 24 ドット	
	フォント B	ANK 10 x 24 ドット / 漢字 20 x 24 ドット	
	フォント C	ANK 8 x 16 ドット / 漢字 16 x 16 ドット	
文字のベースライン	フォント A	文字の上端から 21 ドット目	
	フォント B	文字の上端から 21 ドット目	
	フォント C	文字の上端から 15 ドット目	
初期改行量		30 ドット	
色指定		第 1 色	
ページモード初期領域		420 x 1662 ドット	576 x 1662 ドット
ページモード最大領域		420 x 1662 ドット	576 x 1662 ドット
バーコード		UPC-A, UPC-E, EAN13, JAN13, EAN8, JAN8, CODE39, ITF, CODABAR, CODE93, CODE128, GS1-128, GS1 DataBar Omnidirectional, GS1 DataBar Truncated, GS1 DataBar Limited, GS1 DataBar Expanded	
2 次元シンボル		PDF417, QRCode, MaxiCode, GS1 DataBar Stacked, GS1 DataBar Stacked Omni-directional, GS1 DataBar Expanded Stacked	
用紙のカット		カット / フィードカット	
ドロアーキック		サポート	
ブザー		サポート	
バッテリー		非サポート	

注意事項

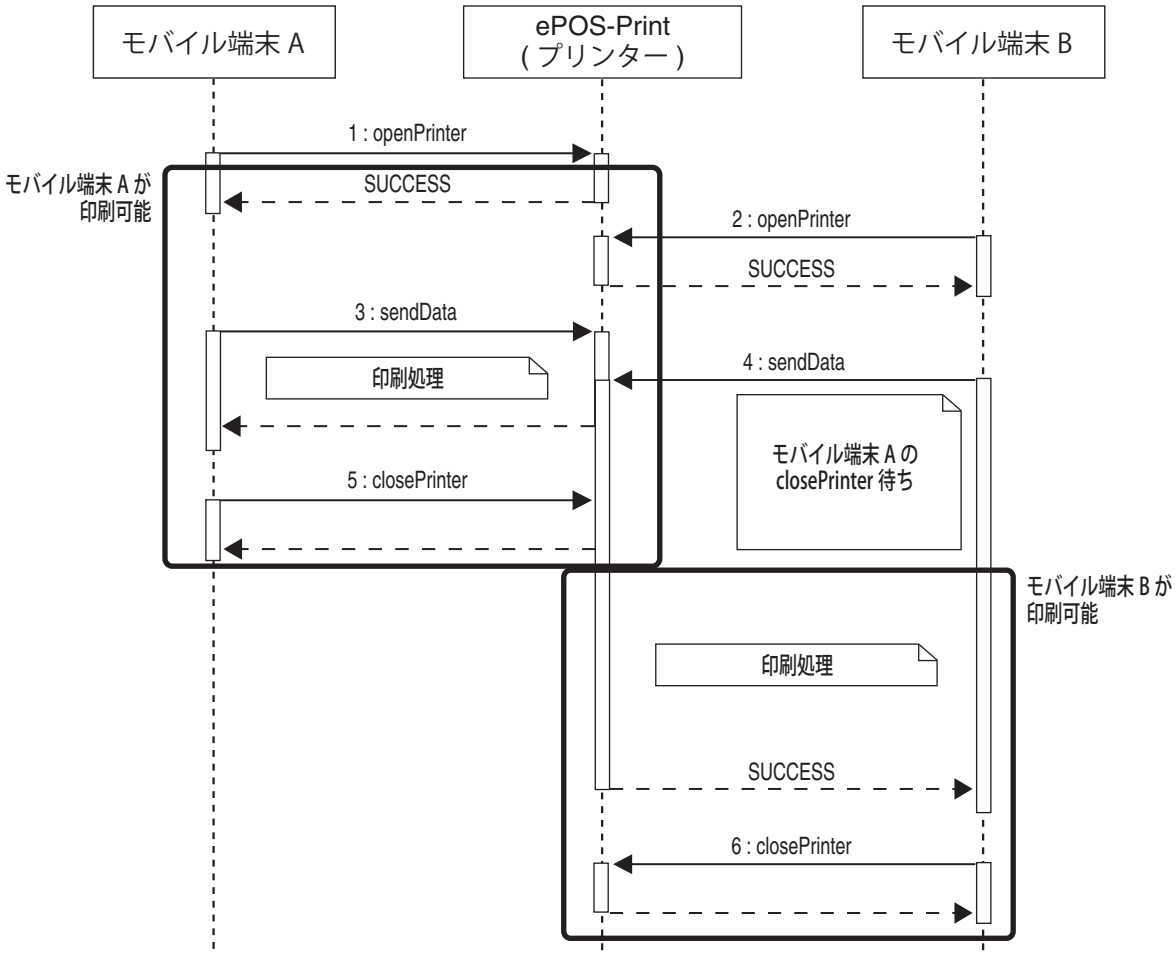
一台のプリンターを複数のモバイル端末から使用する場合

一台のプリンターを複数のモバイル端末から使用する場合、一台の端末から使用している間は他の端末からは印刷ができません。Version 1.6.0 以降では、別の端末によってプリンターが使用されている時に、その処理の終了を openPrinter の処理の中で待つようになります。

以下の図は、モバイル端末 A とモバイル端末 B から 1 台のプリンターを使用する場合の処理の流れを示しています。

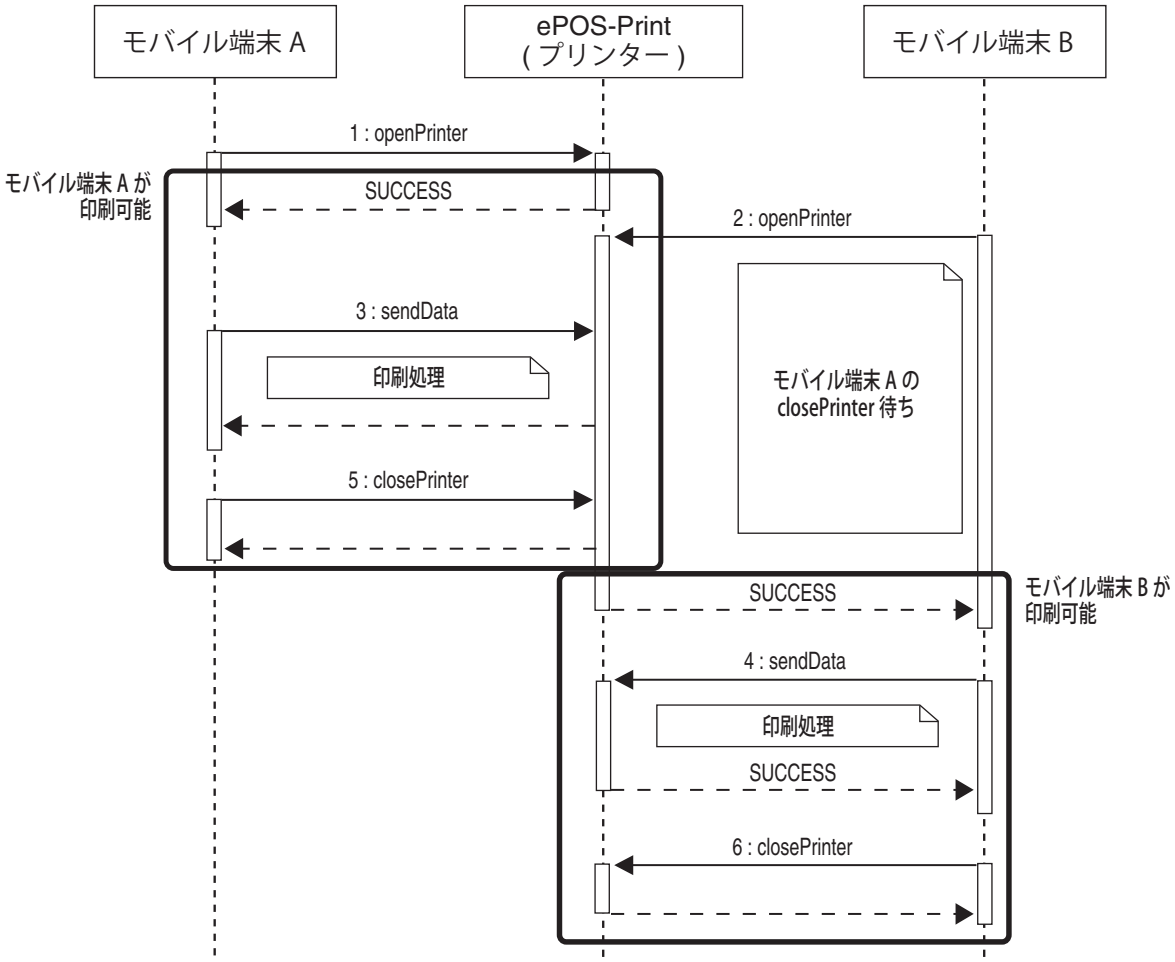
Version 1.5.0 以前

Version 1.5.0 以前では、モバイル端末 B は sendData の処理の中でモバイル端末 A の closePrinter 処理の終了を待ちます。



Version 1.6.0 以降

Version 1.6.0 以降では、モバイル端末 B は openPrinter の処理の中でモバイル端末 A の closePrinter 処理の終了を待ちます。



オープンソースソフト ウェアのライセンス契約

ePOS-Print SDK for Android は、当社が権利を有するソフトウェアのほかにオープンソースソフトウェアを利用しています。ePOS-Print SDK for Android が利用しているオープンソースソフトウェアに関する情報は、以下の URL からご確認ください。

ZXing(https://github.com/zxing/zxing)

ZXing は Apache 2.0 license(http://www.apache.org/licenses/LICENSE-2.0.html) に基づき使用が許諾されます。