EPSON
EXCEED YOUR VISION

ePOS-Print SDK for Windowsストアアプリ 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 Windowsストアアプリで提供している
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 Windowsストアアプリで使用する
プリンター仕様について説明します。

M00074703
Rev.D

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。

Microsoft®、Windows®、Visual Studio®、Visual C#®、Visual Basic®、MSDN® は、米国 Microsoft Corporation の米国およびその他の国における商標または登録商標です。

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。

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弊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

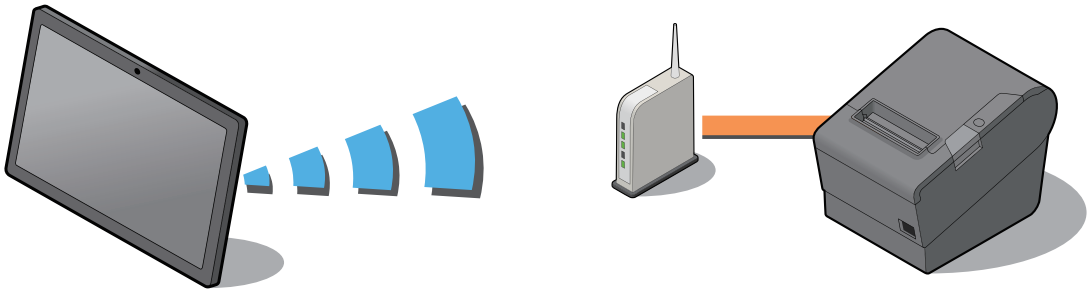
本書は以下のように構成されています。

もくじ

概要

本章では、ePOS-Print SDK for Windows ストアアプリ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 Windows ストアアプリは、エプソン TM プリンターに印刷するための Windows ストアアプリを開発する、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 や、Android アプリケーション向けの ePOS-Print SDK for Android 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 177 ページ「コマンドの送受信」を参照してください。

特徴

- ❏ Windows ストアアプリから、TM プリンターに印刷できます。
- ❏ Windows ストアアプリから、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（位置ぞろえ / 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、193 ページ「プリンターの仕様」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、193 ページ「プリンターの仕様」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

ログ設定 API

- ❏ ログ出力の設定
  （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。）

動作環境

OS

❏ Windows 8.1 (32 bit/ 64 bit)

- 最新のバージョンは、README ファイルを参照してください。
- Windows RT (ARM) は、サポートしていません。

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*®
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 TM 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

Windows ストアアプリを開発するには、以下が必要です。

❏ Visual Studio 2013

開発言語

❏ Visual C#

❏ Visual Basic .NET

以下の開発言語は非サポートです。
- Visual C++ (C++/CX)
- JavaScript

提供物

パッケージ

ファイル名	説明
LibEposPrint.vsix	ePOS-Print ライブラリー （インストーラー形式）
ePOS-Print_Sample_WinStoreApps.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）
EULA.jp.txt	ソフトウェア使用許諾書が記載されています。
EULA.en.txt	ソフトウェア使用許諾書が記載されています。（英語版）
ePOS-Print_SDK_WinStoreApps_ja_revx.pdf	本書です。 ePOS-Print SDK for Windows ストアアプリのプログラミング方法や API の説明をしています。
ePOS-Print_SDK_WinStoreApps_en_revx.pdf	本書です。（英語版） ePOS-Print SDK for Windows ストアアプリのプログラミング方法や API の説明をしています。

マニュアル

ePOS-Print SDK for Windows ストアアプリのマニュアルには以下のものがあります。

- ❏ ePOS-Print SDK for Windows ストアアプリ ユーザーズマニュアル（本書）
- ❏ ePOS-Print SDK for Windows ストアアプリ 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 Windows ストアアプリを使用した、TM プリンター用 Windows ストアアプリのサンプルプログラムがあります。

- ❏ ePOS-Print_Sample_WinStoreApps.zip

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

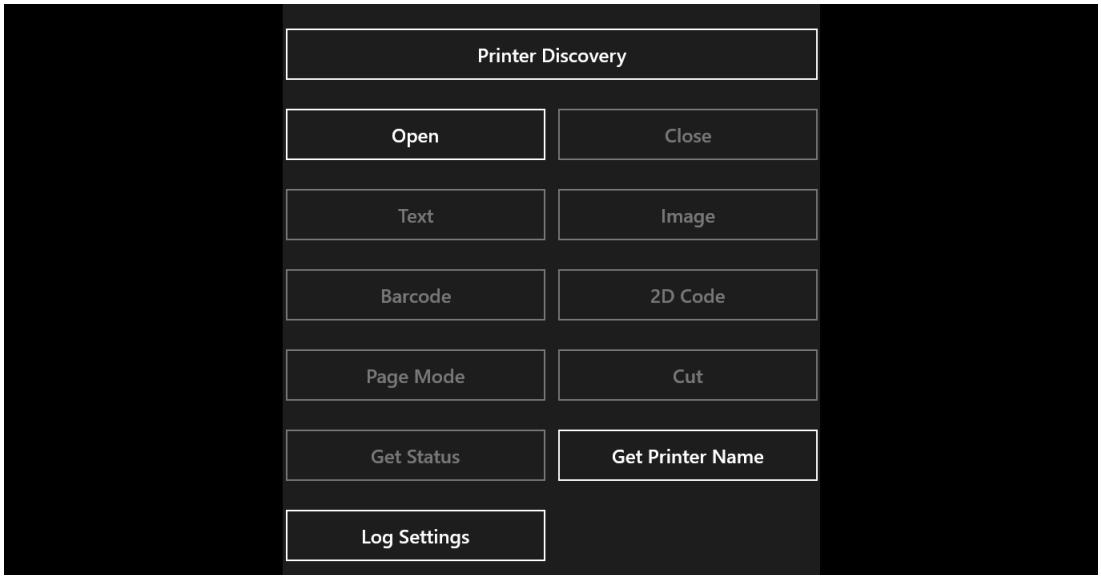
- ❏ ePOS-Print APIの通信API(45ページ)とコマンド送受信API(182ページ)は、同一デバイスに対して同時に使用することはできません。
- ❏ 同じアプリケーション内で同時にオープンできるデバイスポート数は 16 個です。
- ❏ 同じデバイスに対して同時に複数オープンできません。
- ❏ ePOS-Print SDK for Windows ストアアプリは、EPSON Advanced Printer Driver やOPOS ドライバーと共存できません。
- ❏ TCP 接続でプリンターの電源を切った直後に通信を行った場合、エラーとならないことがあります。
- ❏ 複数のネットワークアダプターが有効の環境では使用できません。
- ❏ *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う場合があります。再接続できる場合、自動で復帰します。
- ❏ *Bluetooth* 接続でプリンターとの通信中に、アプリケーションが非アクティブの状態になると、通信が切断されてしまう場合があります。再接続できる場合、自動で復帰します。
- ❏ *Bluetooth* のペアリング時にパスコードの入力を求められる場合があります。この場合、″0000″ または ″4254″ を入力してください。

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 Windows ストアアプリ開発者向けの、ePOS-Print SDK for Windows ストアアプリを使用した Windows ストアアプリ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、Visual C#、Visual Basic のソースファイルを含む Visual Studio 2013 用ストアアプリプロジェクトとして提供しています。

機能



サンプルプログラムは、以下の機能を実装しています。

- ❏ 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

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。

使用環境

開発環境

- Visual Studio 2013

開発環境構築の詳細は、「ePOS-Print SDK for Windows ストアアプリ 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

ターゲット端末

- 開発者用ライセンスを取得済みのデバイス

開発者用ライセンスの取得には、Microsoft アカウントが必要です。

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。

ここでは、ターゲット端末にサイドローディングして起動する方法を説明しています。

1 ePOS-Print SDK for Windowsストアアプリ (LibEposPrint.vsix) をインストールします。

2 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。

3 サンプルプログラムの開発言語を選択し、sln ファイルを起動します。

4 [プロジェクト] メニューから [参照の追加] を選択します。

5 ″参照マネージャー″画面が表示されます。[Windows]-[拡張] を選択します。

6 以下の SDK にチェックし、[OK] をクリックします。

- Epson ePOS-Print SDK
- Microsoft Visual C++ 2013 Runtime Package for Windows

7 [プロジェクト] メニューから [ストア]-[アプリ パッケージの作成] を選択します。

開発環境でサンプルプログラム実行させる場合、(ビルド) メニューから (ソリューションのリビルド) と (ソリューションの配置) を実行し、Windows のスタートメニューからサンプルプログラムを起動します。

8 ″パッケージの作成″画面が表示されます。[いいえ] を選択し、[次へ] をクリックします。

9 ″パッケージの選択と構成″画面が表示されます。
作成するパッケージの設定をし、[作成] をクリックします。

ePOS-Print SDK for Windows ストアアプリは、C++/CX (アンマネージメントコード) で開発されています。そのため、(作成するパッケージとソリューション構成マッピングを選択する) の設定は、″x86″ か ″x64″ 以外は指定しないでください。

10 ″パッケージの作成が完了しました″画面が表示されます。
出力場所を確認し、[OK] をクリックします。

11 出力場所を開き、フォルダーをターゲット端末の任意の場所にコピーします。

12 コピーしたフォルダー内にある〝Add-AppDevPackakge.ps1〟を選択し、コンテキストメニューから、[PowerShell で実行] を選択します。

Windows Power Shell の実行時に、開発者用ライセンスの取得を促すメッセージが表示されます。Microsoft アカウントを使用して、開発者用ライセンスを取得してください。

13 スクリプトの指示に従って、サンプルプログラムをターゲット端末にサイドローディングします。

14 ターゲット端末の Windows のスタートメニューから、プロジェクトの名前を選択します。

サンプルプログラムが起動されます。

使用方法

ここでは、以下の使い方を説明します。

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] を押します。
[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の IP アドレス /Mac アドレスを押します。

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] を押します。
手順 3 で選択したプリンターの ″Device Type″ と ″IP Address/Mac Address″ 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] を押します。

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) を押してください。 詳細はテキスト印刷 (18 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) を押してください。 詳細はグラフィック印刷 (19 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) を押してください。 詳細はバーコード印刷 (19 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) を押してください。 詳細は 2 次元シンボル印刷 (20 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) を押してください。 詳細はページモード印刷 (20 ページ) を参照してください。
用紙カット	メイン画面の (Cut) を押してください。 詳細は用紙カット (20 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) を押してください。
ログ出力設定	メイン画面の (Log Settings) を押してください。 詳細はログ出力設定 (21 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。実行結果を確認し、[OK] をクリックします。

- 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 (22 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 (23 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] を押し、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	位置ぞろえを設定します。
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] を押し、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] を押し、印刷する画像ファイルを選択します。

2 [Color Mode] を押し、階調を選択します。

(Gray 16) (多階調印刷) をサポートしている機種は、TM-T88V/ TM-T70II/ TM-T90II のみです。

3 [Halftone Method] を押し、ハーフトーンの処理方法を選択します。

4 [Brightness] を押し、数値を入力して明るさを指定します。

5 [Print] を押し、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] を押し、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] を押し、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] を押し、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] を押し、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] を押し、ログ出力の設定を有効にします。

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
HR_S_OK	成功
HR_E_INVALIDARG	• 不正なパラメーターが渡された • 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
HR_E_ACCESSDENIED	• オープン処理に失敗した • デバイスとの通信に失敗した • オフライン状態
HR_E_ABORT	処理がタイムアウトされた
HR_E_PENDING	処理を実行できなかった
HR_E_OUTOFMEMORY	処理に必要なメモリーが確保できなかった
HR_E_FAIL	• 不適切な方法で使用された • 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 (41 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
HR_E_INVALIDARG	• 不正なパラメーターが渡された • 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
HR_E_ACCESSDENIED	• オープン処理に失敗した • デバイスとの通信に失敗した • プリンターがオフライン状態だった
HR_E_PENDING	処理を実行できなかった
HR_E_ABORT	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
HR_E_OUTOFMEMORY	処理に必要なメモリーが確保できなかった
HR_E_FAIL	• 不適切な方法で使用された • 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 (177 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] を押します。
[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 アドレスが表示されます。

3 表示された [Printer List] の中から、使用するプリンターの IP アドレス /Mac アドレスを押します。

4 メイン画面の [Get Printer Name] を押します。

5 [Get Printer Name] を押します。

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。

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。

- 本章のソースコードを使った説明は、Visual C# で説明しています。他の言語は、適宜読み替えてください。
- ePOS-Print SDK for Windows ストアアプリを使用した、Windows ストアアプリ開発環境の構築方法については、「ePOS-Print SDK for Windows ストアアプリ 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 Windows ストアアプリの組み込み方法

ePOS-Print SDK for Windows ストアアプリの組み込み方法について説明します。

Visual Studio 2013 で説明しています。

以下の手順で組み込んでください。

1 Visual Studio 2013 で新しいプロジェクトを作成します。

2 プロジェクトの〝Package.appxmanifest〟ファイルに、通信機能の宣言をします。

- TCP の宣言
  1. 〝Package.appxmanifest〟のコンテキストメニューから [デザイナーの表示] を選択し、Package.appxmanifest を表示させます。
  2. [機能] を選択し、〝機能〟画面を表示させます。
  3. 〝インターネット (クライアント)〟と、〝プライベートネットワーク (クライアントとサーバー)〟にチェックし、保存します。
- *Bluetooth* の宣言
  1. 〝Package.appxmanifest〟のコンテキストメニューから [コードの表示] を選択し、Package.appxmanifest を表示させます。
  2. <Capabilities> タグ内に、以下の太字箇所を追記し、保存します。

```
<Capabilities>
  <Capability Name="internetClient" />
  <DeviceCapability Name="proximity" />
  <m2:DeviceCapability Name="bluetooth.rfcomm">
    <m2:Device Id="any">
      <m2:Function Type="name:serialPort" />
    </m2:Device>
  </m2:DeviceCapability>
</Capabilities>
```

3 [プロジェクト] メニューから [参照の追加] を選択します。

4 ″参照マネージャー″画面が表示されます。[Windows]-[拡張] を選択します。

5 以下の SDK にチェックし、[OK] をクリックします。

- Epson ePOS-Print SDK
- Microsoft Visual C++ 2013 Runtime Package for Windows

Epson ePOS-Print SDK は、LibEposPrint.vsix を実行してインストールします。
詳細は、「ePOS-Print SDK for Windows ストアアプリ 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。

6 [ビルド] メニューから [構成マネージャー] を選択します。

7 ″構成マネージャー″画面が表示されます。[アクティブソリューションプラットフォーム] の設定を、″x86″または″x64″に指定し、[閉じる] をクリックします。

ePOS-Print SDK for Windows ストアアプリは、C++/CX (アンマネージドコード) で開発しています。そのため、(アプリケーションプラットフォーム) の設定は、"x86"、または "x64" 以外は指定しないでください。

8 使用したいアプリケーションのソースファイルに、インポート定義を記載します。
下記を参照してください。

- Visual C#: **`using LibEposPrint;`**
- Visual Basic: **`Imports LibEposPrint`**

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. プリンターの検索 (28 ページ) *
   - ❏ プリンターの検索を開始 (28 ページ)
   - ❏ プリンターの検索結果を取得 (28 ページ)
   - ❏ プリンターの検索を終了 (29 ページ)

↓

2. 印刷ドキュメントの作成 (30 ページ)
   - ❏ テキストの印刷ドキュメントを作成する場合 (30 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 (31 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 (32 ページ)

↓

3. 印刷ドキュメントの送信 (33 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 (34 ページ) を参照してください。

プリンターの検索

プリンターの検索を開始

Finder クラスの StartAsync（167 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。

```
// 検索開始
try
{
    //Wi-Fi/Ethernet デバイスの場合
    await Finder.StartAsync(IoDevType.TCP, "255.255.255.255");
    //Bluetooth デバイスの場合
    await Finder.StartAsync(IoDevType.BLUETOOTH, "");
}
// 例外処理
catch (Exception ex)
{
    if (ex.HResult == IoHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

プリンターの検索結果を取得

Finder クラスの GetDeviceInfoList（170 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。

```
DeviceInfo[] list = null;

// デバイス一覧の取得
try
{
    list = Finder.GetDeviceInfoList(IoFilterOption.DEFAULT);
}
// 例外処理
catch (Exception ex)
{
    if (ex.HResult == IoHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

プリンターの検索に時間がかかるため、Finder クラスの StartAsync 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 StopAsync（169 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。

```
// 検索終了
try
{
    await Finder.StopAsync();
}
// 例外処理
catch (Exception ex)
{
    if (ex.HResult == IoHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

印刷ドキュメントの作成

印刷ドキュメントは、Builder クラス（43 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。

```
Builder builder = null;

try
{
    //Builder クラスのインスタンスを初期化
    builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    // 印刷ドキュメントの作成
    builder.AddTextLang(Lang.LANG_JA);
    builder.AddTextSmooth(Smooth.SMOOTH_TRUE);
    builder.AddTextFont(Font.FONT_A);
    builder.AddTextSize(3, 3);
    builder.AddText("Hello World!\n");
    builder.AddCut(Cut.CUT_FEED);
}
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

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

```
Builder builder = null;

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

    // 印刷ドキュメントの作成
    //＜印字文字の設定＞
    builder.AddTextLang(Lang.LANG_JA);
    builder.AddTextSmooth(Smooth.SMOOTH_TRUE);
    builder.AddTextFont(Font.FONT_A);
    builder.AddTextSize(4, 4);
    builder.AddTextStyle(Reverse.REVERSE_FALSE, Underline.UNDERLINE_FALSE,
                         Emphasis.EMPHASIS_TRUE, Color.UNSPECIFIED);

    //＜印刷データを指定＞
    builder.AddText("Hello World!\n");
    builder.AddCut(Cut.CUT_FEED);
}
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

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、Windows.Grapics.Imaging.BitmapDecoder クラスを Builder クラスの AddImageAsync（79 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。

```
using Windows.Storage;
using Windows.Storage.Streams;
using Windows.Graphics.Imaging;

Builder builder = null;

private IRandomAccessStreamWithContentType m_selectStream = null;
Builder builder = null;

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

    // 印刷ドキュメントの作成
    StorageFile imageFile = await Windows.ApplicationModel.Package.Current.
                            InstalledLocation.GetFileAsync(" グラフィックのファイル名 ");
    IRandomAccessStreamWithContentType stream = await imageFile.OpenReadAsync();
    BitmapDecoder imageData = await BitmapDecoder.CreateAsync(stream);

    await builder.AddImageAsync(imageData, 0, 0, (int)imageData.PixelWidth,
                                (int)imageData.PixelHeight, Color.DEFAULT, Mode.DEFAULT,
                                Halftone.DEFAULT, Builder.PARAM_DEFAULT);
    builder.AddCut(Cut.CUT_FEED);
}
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

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 (83 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 (98 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 (102 ページ)) と印字開始位置 (AddPagePosition (106 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 (100 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

```
Builder builder = null;

try
{
    //Builder クラスのインスタンスを初期化
    builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_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(Lang.LANG_JA);
    builder.AddTextSmooth(Smooth.SMOOTH_TRUE);
    builder.AddTextFont(Font.FONT_A);
    builder.AddTextSize(2, 2);
    builder.AddTextStyle(Reverse.REVERSE_FALSE, Underline.UNDERLINE_FALSE,
                         Emphasis.EMPHASIS_TRUE, Color.UNSPECIFIED);
    //< 印刷データを指定 >
    builder.AddText("Hello World!\n");
    //< ページモード終了 >
    builder.AddPageEnd();
    builder.AddCut(Cut.CUT_FEED);
}
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

印刷ドキュメントの送信

印刷ドキュメントは、Print クラス（45 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、SendDataAsync 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。以下のプログラミングを参考にしてください。

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

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

    // 印刷ドキュメントの作成
    //< 印字文字の設定 >
    builder.AddTextLang(Lang.LANG_JA);
    builder.AddTextFont(Font.FONT_A);
    builder.AddTextSize(4, 4);
    builder.AddTextStyle(Reverse.REVERSE_FALSE, Underline.UNDERLINE_FALSE,
                         Emphasis.EMPHASIS_TRUE, Color.UNSPECIFIED);

    //< 印刷データを指定 >
    builder.AddText("Hello World!\n");
    builder.AddCut(Cut.CUT_FEED);

    // 印刷ドキュメントを送信
    //<Wi-Fi®/Ethernet デバイスの場合 >
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, , "192.168.192.168",
                                   Monitoring.MONITORING_FALSE, Print.PARAM_DEFAULT);
    //<Bluetooth デバイスの場合 >
    await printer.OpenPrinterAsync(DevType.DEVTYPE_BLUETOOTH, , "00:00:12:34:56:78",
                                   Monitoring.MONITORING_FALSE, Print.PARAM_DEFAULT);

    //< データを送信 >
    PrinterStatus sendStatus = await printer.SendDataAsync(builder, 100000);

    //< プリンターとの通信を終了 >
    await printer.ClosePrinterAsync();
}
catch(Exception ex){

    // エラーコード取得
    int errCode = ex.HResult;
    // エラー発生時のプリンターステータス取得
    PrinterStatus errStatus = printer.GetPrinterStatus(ex.HResult);

    //・・・例外処理・・・
}
```

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 printer = null;
Builder builder = null;
PrinterStatus sendStatus;

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

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

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

    // 印刷ドキュメントを送信
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, , "192.168.192.168",
                                   Monitoring.MONITORING_FALSE, Print.PARAM_DEFAULT);

    //＜確認用データを送信＞
    PrinterStatus conStatus = await printer.SendDataAsync(conBuilder, 10000);  ③

    if ((conStatus.printerStatus & Print.ST_OFF_LINE) != Print.ST_OFF_LINE)
    {
        //＜印刷データを送信＞
        sendStatus = await printer.SendDataAsync(builder, 10000);          ④
    }

    //＜プリンターとの通信を終了＞
    await printer.ClosePrinterAsync();
}
catch(Exception ex){

    // エラーコード取得
    int errCode = ex.HResult;
    // エラー発生時のプリンターステータス取得
    PrinterStatus errStatus = printer.GetPrinterStatus(ex.HResult);

    //・・・例外処理・・・
}
```

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。

プリンターステータスを自動で取得

ePOS-Print SDK では、プリンターの状態をコールバックによって、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コールバックメソッドを実装
private void Status_Change_Event(string deviceName, int status)
{
   //・・・処理・・・
}

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

     try {
        // プリンターの状態変化の通知先を登録
        StatusChangeEvent statusChangeEvent = new StatusChangeEvent(Status_Change_Event);
         printer.SetStatusChangeEventCallback(statusChangeEvent);

        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, , "192.168.192.168",
                                       Monitoring.MONITORING_TURE, Print.PARAM_DEFAULT);

        //・・・処理・・・

     }
     catch(Exception ex){
        // エラーコード取得
        int errCode = ex.HResult;

        //・・・例外処理・・・
     }

}
```

①／④

②

③

1 イベント発生時の通知先メソッドを実装します。

> 上記では、プリンターステータスを OpenPrinterAsync (126 ページ) で指定した間隔で通知するコールバックメソッドを定義しています。
> ePOS-Print 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コールバックメソッド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 で使用できるコールバックメソッドは、イベント一覧 (36 ページ) を参照してください。

2 プリンターステータスの通知先を登録します。

3 OpenPrinterAsync (126 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。

4 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い場合、Print クラスの ClosePrinterAsync (132 ページ) で終了します。

イベント一覧

コールバックメソッドの詳細は、下記、コールバックメソッド登録 API を説明している、API リファレンス (43 ページ) を参照してください。

機能	コールバックメソッド通知先登録 API
プリンターステータスの通知	SetStatusChangeEventCallback (137 ページ)
オンラインの通知	SetOnlineEventCallback (139 ページ)
オフラインの通知	SetOfflineEventCallback (141 ページ)
無応答の通知	SetPowerOffEventCallback (143 ページ)
カバークローズの通知	SetCoverOkEventCallback (145 ページ)
カバーオープンの通知	SetCoverOpenEventCallback (147 ページ)
用紙ありの通知	SetPaperOkEventCallback (149 ページ)
用紙残量少の通知	SetPaperNearEndEventCallback (151 ページ)
用紙なしの通知	SetPaperEndEventCallback (153 ページ)
ドロアークローズの通知	SetDrawerClosedEventCallback (155 ページ)
ドロアーオープンの通知	SetDrawerOpenEventCallback (157 ページ)
バッテリー残量なしの通知	SetBatteryLowEventCallback (159 ページ)
バッテリー残量ありの通知	SetBatteryOkEventCallback (161 ページ)
バッテリーステータスの通知	SetBatteryStatusChangeEventCallback (163 ページ)

例外処理

ePOS-Print SDK for Windows ストアアプリには、エラー発生時に Windows のシステムの例外を発生させ、呼び元にエラーを通知します。また、印刷データ送信時に例外が発生した場合、プリンターの状態も取得します。

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、エラーステータス一覧 (39 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。 詳細は、プリンターステータス一覧 (40 ページ) を参照してください。
バッテリーステータス	印刷データ送信時の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 (41 ページ) を参照してください。

プリンターステータスとバッテリーステータスは、プリンターの状態が変化したときにも、コールバックでアプリケーションに自動で通知できます。詳細は、プリンターステータスを自動で取得 (35 ページ) を参照してください。

処理方法

ePOS-Print API

以下のプログラミングを参考にしてください。

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

    //Builder クラスのインスタンスを初期化
    builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    // 印刷ドキュメントの作成
    builder.AddText("Hello World!\n");
    builder.AddCut(Cut.CUT_FEED);

    // 印刷ドキュメントを送信
    //<Wi-Fi/Ethernet デバイスの場合 >
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, , "192.168.192.168",
                                   Monitoring.MONITORING_FALSE, Print.PARAM_DEFAULT);
    //<Bluetooth デバイスの場合 >
    await printer.OpenPrinterAsync(DevType.DEVTYPE_BLUETOOTH, , "00:00:12:34:56:78",
                                   Monitoring.MONITORING_FALSE, Print.PARAM_DEFAULT);

    //< データを送信 >
    PrinterStatus sendStatus = await printer.SendDataAsync(builder, 100000);

    //< プリンターとの通信を終了 >
    await printer.ClosePrinterAsync();
}
catch(Exception ex){

    // エラーコード取得
    int errCode = ex.HResult;
    // エラー発生時のプリンターステータス取得
    PrinterStatus errStatus = printer.GetPrinterStatus(ex.HResult);

    //・・・例外処理・・・
}
```

プリンター検索 API

以下のプログラミングを参考にしてください。

```
DeviceInfo[] list = null;

// 検索開始
try
{
    //Wi-Fi/Ethernet デバイスの場合
    await Finder.StartAsync(IoDevType.TCP, "255.255.255.255");
    //Bluetooth デバイスの場合
    await Finder.StartAsync(IoDevType.BLUETOOTH, "");

    // デバイス一覧の取得
    list = Finder.GetDeviceInfoList(IoFilterOption.DEFAULT);

}
catch (Exception ex)
{
    //・・・例外処理・・・
    if (ex.HResult == IoHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
}
```

エラーステータス一覧

ePOS-Print SDK for Windows ストアアプリでは、以下のエラーステータスを持つ Windows のシステムの例外を返します。

エラーステータス	要因
HR_E_INVALIDARG (0x80070057)	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。 • 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。
HR_E_ACCESSDENIED (0x80070005)	• オープン処理に失敗した。 • デバイスとの通信に失敗した。 • プリンターがオフライン状態だった。 < 例 > • 指定したプリンターに接続できなかった。 • プリンターへのデータ送信に失敗した。
HR_E_ABORT (0x80004004)	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。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。 < 例 > 同様の処理が、他のスレッドで実行中のため、処理が実行できなかった。
HR_E_FAIL (0x80004005)	• その他のエラーが発生した。 • 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。

プリンターステータス一覧

プリンターステータス	要因
Print.ST_NO_RESPONSE (0x00000001)	プリンター無応答
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/TM-P20> Print.ST_BATTERY_OFFLINE (0x00000004)	バッテリー残量によるオフライン状態
Print.ST_OFF_LINE (0x00000008)	オフライン状態
Print.ST_COVER_OPEN (0x00000020)	カバーが開いている
Print.ST_PAPER_FEED (0x00000040)	紙送りスイッチによる紙送り中
Print.ST_WAIT_ON_LINE (0x00000100)	オンライン復帰待ち中
Print.ST_PANEL_SWITCH (0x00000200)	紙送りスイッチが押されている
Print.ST_MECHANICAL_ERR (0x00000400)	メカニカルエラー発生
Print.ST_AUTOCUTTER_ERR (0x00000800)	オートカッターエラー発生
Print.ST_UNRECOVER_ERR (0x00002000)	復帰不可能エラー発生
Print.ST_AUTORECOVER_ERR (0x00004000)	自動復帰エラー発生
Print.ST_RECEIPT_NEAR_END (0x00020000)	ロール紙ニアエンド検出器に用紙なし
Print.ST_RECEIPT_END (0x00080000)	ロール紙エンド検出器に用紙なし
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)

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) (41 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、プリンターの仕様 (193 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 Windows ストアアプリで用意されている 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（43 ページ）

❏ Print クラス（45 ページ）

プリンターの仕様によって、使用できる API や指定できる設定値は異なります。
詳細は、プリンターの仕様 (193 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	46
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	48
テキスト	AddTextAlign	位置ぞろえ設定を命令バッファーに追加	49
	AddTextLineSpace	改行量設定を命令バッファーに追加	51
	AddTextRotate	倒立印字設定を命令バッファーに追加	53
	AddText	文字印字を命令バッファーに追加	55
	AddTextLang	言語設定を命令バッファーに追加	57
	AddTextFont	文字フォント設定を命令バッファーに追加	59
	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	63
	AddTextSize	文字倍率設定を命令バッファーに追加	65
	AddTextStyle	文字装飾設定を命令バッファーに追加	67
	AddTextPosition	文字印字位置設定を命令バッファーに追加	69
紙送り	AddFeedUnit	ドット単位の紙送りを命令バッファーに追加	71
	AddFeedLine	行単位の紙送りを命令バッファーに追加	73
	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	118
グラフィック	AddImageAsync (画像圧縮)	イメージデータを圧縮して命令バッファーに追加	75
	AddImageAsync	ラスターイメージ印字を命令バッファーに追加	79
	AddLogo	NV ロゴ印字を命令バッファーに追加	83
バーコード	AddBarcode	バーコード印字を命令バッファーに追加	85
	AddSymbol	2 次元シンボル印字を命令バッファーに追加	91

API		説明	ページ
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	98
	AddPageEnd	ページモード終了を命令バッファーに追加	100
	AddPageArea	ページモード印字領域設定を命令バッファーに追加	102
	AddPageDirection	ページモード印字方向設定を命令バッファーに追加	104
	AddPagePosition	ページモード印字位置設定を命令バッファーに追加	106
	AddPageLine	ページモード直線描画を命令バッファーに追加	108
	AddPageRectangle	ページモード四角形描画を命令バッファーに追加	110
カット	AddCut	用紙カットを命令バッファーに追加	112
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	114
ブザー	AddSound	ブザー鳴動の鳴動周期を設定し、命令バッファーに追加	116
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	120
コマンド送信	AddCommand	コマンドを命令バッファーに追加	123

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	125
OpenPrinterAsync	プリンターとの通信とプリンターステータスのモニタリングを開始	126
OpenPrinterAsync(旧フォーマット)	プリンターとの通信とプリンターステータスのモニタリングを開始 (タイムアウトが設定できません)	129
ClosePrinterAsync	プリンターとの通信を終了	132
SendDataAsync	プリンターにコマンドを送信	134
SetStatusChangeEventCallback	プリンターステータスの通知先を登録	137
SetOnlineEventCallback	オンラインイベントの通知先を登録	139
SetOfflineEventCallback	オフラインイベントの通知先を登録	141
SetPowerOffEventCallback	無応答イベントの通知先を登録	143
SetCoverOkEventCallback	カバークローズイベントの通知先を登録	145
SetCoverOpenEventCallback	カバーオープンイベントの通知先を登録	147
SetPaperOkEventCallback	用紙ありイベントの通知先を登録	149
SetPaperNearEndEventCallback	用紙残量少イベントの通知先を登録	151
SetPaperEndEventCallback	用紙なしイベントの通知先を登録	153
SetDrawerClosedEventCallback	ドロアークローズイベントの通知先を登録	155
SetDrawerOpenEventCallback	ドロアーオープンイベントの通知先を登録	157
SetBatteryLowEventCallback	バッテリー残量なしイベントの通知先を登録	159
SetBatteryOkEventCallback	バッテリー残量ありイベントの通知先を登録	161
SetBatteryStatusChangeEventCallback	バッテリーステータスの通知先を登録	163
GetPrinterStatus	例外発生のスロー時に、プリンターステータスおよびバッテリーステータスを取得します。	165

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。

Builder クラスのインスタンスを複数生成すると、メモリーが大量に消費されます。最小限のインスタンスを生成し、ClearCommandBuffer (48 ページ) で命令バッファーをクリアしながら使用してください。

構文

Visual C#

```
public Builder(System.String printerModel,
               LibEposPrint.ModelLang lang);
```

Visual Basic .NET

```
Public Sub Builder(printerModel As String,
                   lang As LibEposPrint.ModelLang)
```

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
ModelLang.MODEL_LANG_ANK	ANK モデル
ModelLang. MODEL_LANG_JAPANESE	日本語モデル

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	• 不正なパラメーターが渡された。 • 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。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V",ModelLang.MODEL_LANG_JAPANESE)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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。

構文

Visual C#

```
public void ClearCommandBuffer();
```

Visual Basic .NET

```
Public Sub ClearCommandBuffer()
```

例

命令バッファーをクリアする場合

- Visual C#

```
Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
//・・・処理・・・

try
{
    builder.ClearCommandBuffer();
    builder = null;
}
catch (Exception ex)
{
    builder = null;
}
```

- Visual Basic .NET

```
Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
//・・・処理・・・

Try
    builder.ClearCommandBuffer()
    builder = Nothing
Catch ex As Exception
    builder = Nothing
End Try
```

AddTextAlign

位置ぞろ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ぞろえを設定する場合、本 API ではなく、AddPagePosition (106 ページ) で設定してください。

構文

Visual C#

```
public void AddTextAlign(LibEposPrint.Align align);
```

Visual Basic .NET

```
Public Sub AddTextAlign(align As LibEposPrint.Align)
```

パラメーター

- align：　位置ぞろえを指定します。

設定値	説明
Align.ALIGN_LEFT (デフォルト)	左ぞろえ
Align.ALIGN_CENTER	中央ぞろえ
Align.ALIGN_RIGHT	右ぞろえ

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

中央ぞろえに設定する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextAlign(Align.ALIGN_CENTER);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextAlign(Align.ALIGN_CENTER)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddTextLineSpace

改行量設定を命令バッファーに追加します。

構文

Visual C#

```
public void AddTextLineSpace(int linespc);
```

Visual Basic .NET

```
Public Sub AddTextLineSpace(linespc As Integer)
```

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。

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

改行量を 30 ドットに設定する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextLineSpace(30);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextLineSpace(30)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (104 ページ) で設定してください。

構文

Visual C#

```
public void AddTextRotate(LibEposPrint.Rotate rotate);
```

Visual Basic .NET

```
Public Sub AddTextRotate(rotate As LibEposPrint.Rotate)
```

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
Rotate.ROTATE_TRUE	倒立印字を指定
Rotate.ROTATE_FALSE (デフォルト)	倒立印字を解除

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

倒立印字を設定する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextRotate(Rotate.ROTATE_TRUE);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextRotate(Rotate.ROTATE_TRUE)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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。

構文

Visual C#

```
public void AddText(System.String data);
```

Visual Basic .NET

```
Public Sub AddText(data As String)
```

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、以下を指定します。

文字列	説明
• Visual C# (エスケープシーケンス) \t • Visual Basic .NET vbTab	水平タブ (HT)
• Visual C# (エスケープシーケンス) \n • Visual Basic .NET vbCrLf	改行 (CR+LF)
• Visual C# (エスケープシーケンス) \\ • Visual Basic .NET \	バックスラッシュ

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

文字列を追加する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddText("Hello,\t");
    builder.AddText("World\n");
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddText("Hello, " + vbTab)
    builder.AddText("World" + vbCrLF)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (55 ページ) で指定された文字列をエンコードします。

本 API は、AddText (55 ページ) を呼び出す前に呼び出す API です。

構文

Visual C#

```
public void AddTextLang(LibEposPrint.Lang lang);
```

Visual Basic .NET

```
Public Sub AddTextLang(lang As LibEposPrint.Lang)
```

パラメーター

- lang :　対象言語を指定します。

設定値	説明
Lang.LANG_EN(デフォルト)	英語 (ANK 仕様)
Lang.LANG_JA	日本語

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

日本語に設定する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextLang(Lang.LANG_JA);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextLang(Lang.LANG_JA)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddTextFont(LibEposPrint.Font font);
```

Visual Basic .NET

```
Public Sub AddTextFont(font As LibEposPrint.Font)
```

パラメーター

- font：　フォントを指定します。

設定値	説明
Font.FONT_A (デフォルト)	フォント A
Font.FONT_B	フォント B
Font.FONT_C	フォント C
Font.FONT_D	フォント D
Font.FONT_E	フォント E

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextFont(Font.FONT_B);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextFont(Font.FONT_B)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddTextSmooth(LibEposPrint.Smooth smooth);
```

Visual Basic .NET

```
Public Sub AddTextSmooth(smooth As LibEposPrint.Smooth)
```

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
Smooth.SMOOTH_TRUE	スムージングを指定
Smooth.SMOOTH_FALSE （デフォルト）	スムージングを解除

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextSmooth(Smooth.SMOOTH_TRUE);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextSmooth(Smooth.SMOOTH_TRUE)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddTextDouble

文字の倍角設定を命令バッファーに追加します。

構文

Visual C#

```
public void AddTextDouble(LibEposPrint.DoubleSize dw,
                          LibEposPrint.DoubleSize dh);
```

Visual Basic .NET

```
Public Sub AddTextDouble(dw As LibEposPrint.DoubleSize,
                         dh As LibEposPrint.DoubleSize)
```

パラメーター

- dw：　文字の横倍角を指定します。

設定値	説明
DoubleSize.DOUBLE_TRUE	横倍角を指定
DoubleSize.DOUBLE_FALSE (デフォルト)	横倍角を解除
DoubleSize.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
DoubleSize.DOUBLE_TRUE	縦倍角を指定
DoubleSize.DOUBLE_FALSE (デフォルト)	縦倍角を解除
DoubleSize.UNSPECIFIED	設定を変更しない

dw と dh のパラメーターの両方を DoubleSize.DOUBLE_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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

4 倍角を設定する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextDouble(DoubleSize.DOUBLE_TRUE, DoubleSize.DOUBLE_TRUE);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextDouble([DoubleSize].DOUBLE_TRUE,[DoubleSize].DOUBLE_TRUE)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddTextSize

文字の倍率設定を命令バッファーに追加します。

構文

Visual C#

```
public void AddTextSize(int width, int height);
```

Visual Basic .NET

```
Public Sub AddTextSize(width As Integer,
                       height As Integer)
```

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 (デフォルト : 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 (デフォルト : 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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextSize(4, 4);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextSize(4,4)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddTextStyle

文字の装飾設定を命令バッファーに追加します。

構文

Visual C#

```
public void AddTextStyle(LibEposPrint.Reverse reverse,
                         LibEposPrint.Underline ul,
                         LibEposPrint.Emphasis em,
                         LibEposPrint.Color color);
```

Visual Basic .NET

```
Public Sub AddTextStyle(reverse As LibEposPrint.Reverse,
                        ul As LibEposPrint.Underline,
                        em As LibEposPrint.Emphasis,
                        color As LibEposPrint.Color)
```

パラメーター

- reverse：　白黒反転文字を指定します。

設定値	説明
Reverse.REVERSE_TRUE	白黒反転文字を指定
Reverse.REVERSE_FALSE（デフォルト）	白黒反転文字を解除
Reverse.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
Underline.UNDERLINE_TRUE	アンダーラインを指定
Underline.UNDERLINE_FALSE（デフォルト）	アンダーラインを解除
Underline.UNSPECIFIED	設定を変更しない

- em：　太字を指定します。

設定値	説明
Emphasis.EMPHASIS_TRUE	太字を指定
Emphasis.EMPHASIS_FALSE（デフォルト）	太字を解除
Emphasis.UNSPECIFIED	設定を変更しない

- color： 色を指定します。

設定値	説明
Color.COLOR_NONE	非印字
Color.COLOR_1（デフォルト）	第 1 色
Color.COLOR_2	第 2 色
Color.COLOR_3	第 3 色
Color.COLOR_4	第 4 色
Color.UNSPECIFIED	設定を変更しない

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextStyle(Reverse.UNSPECIFIED, Underline.UNDERLINE_TRUE,
                         Emphasis.UNSPECIFIED, Color.UNSPECIFIED);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextStyle(Reverse.UNSPECIFIED, Underline.UNDERLINE_TRUE,
                         Emphasis.UNSPECIFIED, Color.UNSPECIFIED)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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。

構文

Visual C#

```
public void AddTextPosition(int x);
```

Visual Basic .NET

```
Public Sub AddTextPosition(x As Integer)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddTextPosition(120);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddTextPosition(120)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddFeedUnit(int unit);
```

Visual Basic .NET

```
Public Sub AddFeedUnit(unit As Integer)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddFeedUnit(30);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddFeedUnit(30)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddFeedLine

行単位の紙送りを命令バッファーに追加します。

構文

Visual C#

```
public void AddFeedLine(int line);
```

Visual Basic .NET

```
Public Sub AddFeedLine(line As Integer)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

紙送りを３行する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddFeedLine(3);
    //・・・処理・・・
} catch (Exception ex) {
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddFeedLine(3)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddImageAsync（画像圧縮）

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。

Windows.Grapics.Imaging.BitmapDecoder クラスのグラフィックを印字します。

Windows.Grapics.Imaging.BitmapDecoder 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。

- 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。
- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- ラスターイメージを高速に印字する場合、AddTextAlign (49 ページ) を Align.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できます。
- 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

Visual C#

```
public Windows.Foundation.IAsyncAction AddImageAsync
          (Windows.Grapics.Imaging.BitmapDecoder data,
           int x, int y, int width, int height,
           LibEposPrint.Color color,
           LibEposPrint.Mode mode,
           LibEposPrint.Halftone halftone,
           double brightness,
           LibEposPrint.Compress compress);
```

Visual Basic .NET

```
Public Function AddImageAsync
        (data As Windows.Graphics.Imaging.BitmapDecoder,
         x As Integer, y As Integer, width As Integer,
         height As Integer, color As LibEposPrint.Color,
         mode As LibEposPrint.Mode,
         halftone As LibEposPrint.Halftone,
         brightness As Double,
         compress As LibEposPrint.Compress)
         As Windows.Foundation.IAsyncAction
```

パラメーター

- data： Windows.Grapics.Imaging.BitmapDecoder 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、HResult に E_INVALIDARG(0x80070057) を格納した Exception が発生します。

- color： 色を指定します。

設定値	説明
Color.COLOR_NONE	非印字
Color.COLOR_1	第 1 色
Color.COLOR_2	第 2 色
Color.COLOR_3	第 3 色
Color.COLOR_4	第 4 色
Color.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				
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Mode.MODE_MONO	モノクロ (2 階調)	○	○	○	○	○	○	○
Mode.MODE_GRAY16	多階調 (16 階調)	-	-	-	-	○	○	○
Mode.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
Halftone.HALFTONE_DITHER	ディザー (グラフィックの印刷に適しています。)
Halftone.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。)
Halftone.HALFTONE_THRESHOLD	しきい値 (文字の印刷に適しています。)
Halftone.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)
Buider.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。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。

設定値	説明			TM プリンター				
		TM-P20	TM-P60II	TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II
Compress. COMPRESS_ DEFLATE	画像圧縮します	○	-	○	-	○	○	-
Compress. COMPRESS_NONE	画像圧縮しません	○	○	○	○	○	○	○
Compress.DEFAULT	既定値を選択 (画像圧縮しません)	○	○	○	○	○	○	○

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

- Visual C#

```
using Windows.Graphics.Imaging;

private async void sampleImageFunction()
  try
  {
      BitmapDecoder imageData = null;
      //・・・処理・・・
      Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
      await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                    Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0,
                    Compress.COMPRESS_ DEFLATE);
      //・・・処理・・・
  }
  catch (Exception ex)
  {
      if (ex.HResult == EposHResult.HR_E_FAIL)
      {
          //・・・処理・・・
      }
      //・・・処理・・・
  }
```

- Visual Basic .NET

```
Imports Windows.Graphics.Image

Private Async Sub sampleImageFunction()
  Dim ImageData As BitmapDecoder = Nothing
  `・・・処理・・・
  Try
      Dim builder As Builder
                      = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
      Await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                    Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0,
                    Compress.COMPRESS_ DEFLATE)
      `・・・処理・・・
  Catch ex As Exception
      If ex.HResult = EposHResult.HR_E_FAIL Then
         `・・・処理・・・
      End If
      `・・・処理・・・
  End Try
```

AddImageAsync

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。

Windows.Grapics.Imaging.BitmapDecoder クラスのグラフィックを印字します。

Windows.Grapics.Imaging.BitmapDecoder 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。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- ラスターイメージを高速に印字する場合、AddTextAlign (49 ページ) を Align.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できます。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction AddImageAsync
          (Windows.Grapics.Imaging.BitmapDecoder data,
           int x, int y, int width, int height,
           LibEposPrint.Color color,
           LibEposPrint.Mode mode,
           LibEposPrint.Halftone halftone,
           double brightness);
```

Visual Basic .NET

```
Public Function AddImageAsync
         (data As Windows.Graphics.Imaging.BitmapDecoder,
          x As Integer, y As Integer, width As Integer,
          height As Integer, color As LibEposPrint.Color,
          mode As LibEposPrint.Mode,
          halftone As LibEposPrint.Halftone,
          brightness As Double)
          As Windows.Foundation.IAsyncAction
```

パラメーター

- data：　Windows.Grapics.Imaging.BitmapDecoder 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、HResult に E_INVALIDARG(0x80070057) を格納した Exception が発生します。

- color： 色を指定します。

設定値	説明
Color.COLOR_NONE	非印字
Color.COLOR_1	第 1 色
Color.COLOR_2	第 2 色
Color.COLOR_3	第 3 色
Color.COLOR_4	第 4 色
Color.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						
		TM-T20II	TM-T70	TM-T70II	TM-T88V	TM-T90II	TM-P60II	TM-P20
Mode.MODE_MONO	モノクロ (2 階調)	○	○	○	○	○	○	○
Mode.MODE_GRAY16	多階調 (16 階調)	-	-	○	○	○	-	-
Mode.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
Halftone.HALFTONE_DITHER	ディザー (グラフィックの印刷に適しています。)
Halftone.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。)
Halftone.HALFTONE_THRESHOLD	しきい値 (文字の印刷に適しています。)
Halftone.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)
Buider.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

- Visual C#

```
using Windows.Graphics.Imaging;

private async void sampleImageFunction()
  try
  {
      BitmapDecoder imageData = null;
      //・・・処理・・・
      Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
      await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                      Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0);
      //・・・処理・・・
  }
  catch (Exception ex)
  {
      if (ex.HResult == EposHResult.HR_E_FAIL)
      {
          //・・・処理・・・
      }
      //・・・処理・・・
  }
```

- Visual Basic .NET

```
Imports Windows.Graphics.Image

Private Async Sub sampleImageFunction()
  Dim ImageData As BitmapDecoder = Nothing
  `・・・処理・・・
  Try
      Dim builder As Builder
                      = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
      Await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                      Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0)
      `・・・処理・・・
  Catch ex As Exception
      If ex.HResult = EposHResult.HR_E_FAIL Then
         `・・・処理・・・
      End If
      `・・・処理・・・
  End Try
```

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

- Visual C#

```
try
{
    BitmapDecoder imageData = null;
    //・・・処理・・・
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPagePosition(0, 255);
    await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                  Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0);
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim ImageData As BitmapDecoder = Nothing
    Dim builder As Builder
                      = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddTextPosition(0, 255)
    Await builder.AddImageAsync(imageData, 0, 0, 256, 256, Color.DEFAULT,
                  Mode.MODE_MONO, Halftone.HALFTONE_DITHER, 1.0)
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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できます。

構文

Visual C#

```
public void AddLogo(int key1, int key2);
```

Visual Basic .NET

```
Public Sub AddLogo(key1 As Integer, key2 As Integer)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddLogo(48, 48);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddLogo(48, 48)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

AddBarcode

バーコード印字を命令バッファーに追加します。

構文

Visual C#

```
public void AddBarcode
    (System.String data, LibEposPrint.BarcodeType type,
     LibEposPrint.Hri hri, LibEposPrint.Font font,
     int width, int height);
```

Visual Basic .NET

```
Public Sub AddBarcode
    (data As String, type As LibEposPrint.BarcodeType,
     hri As LibEposPrint.Hri, font As LibEposPrint.Font,
     width As Integer, height As Integer)
```

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。

種類	説明
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。先頭の文字が * 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。
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 (: {( ): {) *: {* {: {{

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

- hri：　HRI の位置を指定します。

設定値	説明
Hri.HRI_NONE(デフォルト)	印字しない
Hri.HRI_ABOVE	バーコードの上
Hri.HRI_BELOW	バーコードの下
Hri.HRI_BOTH	バーコードの上と下の両方
Hri.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
Font.FONT_A(デフォルト)	フォント A
Font.FONT_B	フォント B
Font.FONT_C	フォント C
Font.FONT_D	フォント D
Font.FONT_E	フォント E
Font.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddBarcode("01234567890", BarcodeType.BARCODE_UPC_A,Hri.HRI_BELOW,
             Font.UNSPECIFIED, 2, 64);
    builder.AddBarcode("01234500005", BarcodeType.BARCODE_UPC_E, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("201234567890", BarcodeType.BARCODE_EAN13, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("201234567890", BarcodeType.BARCODE_JAN13, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("2012345", BarcodeType.BARCODE_EAN8, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("2012345", BarcodeType.BARCODE_JAN8, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("ABCDE", BarcodeType.BARCODE_CODE39, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("012345", BarcodeType.BARCODE_ITF, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("A012345A", BarcodeType.BARCODE_CODABAR, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("ABCDE", BarcodeType.BARCODE_CODE93, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("{Babcde", BarcodeType.BARCODE_CODE128, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("(01)201234567890*", BarocdeType.BARCODE_GS1_128,
             Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
             Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("0201234567890",
             BarcodeType.BARCODE_GS1_DATABAR_OMNIDIRECTIONAL, Hri.UNSPECIFIED,
             Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("0201234567890", BarcodeType.BARCODE_GS1_DATABAR_TRUNCATED,
             Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
             Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("0201234567890", BarcodeType.BARCODE_GS1_DATABAR_LIMITED,
             Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
             Builder.PARAM_UNSPECIFIED);
    builder.AddBarcode("(01)2012345678903", BarcodeType.BARCODE_GS1_DATABAR_EXPANDED,
             Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
             Builder.PARAM_UNSPECIFIED);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddBarcode("01234567890", BarcodeType.BARCODE_UPC_A,Hri.HRI_BELOW,
                       Font.UNSPECIFIED, 2, 64)
    builder.AddBarcode("01234500005", BarcodeType.BARCODE_UPC_E, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("201234567890", BarcodeType.BARCODE_EAN13, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("201234567890", BarcodeType.BARCODE_JAN13, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("2012345", BarcodeType.BARCODE_EAN8, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("2012345", BarcodeType.BARCODE_JAN8, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("ABCDE", BarcodeType.BARCODE_CODE39, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("012345", BarcodeType.BARCODE_ITF, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("A012345A", BarcodeType.BARCODE_CODABAR, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("ABCDE", BarcodeType.BARCODE_CODE93, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("{Babcde", BarcodeType.BARCODE_CODE128, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
   builder.AddBarcode("(01)201234567890*", BarocdeType.BARCODE_GS1_128, Hri.UNSPECIFIED,
                       Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("0201234567890", BarcodeType.BARCODE_GS1_DATABAR_OMNIDIRECTIONAL,
                       Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("0201234567890", BarcodeType.BARCODE_GS1_DATABAR_TRUNCATED,
                       Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("0201234567890", BarcodeType.BARCODE_GS1_DATABAR_LIMITED,
                       Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    builder.AddBarcode("(01)2012345678903", BarcodeType.BARCODE_GS1_DATABAR_EXPANDED,
                       Hri.UNSPECIFIED, Font.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                       Builder.PARAM_UNSPECIFIED)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddSymbol
    (System.String data, LibEposPrint.SymbolType type,
     LibEposPrint.Level level1, int level2, int width,
     int height, int size);
```

Visual Basic .NET

```
Public Sub AddSymbol
    (data As String, type As LibEposPrint.SymbolType,
     level1 As LibEposPrint.Level, level2 As Integer,
     width As Integer, height As Integer,
     size As Integer)
```

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、最初のデータ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 で変換できるデータ) 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
SymbolType.SYMBOL_PDF417_STANDARD	Standard PDF417
SymbolType.SYMBOL_PDF417_TRUNCATED	Truncated PDF417
SymbolType.SYMBOL_QRCODE_MODEL_1	QR Code Model 1
SymbolType.SYMBOL_QRCODE_MODEL_2	QR Code Model 2
SymbolType.SYMBOL_MAXICODE_MODE_2	MaxiCode Mode 2
SymbolType.SYMBOL_MAXICODE_MODE_3	MaxiCode Mode 3
SymbolType.SYMBOL_MAXICODE_MODE_4	MaxiCode Mode 4
SymbolType.SYMBOL_MAXICODE_MODE_5	MaxiCode Mode 5
SymbolType.SYMBOL_MAXICODE_MODE_6	MaxiCode Mode 6
SymbolType.SYMBOL_GS1_DATABAR_STACKED	GS1 DataBar Stacked
SymbolType. SYMBOL_GS1_DATABAR_STACKED_OMNIDIRECTIONAL	GS1 DataBar Stacked Omnidirectional
SymbolType. SYMBOL_GS1_DATABAR_EXPANDED_STACKED	GS1 DataBar Expanded Stacked
SymbolType. SYMBOL_AZTECCODE_FULLRANGE	Aztec Code Full-Range モード
SymbolType. SYMBOL_AZTECCODE_COMPACT	Aztec Code Compact モード
SymbolType.SYMBOL_DATAMATRIX_SQUARE	DataMatrix 正方形
SymbolType. SYMBOL_DATAMATRIX_RECTANGLE_8	DataMatrix 長方形、行数 8
SymbolType. SYMBOL_DATAMATRIX_RECTANGLE_12	DataMatrix 長方形、行数 12
SymbolType. SYMBOL_DATAMATRIX_RECTANGLE_16	DataMatrix 長方形、行数 16

- level1： エラー訂正レベルを指定します。

設定値	説明
Level.LEVEL_0	PDF417 エラー訂正レベル 0
Level.LEVEL_1	PDF417 エラー訂正レベル 1
Level.LEVEL_2	PDF417 エラー訂正レベル 2
Level.LEVEL_3	PDF417 エラー訂正レベル 3
Level.LEVEL_4	PDF417 エラー訂正レベル 4
Level.LEVEL_5	PDF417 エラー訂正レベル 5
Level.LEVEL_6	PDF417 エラー訂正レベル 6
Level.LEVEL_7	PDF417 エラー訂正レベル 7
Level.LEVEL_8	PDF417 エラー訂正レベル 8
Level.LEVEL_L	QR Code エラー訂正レベル L
Level.LEVEL_M	QR Code エラー訂正レベル M
Level.LEVEL_Q	QR Code エラー訂正レベル Q
Level.LEVEL_H	QR Code エラー訂正レベル H
Level.LEVEL_DEFAULT	既定レベル
Level.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。
- Aztec Code/ MaxiCode/ 2 次元 GS1 DataBar/ DataMatrix の場合、Level.LEVEL_DEFAULT を選択してください。

- level2： エラー訂正レベルを指定します。

設定値	説明
5 ～ 95 の整数	Aztec Code エラー訂正レベル (パーセント単位)
Builder.PARAM_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。
- PDF417/ QR Code/ MaxiCode/ 2 次元 GS1 DataBar/ DataMatrix の場合、Builder.PARAM_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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_PDF417_STANDARD,
            Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_QRCODE_MODEL_2, Level.LEVEL_Q,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("908063840\\x1d850\\x1d001\\x1d\\x04",
            SymbolType.SYMBOL_MAXICODE_MODE_2, Level.UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("0201234567890", SymbolType.SYMBOL_GS1_DATABAR_STACKED,
            Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("0201234567890",
            SymbolType.SYMBOL_GS1_DATABAR_STACKED_OMNIDIRECTIONAL,
            Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("(01)02012345678903",
            SymbolType.SYMBOL_GS1_DATABAR_EXPANDED_STACKED, Level.UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_AZTECCODE_FULLRANGE,
            Level.UNSPECIFIED, 23, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED);
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_DATAMATRIX_SQUARE,
            Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
            Builder.PARAM_UNSPECIFIED);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_PDF417_STANDARD,
                      Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_QRCODE_MODEL_2, Level.LEVEL_Q,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("908063840\\x1d850\\x1d001\\x1d\\x04",
                      SymbolType.SYMBOL_MAXICODE_MODE_2, Level.UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("0201234567890", SymbolType.SYMBOL_GS1_DATABAR_STACKED,
                      Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("0201234567890",
                      SymbolType.SYMBOL_GS1_DATABAR_STACKED_OMNIDIRECTIONAL,
                      Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("(01)02012345678903",
                      SymbolType.SYMBOL_GS1_DATABAR_EXPANDED_STACKED,
                      Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_AZTECCODE_FULLRANGE,
                      Level.UNSPECIFIED, 23, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED)
    builder.AddSymbol("ABCDE", SymbolType.SYMBOL_DATAMATRIX_SQUARE,
                      Level.UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED, Builder.PARAM_UNSPECIFIED,
                      Builder.PARAM_UNSPECIFIED)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (100 ページ) と一緒にお使いください。

構文

Visual C#

```
public void AddPageBegin();
```

Visual Basic .NET

```
Public Sub AddPageBegin()
```

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddText("ABCDE");
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddText("ABCDE")
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98 ページ) と一緒にお使いください。

構文

Visual C#

```
public void AddPageEnd();
```

Visual Basic .NET

```
Public Sub AddPageEnd()
```

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddText("ABCDE");
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddText("ABCDE")
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98ページ) とAddPageEnd (100ページ)に挟んでお使いください。

構文

Visual C#

```
public void AddPageArea(int x, int y, int width,
                        int height);
```

Visual Basic .NET

```
Public Sub AddPageArea
                (x As Integer, y As Integer,
                 width As Integer, height As Integer)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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」を印字する場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPageArea(100, 50, 200, 30);
    builder.AddText("ABCDE");
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddPageArea(100, 50, 200, 30)
    builder.AddText("ABCDE")
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98 ページ) と AddPageEnd (100 ページ) に挟んでお使いください。

構文

Visual C#

```
public void AddPageDirection
                        (LibEposPrint.Direction dir);
```

Visual Basic .NET

```
Public Sub AddPageDirection
                        (dir As LibEposPrint.Direction)
```

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
Direction.DIRECTION_LEFT_TO_RIGHT (デフォルト)	回転しない (左上を始点に右方向へ印字)
Direction.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)
Direction.DIRECTION_RIGHT_TO_LEFT	180 度回転 (右下を始点に左方向へ印字)
Direction.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPageArea(100, 50, 200, 30);
    builder.AddPageDirection(Direction.DIRECTION_TOP_TO_BOTTOM);
    builder.AddText("ABCDE");
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddPageArea(100, 50, 200, 30)
    builder.AddPageDirection(Direction.DIRECTION_TOP_TO_BOTTOM)
    builder.AddText("ABCDE")
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98 ページ) と AddPageEnd (100 ページ) に挟んでお使いください。

構文

Visual C#

```
public void AddPagePosition(int x, int y);
```

Visual Basic .NET

```
Public Sub AddPagePosition(x As Integer, y As Integer)
```

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できます。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、省略できます。

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPageArea(100, 50, 200, 100);
    builder.AddPagePosition(50, 30);
    builder.AddText("ABCDE");
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddPageArea(100, 50, 200, 30)
    builder.AddPagePosition(50, 30)
    builder.AddText("ABCDE")
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98ページ) とAddPageEnd (100ページ)に挟んでお使いください。

構文

Visual C#

```
public void AddPageLine(int x1, int y1, int x2, int y2,
                        LibEposPrint.Line style);
```

Visual Basic .NET

```
Public Sub AddPageLine(x1 As Integer, y1 As Integer,
                       x2 As Integer, y2 As Integer,
                       style As LibEposPrint.Line)
```

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
Line.LINE_THIN	実線 : 細
Line.LINE_MEDIUM	実線 : 中太
Line.LINE_THICK	実線 : 太
Line.LINE_THIN_DOUBLE	二重線 : 細
Line.LINE_MEDIUM_DOUBLE	二重線 : 中太
Line.LINE_THICK_DOUBLE	二重線 : 太
Line.DEFAULT	実線 : 細

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPageLine(100, 0, 500, 0, Line.LINE_THIN);
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddPageLine(100, 0, 500, 0, Line.LINE_THIN)
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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 (98 ページ) と AddPageEnd (100 ページ) に挟んでお使いください。

構文

Visual C#

```
public void AddPageRectangle
                        (int x1, int y1, int x2,
                         int y2, LibEposPrint.Line style);
```

Visual Basic .NET

```
Public Sub AddPageRectangle
          (x1 As Integer, y1 As Integer, x2 As Integer,
           y2 As Integer, style As LibEposPrint.Line)
```

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
Line.LINE_THIN	実線 : 細
Line.LINE_MEDIUM	実線 : 中太
Line.LINE_THICK	実線 : 太
Line.LINE_THIN_DOUBLE	二重線 : 細
Line.LINE_MEDIUM_DOUBLE	二重線 : 中太
Line.LINE_THICK_DOUBLE	二重線 : 太
Line.DEFAULT	実線 : 細

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddPageBegin();
    builder.AddPageRectangle(100, 0, 500, 200, Line.LINE_THIN);
    builder.AddPageEnd();
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddPageBegin()
    builder.AddPageRectangle(100, 0, 500, 200, Line.LINE_THIN)
    builder.AddPageEnd()
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddCut(LibEposPrint.Cut type);
```

Visual Basic .NET

```
Public Sub AddCut(type As LibEposPrint.Cut)
```

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
Cut.CUT_NO_FEED	フィードなしカット (紙送りせずにカット)
Cut.CUT_FEED	フィードカット (紙送り後カット)
Cut.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)
Cut.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

フィードカットする場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
    builder.AddCut(Cut.CUT_FEED);
    //・・・処理・・・
}
catch
(Exception ex) {
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    `・・・処理・・・
    builder.AddCut(Cut.CUT_FEED)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddPulse(LibEposPrint.Drawer drawer,
                     LibEposPrint.Pulse time);
```

Visual Basic .NET

```
Public Sub AddPulse(drawer As LibEposPrint.Drawer,
                    time As LibEposPrint.Pulse)
```

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
Drawer.DRAWER_1	ドロアーキックコネクター 2 番ピン
Drawer.DRAWER_2	ドロアーキックコネクター 5 番ピン
Drawer.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通電時間を指定します。

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

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
    builder.AddPulse(Drawer.DRAWER_1, Pulse.PULSE_100);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    `・・・処理・・・
    builder.AddPulse(Drawer.DRAWER_1, Pulse.PULSE_100)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddSound(LibEposPrint.Pattern pattern,
                     int repeat, int cycle);
```

Visual Basic .NET

```
Public Sub AddSound(pattern As LibEposPrint.Pattern,
                    repeat As Integer, cycle As Integer)
```

パラメーター

- pattern： ブザーの音色を指定します。

設定値	説明
Pattern.PATTERN_A	パターン A
Pattern.PATTERN_B	パターン B
Pattern.PATTERN_C	パターン C
Pattern.PATTERN_D	パターン D
Pattern.PATTERN_E	パターン E
Pattern.PATTERN_ERROR	エラー鳴動パターン
Pattern.PATTERN_PAPER_END	用紙なし鳴動パターン
Pattern.PATTERN_1	パターン 1
Pattern.PATTERN_2	パターン 2
Pattern.PATTERN_3	パターン 3
Pattern.PATTERN_4	パターン 4
Pattern.PATTERN_5	パターン 5
Pattern.PATTERN_6	パターン 6
Pattern.PATTERN_7	パターン 7
Pattern.PATTERN_8	パターン 8
Pattern.PATTERN_9	パターン 9
Pattern.PATTERN_10	パターン 10
Pattern.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
    builder.AddSound(Pattern.PATTERN_1, 3, 1000);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    `・・・処理・・・
    builder.AddSound(Pattern.PATTERN_1, 3, 1000)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddFeedPosition
                        (LibEposPrint.Feed position);
```

Visual Basic .NET

```
Public Sub AddFeedPosition
                        (position As LibEposPrint.Feed)
```

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
Feed.FEED_PEELING	はくり位置まで紙送り
Feed.FEED_CUTTING	カット位置まで紙送り
Feed.FEED_CURRENT_TOF	現在のラベル頭出し位置まで紙送り
Feed.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

ラベル紙をはくり位置まで紙送りする場合

- Visual C#

```
try
{
    Builder builder = new Builder("TM-P60II", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
    builder.AddFeedPosition(Feed.FEED_PEELING);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-P60II", ModelLang.MODEL_LANG_JAPANESE)
    `・・・処理・・・
    builder.AddFeedPosition(Feed.FEED_PEELING)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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

Visual C#

```
public void AddLayout(LibEposPrint.Layout type,
                      int width, int height,
                      int marginTop, int marginBottom,
                      int offsetCut, int offsetLabel);
```

Visual Basic .NET

```
Public Sub AddLayout
                (type As LibEposPrint.Layout,
                 width As Integer, height As Integer,
                 marginTop As Integer,
                 marginBottom As Integer,
                 offsetCut As Integer,
                 offsetLabel As Integer)
```

パラメーター

- type : 用紙種類を指定します。

設定値	説明
Layout.LAYOUT_RECEIPT	レシート紙 (ブラックマークなし)
Layout.LAYOUT_LABEL	ラベル紙 (ブラックマークなし)
Layout.LAYOUT_LABEL_BM	ラベル紙 (ブラックマークあり)
Layout.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	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

- Visual C#

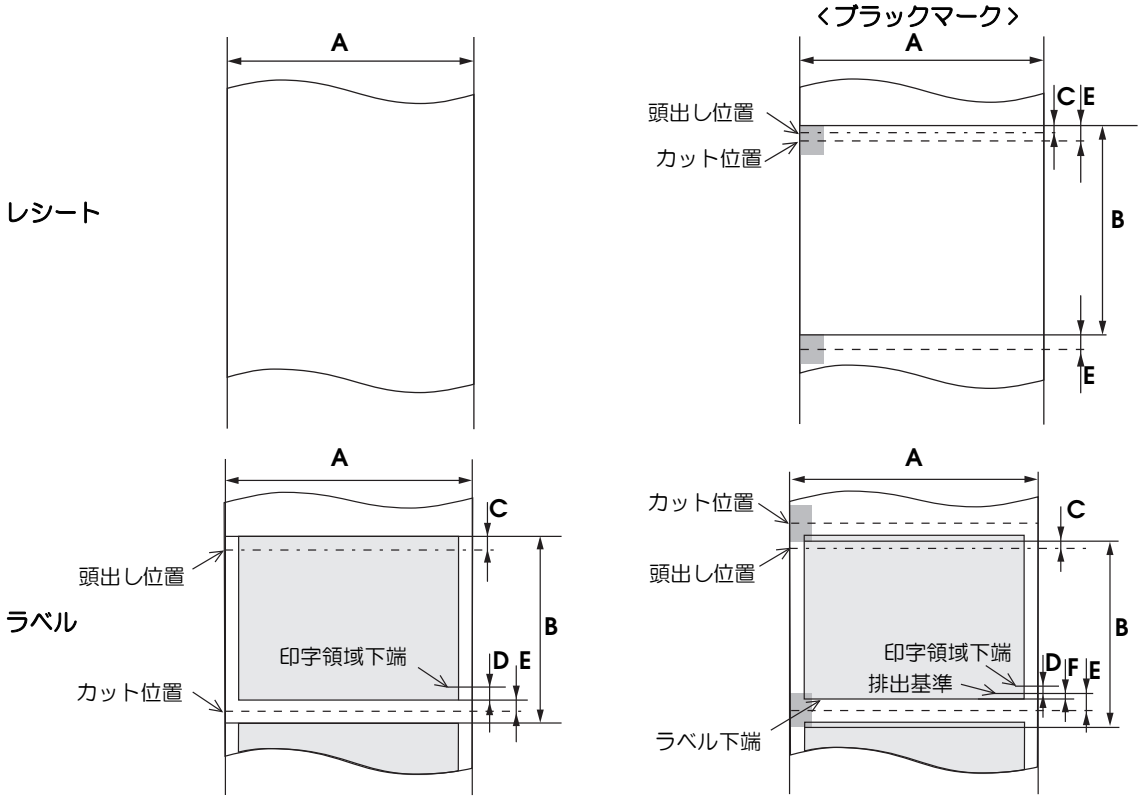
```
try
{
    Builder builder = new Builder("TM-P60II", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddLayout(Layout.LAYOUT_LABEL_BM, 600, 0, 15, -15, 15, 0);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-P60II", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddLayout(Layout.LAYOUT_LABEL_BM, 600, 0, 15, -15, 15, 0)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

詳細説明

❏ 用紙ごと指定できるパラメーターと、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

Visual C#

```
public void AddCommand(byte[] data);
```

Visual Basic .NET

```
Public Sub AddCommand(data() As Byte)
```

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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	その他のエラーが発生した。

例

- Visual C#

```
try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    byte[] data = null;
    //・・・処理・・・
    builder.AddCommand(data);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    Dim data As Byte() = Nothing
    `・・・処理・・・
    builder.AddCommand(data)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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。

構文

Visual C#

```
public Print();
```

Visual Basic .NET

```
Public Sub New()
```

例

- Visual C#

```
try
{
    Print printer = new Print();
    //・・・処理・・・
}
catch (Exception ex)
{
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim printer As Print = New Print()
    `・・・処理・・・
Catch ex As Exception
    `・・・処理・・・
End Try
```

OpenPrinterAsync

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 ClosePrinterAsync (132 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 は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。 • 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。 • プリンターとの接続時、接続確認の画面が表示される場合があります。そのため、本 API は UI スレッドから実行してください。 • 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 (35 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、ClosePrinterAsync (132 ページ) を呼び出してください。 • *Bluetooth* 接続の場合、本 API はペアリングされている状態で実行してください。 • 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 (210 ページ) を参照してください。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction OpenPrinterAsync
        (LibEposPrint.DevType deviceType,
         System.String deviceName,
         LibEposPrint.Monitoring enabled,
         int interval, int timeout);
```

Visual Basic .NET

```
Public Function OpenPrinterAsync
        (deviceType As LibEposPrint.DevType,
         deviceName As String,
         enabled As LibEposPrint.Monitoring,
         interval As Integer, timeout As Integer)
         As Windows.Foundation.IAsyncAction
```

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
DevType.DEVTYPE_TCP	Wi-Fi/Ethernet デバイス
DevType.DEVTYPE_*BLUETOOTH*	*Bluetooth* 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
DevType.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.DEVTYPE_*BLUETOOTH*	BD アドレス

- 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
Monitoring.MONITORING_TRUE	有効
Monitoring.MONITORING_FALSE	無効
Monitoring.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 DevType.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* 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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_ACCESSDENIED (0x80070005)	オープン処理に失敗した。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_ABORT (0x80004004)	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。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• 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。 • 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

- Visual C#

```
try
{
    Print printer = new Print();
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT,
                  Print.PARAM_DEFAULT);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim printer As Print = New Print()
    Await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT,
                  Print.PARAM_DEFAULT)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

OpenPrinterAsync (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 ClosePrinterAsync (132 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、OpenPrinterAsync (126 ページ) を使用してください。
- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- プリンターとの接続時、接続確認の画面が表示される場合があります。そのため、本 API は UI スレッドから実行してください。
- 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 (35 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、ClosePrinterAsync (132 ページ) を呼び出してください。
- *Bluetooth* 接続の場合、本 API はペアリングされている状態で実行してください。
- 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 (210 ページ) を参照してください。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction OpenPrinterAsync
        (LibEposPrint.DevType deviceType,
         System.String deviceName,
         LibEposPrint.Monitoring enabled,
         int interval);
```

Visual Basic .NET

```
Public Function OpenPrinterAsync
        (deviceType As LibEposPrint.DevType,
         deviceName As String,
         enabled As LibEposPrint.Monitoring,
         interval As Integer)
         As Windows.Foundation.IAsyncAction
```

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
DevType.DEVTYPE_TCP	Wi-Fi/Ethernet デバイス
DevType.DEVTYPE_*BLUETOOTH*	*Bluetooth* 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
DevType.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.DEVTYPE_*BLUETOOTH*	BD アドレス

- 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
Monitoring.MONITORING_TRUE	有効
Monitoring.MONITORING_FALSE	無効
Monitoring.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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_ACCESSDENIED (0x80070005)	オープン処理に失敗した。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• 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。 • 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

- Visual C#

```
try
{
    Print printer = new Print();
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT);
    //・・・処理・・・
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim printer As Print = New Print()
    Await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
    `・・・処理・・・
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

ClosePrinterAsync

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。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction
                    ClosePrinterAsync();
```

Visual Basic .NET

```
Public Function ClosePrinterAsync()
                    As Windows.Foundation.IAsyncAction
```

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_FAIL (0x80004005)	• 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。 • その他のエラーが発生した。

例

- Visual C#

```
try
{
    Print printer = new Print();
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT);
    //・・・処理・・・
    await printer.ClosePrinterAsync();
}
catch (Exception ex)
{
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Dim printer As Print = New Print()
    Await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
    `・・・処理・・・
    Await printer.ClosePrinterAsync()
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

SendDataAsync

Builder クラスで作成した印刷ドキュメントを送信します。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- *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。
- 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 (210 ページ) を参照してください。

構文

Visual C#

```
public Windows.Foundation.IAsyncOperation
  <PrinterStatus> SendDataAsync
  (LibEposPrint.Builder builder, int timeout);
```

Visual Basic .NET

```
Public Function SendDataAsync
  (builder As LibEposPrint.Builder, timeout As Integer)
   As Windows.Foundation.IAsyncOperation
  (Of LibEposPrint.PrinterStatus)
```

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 (43 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。
  0 ～ 600000(ミリ秒単位) の整数値を指定します。

timeout に指定されたタイムアウトは、印刷ドキュメントの送信 から、プリンターステータスを取得するまでの時間です。

戻り値

プリンターステータスと、バッテリーステータスが返ります。

- プリンターステータスは、PrinterStatus.printerStatus に返ります。
  プリンターステータスの詳細は、プリンターステータス一覧 (40 ページ) を参照してください。
- バッテリーステータスは、PrinterStatus.batteryStatus に返ります。
  バッテリーステータスの詳細は、バッテリーステータス (41 ページ) を参照してください。

OpenPrinterAsync (126 ページ) のプリンターステータスのモニタリングの設定で、返されるプリンターステータスとバッテリーステータスが異なります。

プリンターステータスのモニタリングの設定	説明
有効	最後にモニタリングしたプリンターステータスとバッテリーステータスが返ります。
無効	印刷ドキュメント送信終了時のプリンターステータスが返ります。 取得できなかった場合、プリンターステータスは ST_NO_RESPONSE が、バッテリーステータスは 0x0000 が返ります。

例外が発生した場合、プリンターステータスとバッテリーステータスは、例外処理時に GetPrinterStatus (165 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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_ABORT (0x80004004)	指定された時間内に全データを送信できなかった。
HR_E_ACCESSDENIED (0x80070005)	• 通信エラーが発生した。 • プリンターがオフライン状態だった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• 通信が開始していない状態で本 API が呼び出された。 • その他のエラーが発生した。

HR_E_ACCESSDENIED が発生した場合、プリンターがオフライン状態かどうかは、プリンターステータスを確認して識別します。プリンターステータスが、"ST_NO_RESPONSE" が設定されていない、かつ "ST_OFF_LINE" が設定されている場合、プリンターがオフライン状態です。

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

- Visual C#

```
Print printer = new Print();
PrinterStatus status;
status.printerStatus = 0;
status.batteryStatus = 0;

try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    builder.AddText("ABCDE\n");

    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                  Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT);

    status = await printer.SendDataAsync(builder, 10000);

    //・・・処理・・・

    await printer.ClosePrinterAsync();

}
catch (Exception ex)
{
    PrinterStatus errStatus = printer.GetPrinterStatus(ex.HResult);
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Dim printer As Print = New Print()
Dim status As PrinterStatus
status.printerStatus = 0
status.batteryStatus = 0

Try

    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddText("ABCDE" + vbCrLF)

    status = Awiat printer.OpenPrinterAsync(DevType.DEVTYPE_TCP,
                              "192.168.192.168", Monitoring.MONITORING_TRUE,
                              Print.PARAM_DEFAULT)
    status = Await printer.SendDataAsync(builder, 10000)

    `・・・処理・・・

    Await printer.ClosePrinterAsync()
Catch ex As Exception
    Dim errStatus As PrinterStatus = printer.GetPrinterStatus(ex.HResult)
    `・・・処理・・・
End Try
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetStatusChangeEventCallback
            (LibEposPrint.StatusChangeEvent target);
```

Visual Basic .NET

```
Public Sub SetStatusChangeEventCallback
           (target As LibEposPrint.StatusChangeEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName, int status);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String, status As Integer)
```

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 / 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

- Visual C#

```
private void Status_Change_Event(string deviceName, int status)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();
    StatusChangeEvent statusChangeEvent
                                        = new StatusChangeEvent(Status_Change_Event);
    printer.SetStatusChangeEventCallback(statusChangeEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Status_Change_Event(deviceName As String, status As Integer)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim statusChangeEvent As StatusChangeEvent
                             = New StatusChangeEvent(AddressOf statusChangeEvent)
    printer.SetStatusChangeEventCallback(statusChangeEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetOnlineEventCallback
            (LibEposPrint.OnlineEvent target);
```

Visual Basic .NET

```
Public Sub SetOnlineEventCallback
           (target As LibEposPrint.OnlineEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Online_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    OnlineEvent onlineEvent = new OnlineEvent(Online_Event);
    printer.SetOnlineEventCallback(onlineEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Online_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim onlineEvent As OnlineEvent = New OnlineEvent(AddressOf Online_Event)
    printer.SetOnlineEventCallback(onlineEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetOfflineEventCallback
            (LibEposPrint.OfflineEvent target);
```

Visual Basic .NET

```
Public Sub SetOfflineEventCallback
           (target As LibEposPrint.OfflineEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名) がセットされます。

例

- Visual C#

```
private void Offline_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    OfflineEvent offlineEvent = new OfflineEvent(Offline_Event);
    printer.SetOfflineEventCallback(offlineEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Offline_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim offlineEvent As OfflineEvent = New OfflineEvent(AddressOf Offline_Event)
    printer.SetOfflineEventCallback(offlineEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetPowerOffEventCallback
            (LibEposPrint.PowerOffEvent target);
```

Visual Basic .NET

```
Public Sub SetPowerOffEventCallback
           (target As LibEposPrint.PowerOffEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名) がセットされます。

例

- Visual C#

```
private void Power_Off_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    PowerOffEvent powerOffEvent = new PowerOffEvent(Power_Off_Event);
    printer.SetPowerOffEventCallback(powerOffEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Power_Off_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim powerOffEvent As PowerOffEvent
                       = New PowerOffEvent(AddressOf Power_Off_Event)
    printer.SetPowerOffEventCallback(powerOffEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                      Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetCoverOkEventCallback
            (LibEposPrint.CoverOkEvent target);
```

Visual Basic .NET

```
Public Sub SetCoverOkEventCallback
           (target As LibEposPrint.CoverOkEvent)
```

パラメーター

- target :　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Cover_Ok_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    CoverOkEvent coverOkEvent = new CoverOkEvent(Cover_Ok_Event);
    printer.SetCoverOkEventCallback(coverOkEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Cover_Ok_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim coverOkEvent As CoverOkEvent = New CoverOkEvent(AddressOf Cover_Ok_Event)
    printer.SetCoverOkEventCallback(coverOkEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetCoverOpenEventCallback
            (LibEposPrint.CoverOpenEvent target);
```

Visual Basic .NET

```
Public Sub SetCoverOpenEventCallback
           (target As LibEposPrint.CoverOpenEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Cover_Open_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    CoverOpenEvent coverOpenEvent = new CoverOpenEvent(Cover_Open_Event);
    printer.SetCoverOpenEventCallback(coverOpenEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Cover_Open_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim coverOpenEvent As CoverOpenEvent
                        = New CoverOpenEvent(AddressOf Cover_Open_Event)
    printer.SetCoverOpenEventCallback(coverOpenEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetPaperOkEventCallback
            (LibEposPrint.PaperOkEvent target);
```

Visual Basic .NET

```
Public Sub SetPaperOkEventCallback
           (target As LibEposPrint.PaperOkEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Paper_Ok_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    PaperOkEvent paperOkEvent = new PaperOkEvent(Paper_Ok_Event);
    printer.SetPaperOkEventCallback(paperOkEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Paper_Ok_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim paperOkEvent As PaperOkEvent = New PaperOkEvent(AddressOf Paper_Ok_Event)
    printer.SetPaperOkEventCallback(paperOkEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetPaperNearEndEventCallback
            (LibEposPrint.PaperNearEndEvent target);
```

Visual Basic .NET

```
Public Sub SetPaperNearEndEventCallback
           (target As LibEposPrint.PaperNearEndEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Paper_Near_End_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    PaperNearEndEvent paperNearEndEvent
                                        = new PaperNearEndEvent(Paper_Near_End_Event);
    printer.SetPaperNearEndEventCallback(paperNearEndEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Paper_Near_End_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim paperNearEndEvent As PaperNearEndEvent
                          = New PaperNearEndEvent(AddressOf Paper_Near_End_Event)
    printer.SetPaperNearEndEventCallback(paperNearEndEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                         Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetPaperEndEventCallback
            (LibEposPrint.PaperEndEvent target);
```

Visual Basic .NET

```
Public Sub SetPaperEndEventCallback
           (target As LibEposPrint.PaperEndEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Paper_End_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    PaperEndEvent paperEndEvent = new PaperEndEvent(Paper_End_Event);
    printer.SetPaperEndEventCallback(paperEndEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Paper_End_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim paperEndEvent As PaperEndEvent
                      = New PaperEndEvent(AddressOf Paper_End_Event)
    printer.SetPaperEndEventCallback(paperEndEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetDrawerClosedEventCallback
            (LibEposPrint.DrawerClosedEvent target);
```

Visual Basic .NET

```
Public Sub SetDrawerClosedEventCallback
           (target As LibEposPrint.DrawerClosedEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Drawer_Closed_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    DrawerClosedEvent drawerCLosedEvent
                                        = new DrawerClosedEvent(Drawer_Closed_Event);
    printer.SetDrawerClosedEventCallback(drawerCLosedEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Drawer_Closed_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim Dim drawerClosedEvent As DrawerClosedEvent
                        = New DrawerClosedEvent(AddressOf Drawer_Closed_Event)
    printer.SetDrawerClosedEventCallback(drawerClosedEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetDrawerOpenEventCallback
            (LibEposPrint.DrawerOpenEvent target);
```

Visual Basic .NET

```
Public Sub SetDrawerOpenEventCallback
           (target As LibEposPrint.DrawerOpenEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Drawer_Open_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    DrawerOpenEvent drawerOpenEvent = new DrawerOpenEvent(Drawer_Open_Event);
    printer.SetDrawerOpenEventCallback(drawerOpenEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Drawer_Open_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim drawerOpenEvent As DrawerOpenEvent
                        = New DrawerOpenEvent(AddressOf Drawer_Open_Event)
    printer.SetDrawerOpenEventCallback(drawerOpenEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetBatteryLowEventCallback
            (LibEposPrint.BatteryLowEvent target);
```

Visual Basic .NET

```
Public Sub SetBatteryLowEventCallback
           (target As LibEposPrint.BatteryLowEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String)
```

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 / プリンターホスト名）がセットされます。

例

- Visual C#

```
private void Battery_Low_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    BatteryLowEvent batteryLowEvent = new BatteryLowEvent(Battery_Low_Event);
    printer.SetBatteryLowEventCallback(batteryLowEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Battery_Low_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim batteryLowEvent As BatteryLowEvent
                        = New BatteryLowEvent(AddressOf Battery_Low_Event)
    printer.SetBatteryLowEventCallback(batteryLowEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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ときに通知されるイベントです。

- 本 API は、OpenPrinterAsync (126 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

Visual C#

```
public void SetBatteryOkEventCallback
            (LibEposPrint.BatteryOkEvent target);
```

Visual Basic .NET

```
Public Sub SetBatteryOkEventCallback
           (target As LibEposPrint.BatteryOkEvent)
```

パラメーター

- target :　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名 (System.String deviceName);
```

Visual Basic .NET

```
Sub メソッド名 (deviceName As String)
```

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 / プリンターホスト名) がセットされます。

例

- Visual C#

```
private void Battery_Ok_Event(string deviceName)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();

    BatteryOkEvent batteryOkEvent = new BatteryOkEvent(Battery_Ok_Event);
    printer.SetBatteryOkEventCallback(batteryOkEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Battery_Ok_Event(deviceName As String)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim batteryOkEvent As BatteryOkEvent
                        = New BatteryOkEvent(AddressOf Battery_Ok_Event)
    printer.SetBatteryOkEventCallback(batteryOkEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                        Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

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、OpenPrinterAsync (126 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

Visual C#

```
public void SetBatteryStatusChangeEventCallback
        (LibEposPrint.BatteryStatusChangeEvent target);
```

Visual Basic .NET

```
Public Sub SetBatteryStatusChangeEventCallback
       (target As LibEposPrint.BatteryStatusChangeEvent)
```

パラメーター

- target：　通知先メソッドを持つオブジェクトを指定します。
  通知先の登録を解除する場合、以下を指定します。

開発言語	コード
Visual C#	null
Visual Basic .NET	Nothing

通知先メソッド

Visual C#

```
void メソッド名(System.String deviceName, int battery);
```

Visual Basic .NET

```
Sub メソッド名(deviceName As String, battery As Integer)
```

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 / 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

- Visual C#

```
private void Battery_Status_Change_Event(string deviceName, int battery)
{
    //・・・処理・・・
}

private async void openPrinter()
{
    Print printer = new Print();
    BatteryStatusChangeEvent batteryStatusChangeEvent
                  = new BatteryStatusChangeEvent(Battery_Status_Change_Event);
    printer.SetBatteryStatusChangeEventCallback(batteryStatusChangeEvent);

    try
    {
        await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                       Monitoring.MONITORING_TRUE,
                                       Print.PARAM_DEFAULT);
        //・・・処理・・・
    }
    catch (Exception ex)
    {
        if (ex.HResult == EposHResult.HR_E_FAIL)
        {
            //・・・処理・・・
        }
    }
}
```

- Visual Basic .NET

```
Private Sub Battery_Status_Change_Event(deviceName As String, battery As Integer)
  '・・・処理・・・
End Sub

Private Async Sub openPrinter()
    Dim printer As Print = New Print()

    Dim batteryStatusChangeEvent As BatteryStatusChangeEvent
            = New BatteryStatusChangeEvent(AddressOf Battery_Status_Change_Event)
    printer.SetBatteryStatusChangeEventCallback(batteryStatusChangeEvent)

    Try
        Await printer.OpenPrintAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                      Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
        '・・・処理・・・
    Catch ex As Exception
        if ex.HResult = EposHResult.HR_E_FAIL Then
            '・・・処理・・・
        End If
      '・・・処理・・・
    End Try
End Sub
```

GetPrinterStatus

SendDataAsync（134 ページ）で発生した例外から、プリンターステータスとバッテリーステータスを取得します。

構文

Visual C#

```
public LibEposPrint.PrinterStatus GetPrinterStatus
                    (int hresult);
```

Visual Basic .NET

```
Public Function GetPrinterStatus(hresult As Integer)
                As LibEposPrint.PrinterStatus
```

パラメーター

- hresult：　例外の HResult を指定します。

戻り値

プリンターステータスと、バッテリーステータスが返ります。

- プリンターステータスは、PrinterStatus.printerStatus に返されます。
  プリンターステータスの詳細は、プリンターステータス一覧（40 ページ）を参照してください。
- バッテリーステータスは、PrinterStatus.batteryStatus に返されます。
  バッテリーステータスの詳細は、バッテリーステータス（41 ページ）を参照してください。

例

Exception からプリンターステータスを取得する場合

- Visual C#

```
Print printer = new Print();
PrinterStatus status;
status.printerStatus = 0;
status.batteryStatus = 0;

try
{
    Builder builder = new Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE);
    //・・・処理・・・
    await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT);
    status = await printer.SendDataAsync(builder, 10000);
    //・・・処理・・・
    await printer.ClosePrinterAsync();
}
catch (Exception ex)
{
    status = printer.GetPrinterStatus(ex.HResult);
    //・・・処理・・・
    if((status.printerStatus & Print.ST_PRINT_SUCCESS) == Print.ST_PRINT_SUCCESS)
    {
        //・・・処理・・・
    }
}
```

- Visual Basic .NET

```
Dim printer As Print = New Print()
Dim status As PrinterStatus
status.printerStatus = 0
status.batteryStatus = 0

Try
    Dim builder As Builder = New Builder("TM-T88V", ModelLang.MODEL_LANG_JAPANESE)
    builder.AddText("ABCDE" + vbCrLf)

    Await printer.OpenPrinterAsync(DevType.DEVTYPE_TCP, "192.168.192.168",
                                Monitoring.MONITORING_TRUE, Print.PARAM_DEFAULT)
    status = Await printer.SendDataAsync(builder, 10000)
    Await printer.ClosePrinterAsync()

    `・・・処理・・・

Catch ex As Exception
    Dim errStatus As PrinterStatus = printer.GetPrinterStatus(ex.HResult)
    If (status.printerStatus And Print.ST_PRINT_SUCCESS)
                                    = Print.ST_PRINT_SUCCESS Then
        `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

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（167 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
StartAsync	プリンター検索を開始	167
StopAsync	プリンターとの通信を終了	169
GetDeviceInfoList	プリンターの検索結果を取得	170
GetResult (旧フォーマット)		172

StartAsync

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 StopAsync (169 ページ) で検索終了してください。

- TCP デバイスの検索時、ネットワークアダプターが複数存在する場合、ホップ数が最も少ないアダプターを検索します。
- 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。
- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public static Windows.Foundation.IAsyncAction
        StartAsync
        (LibEposPrint.IoDevType deviceType,
         System.String findOption);
```

Visual Basic .NET

```
Public Shared Function StartAsync
                (deviceType As LibEposPrint.IoDevType,
                 findOption As String)
                 As Windows.Foundation.IAsyncAction
```

パラメーター

- 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
IoDevType.TCP	ネットワークに接続された TM プリンターを検索します。
IoDevType.*BLUETOOTH*	ペアリング済みのシリアルポートサービス接続された *Bluetooth* 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
IoDevType.TCP	検索範囲のブロードキャストアドレス
IoDevType.*BLUETOOTH*	""(空文字)

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• すでに検索を開始した状態で本 API が呼び出された。 • その他のエラーが発生した。

StopAsync

プリンター検索を終了します。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public static Windows.Foundation.IAsyncAction
        StopAsync();
```

Visual Basic .NET

```
Public Shared Function StopAsync()
                  As Windows.Foundation.IAsyncAction
```

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_FAIL (0x80004005)	• すでに検索を開始した状態で本 API が呼び出された。 • 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。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public static DeviceInfo[]
            GetDeviceInfoList(LibEposPrint.IoFilterOption
            filterOption);
```

Visual Basic .NET

```
Public Shared Function GetDeviceInfoList(filterOption As
            LibEposPrint.IoFilterOption) As DeviceInfo()
```

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 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が返されます。(DeviceInfo)
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
IoDevType.TCP	DeviceType	IoDevType.TCP(固定)
	PrinterName	プリンターモデル名
	DeviceName	• DHCP 無効の場合 : IP アドレス • DHCP 有効の場合 : MAC アドレス
	IpAddress	IP アドレス
	MacAddress	MAC アドレス

deviceType	DeviceInfo	取得する情報
IoDevType.*BLUETOOTH*	DeviceType	IoDevType.*BLUETOOTH*(固定)
	PrinterName	*Bluetooth* デバイス名
	DeviceName	BD アドレス (MAC アドレスと同じ形式)
	IpAddress	"" (空文字)
	MacAddress	"" (空文字)

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• すでに検索を開始した状態で本 API が呼び出された。 • 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。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public static System.String[] GetResult();
```

Visual Basic .NET

```
Public Shared Function GetResult() As String()
```

戻り値

検索したデバイスのリストが返されます。(deviceList)
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
IoDevType.TCP	プリンターの IP アドレスのリスト
IoDevType.*BLUETOOTH*	*Bluetooth* デバイスの BD アドレスのリスト

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• 検索を開始していない状態で本 API が呼び出された。 • その他のエラーが発生した。

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（173 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	173

SetLogSettings

ログ出力機能を設定します。

TCP のログ出力を有効にした場合、環境によって各 API の処理に時間がかかることがあります。

構文

Visual C#

```
public static void SetLogSettings
        (LibEposPrint.LogPeriod period,
         LibEposPrint.LogEnabled enabled,
         System.String ipAddress, int port,
         int logSize, LibEposPrint.LogLevel logLevel);
```

Visual Basic .NET

```
Public Shared Sub SetLogSettings
            (period As LibEposPrint.LogPeriod,
             enabled As LibEposPrint.LogEnabled,
             ipAddress As String, port As Integer,
             logSize As Integer,
             logLevel As LibEposPrint.LogLevel)
```

パラメーター

- period：　ログ出力機能の設定方法を指定します。

設定値	説明
LogPeriod.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。
LogPeriod.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。

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

- 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、"" (空文字) も指定できます。
* LogEnabled.LOG_DISABLE
* LogEnabled.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。
* LogEnabled.LOG_DISABLE
* LogEnabled.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。
* LogEnabled.LOG_DISABLE
* LogEnabled.LOG_TCP

- logLevel： ログの出力レベルを指定します。

設定値	説明
LogLevel.LOG_LOW	低レベル

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_FAIL (0x80004005)	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

- Visual C#

```
try
{
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_TCP,
                       “192.168.192.168”, 8080, 10, LogLevel.LOG_LOW);
} catch (Exception ex) {
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_TCP,
                       "192.168.192.168", 8080, 10, LogLevel.LOG_LOW)
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

❏ 端末のストレージにログを出力する場合

- Visual C#

```
try
{
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_STORAGE, "",
                       0, 10, LogLevel.LOG_LOW);
} catch (Exception ex) {
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_STORAGE, "",
                       0, 10, LogLevel.LOG_LOW)
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

❑ ログ出力機能を無効にする場合

- Visual C#

```
try
{
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_DISABLE, "",
                       0, 10, LogLevel.LOG_LOW);
} catch (Exception ex) {
    if (ex.HResult == EposHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
    //・・・処理・・・
}
```

- Visual Basic .NET

```
Try
    Log.SetLogSettings(LogPeriod.LOG_PERMANENT, LogEnabled.LOG_DISABLE, "",
                       0, 10, LogLevel.LOG_LOW)
Catch ex As Exception
    If ex.HResult = EposHResult.HR_E_FAIL Then
       `・・・処理・・・
    End If
    `・・・処理・・・
End Try
```

コマンドの送受信

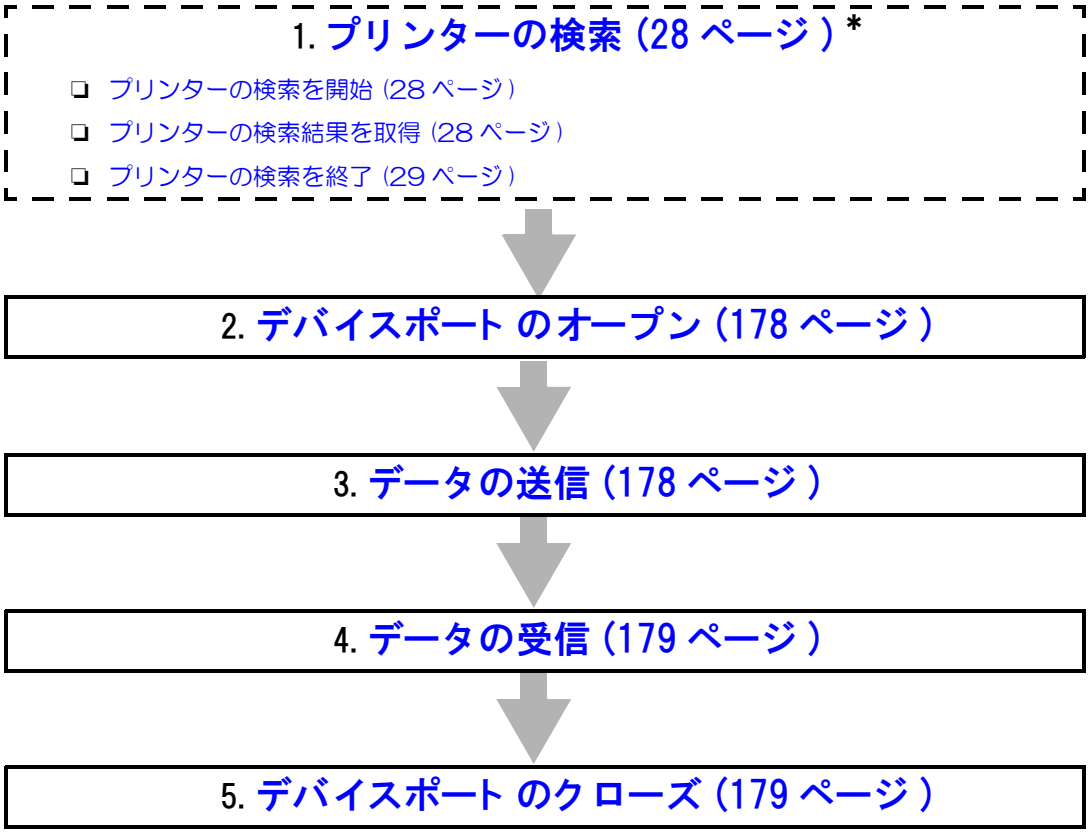
本章では、コマンド (ESC/POS コマンドなど) を送受信するための API について説明しています。

- コマンド送受信 API は、ePOS-Print API の Print クラス (45 ページ) と同時に使用できません。
- 本章のソースコードを使った説明は、Visual C# で説明しています。他の言語は、適宜読み替えてください。

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。



*: 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 OpenAsync（182 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。

```
try {
//EpsonIo クラスの生成
    EpsonIo port = new EpsonIo();

// デバイスポートのオープン
////Wi-Fi/Ethernet デバイスの場合
    async port.OpenAsync(IoDevType.TCP, "192.168.192.168", "");
////Bluetooth デバイスの場合
    async port.OpenAsync(IoDevType.BLUETOOTH, "00:00:12:34:56:78", "");
}
// 例外処理
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

データの送信

EpsonIo クラスの WriteBytes（185 ページ）を使ってデータをバッファリングし、WriteAsync（186 ページ）で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
System.String str = "Hello World!\n";
byte[] data = System.Text.Encoding.GetEncoding("shift_jis").GetBytes(str);
int offset = 0;
int size = data.length;
int timeout = 5000;
int sizeCopy = 0;
int sizeWritten = 0;

try {
// データのバッファリング
    sizeCopy = port.WriteBytes(data, offset, size);

// データの送信
    sizeWritten = await port.WriteAsync(timeout);
}
// 例外処理
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

データの受信

EpsonIo クラスの ReadAsync（190 ページ）を使ってプリンターからのデータを受信し、ReadBytes（188 ページ）を使ってデータを読み取りします。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 sizeCopy = 0;

try {
// データの受信
    sizeRead = await port.ReadAsync(size, timeout);

// データの読み取り
    sizeCopy = port.ReadBytes(out data, data, offset, size);
}
// 例外処理
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

デバイスポートのクローズ

EpsonIo クラスの CloseAsync（184 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。

```
try {
    EpsonIo port = new EpsonIo();
    async port.OpenAsync(IoDevType.TCP, "192.168.192.168", "");
    // デバイスのクローズ
    async port.CloseAsync();
}
// 例外処理
catch (Exception ex)
{
    int errCode = ex.HResult;
}
```

例外処理

コマンド送受信 API は、エラー発生時に Windows のシステムの例外を発生させ、呼び元にエラーを通知します。

処理方法

以下のプログラミングを参考にしてください。

```
// 送信設定
System.String str = "Hello World!\n";
byte[] data = System.Text.Encoding.GetEncoding("shift_jis").GetBytes(str);
int offset = 0;
int size = data.length;
int timeout = 5000;
int sizeCopy = 0;
int sizeWritten = 0;

try {
// データのバッファリング
    sizeCopy = port.WriteBytes(data, offset, size);
// データの送信
    sizeWritten = await port.WriteAsync(timeout);
}
// 例外処理
catch (Exception ex)
{
    if(ex.HResult == IoHResult.HR_E_FAIL)
    {
        //・・・処理・・・
    }
}
```

エラー値一覧

ePOS-Print SDK for Windows ストアアプリでは、Windows のシステムエラーコードにマッピングして、エラーステータスを返します。

エラー値	要因
HR_E_INVALIDARG (0x80070057)	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。
HR_E_ACCESSDENIED (0x80070005)	• オープン処理に失敗した。 • デバイスとの通信に失敗した。 < 例 > • TCP、*Bluetooth* 通信用の Socket の作成に失敗した。 • 指定したパスのデバイスファイルがオープンできなかった。 • 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。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。

エラー値	要因
HR_E_PENDING (0x8000000A)	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。
HR_E_FAIL (0x80004005)	• 不適切な方法で使用された。 • その他のエラーが発生した。 < 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。

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
OpenAsync	デバイスポートのオープン	182
CloseAsync	デバイスポートのクローズ	184
WriteBytes	送信データの書き込み	185
WriteAsync	データ送信	186
ReadBytes	受信データの読み取り	188
ReadAsync	データ受信	190

OpenAsync

指定されたデバイスポートをオープンします。

- プリンターとの通信が不要になった場合、必ず CloseAsync (184 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。
- 同一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。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- プリンターとの接続時、接続確認の画面が表示される場合があります。そのため、本 API は UI スレッドから実行してください。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction OpenAsync
            (LibEposPrint.IoDevType deviceType,
             System.String deviceName,
             System.String deviceSettings);
```

Visual Basic .NET

```
Public Function OpenAsync
       (deviceType As LibEposPrint.IoDevType,
        deviceName As String, deviceSettings As String)
        As Windows.Foundation.IAsyncAction
```

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
IoDevType.TCP	オープンするプリンターが Wi-Fi/Ethernet のときに指定します。
IoDevType.*BLUETOOTH*	オープンするプリンターが *Bluetooth* のときに指定します。

- 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
IoDevType.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)
IoDevType.*BLUETOOTH*	BD アドレス

- deviceSettings：
  ""（空文字）を指定します。

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_ACCESSDENIED (0x80070005)	オープン処理に失敗した。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• 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。 • その他のエラーが発生した。

CloseAsync

指定されたデバイスポートをクローズします。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。

構文

Visual C#

```
public Windows.Foundation.IAsyncAction CloseAsync();
```

Visual Basic .NET

```
Public Function CloseAsync()
                As Windows.Foundation.IAsyncAction
```

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_FAIL (0x80004005)	• オープンしていない状態で本 API が呼び出された。 • その他のエラーが発生した。

WriteBytes

送信データを Epsonlo クラスのバッファーに書き込みます。

構文

Visual C#

```
public int WriteBytes(byte[] data, int offset,
                      int size);
```

Visual Basic .NET

```
Public Function WriteBytes(data() As Byte,
          offset As Integer, size As Integer) As Integer
```

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、Epsonlo クラスのバッファーには書き込まれません。この場合、戻り値に 0 が返ります。

戻り値

Epsonlo クラスのバッファーに書き込まれている、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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• オープンしていない状態で本 API が呼び出された。 • その他のエラーが発生した。

WriteAsync (186 ページ) が実行中に、本 API が実行された場合、排他制御により HR_E_PENDING の例外が発生します。

WriteAsync

Epsonlo クラスのバッファーをデバイスポートへ送信します。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- プリンター側で印刷が完了する前に CloseAsync (184 ページ) を呼び出した場合、データ送信が中断されることがあります。

構文

Visual C#

```
public Windows.Foundation.IAsyncOperation<int>
            WriteAsync(int timeout);
```

Visual Basic .NET

```
Public Function WriteAsync(timeout As Integer) As
    Windows.Foundation.IAsyncOperation(Of Integer)
```

パラメーター

- timeout : 送信待ちのタイムアウト時間を、msec 単位で指定します。
  指定できる最長時間は 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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_ACCESSDENIED (0x80070005)	通信エラーが発生した。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• オープンしていない状態で本 API が呼び出された。 • その他のエラーが発生した。

- 本 API が同一オブジェクトから同時に実行された場合、排他制御により所有権を持たない処理は HR_E_PENDING の例外が発生します。
- TM プリンターの接続が切れた後、20 秒以内に本 API を呼び出した場合、*Bluetooth* デバイス制御の仕様で、例外は発生しません。

ReadBytes

受信データを Epsonlo クラスのバッファーから読み込みます。

本 API は、Epsonlo クラスのバッファーにデータがある限り、複数回実行できます。

構文

Visual C#

```
public int ReadBytes(out byte[] outData, byte[] data,
                     int offset, int size);
```

Visual Basic .NET

```
Public Function ReadBytes
          (ByRef outData() As Byte, data() As Byte,
           offset As Integer, size As Integer) As Integer
```

パラメーター

- outData：　受信データの格納先バッファーです。(ePOS-Print SDK が確保するバッファー)
- data：　受信データの格納先バッファーです。(本 API で確保するバッファー)
- 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：　受信できるバイト数を指定します。

size に 0 が指定された場合、読み込めません。この場合、戻り値に 0 が返ります。

戻り値

Epsonlo クラスのバッファーから読み込んだ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 HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_ACCESSDENIED (0x80070005)	通信エラーが発生した。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_FAIL (0x80004005)	• オープンしていない状態で本 API が呼び出された。 • その他のエラーが発生した。

ReadAsync (190 ページ) が実行中に、本 API が実行された場合、排他制御により HR_E_PENDING の例外が発生します。

ReadAsync

データをデバイスポートから Epsonlo クラスのバッファーに受信します。

- 本 API は非同期 API です。次の処理を実行するには、本 API の処理が終了するのを待ってから実行してください。
- 本 API は、Windows.Foundation.IAsyncInfo.Cancel() による非同期操作の取り消しには対応していません。
- 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

Visual C#

```
public Windows.Foundation.IAsyncOperation<int>
            ReadAsync(int size, int timeout);
```

Visual Basic .NET

```
Public Function ReadAsync
       (size As Integer, timeout As Integer) As
        Windows.Foundation.IAsyncOperation(Of Integer)
```

パラメーター

- size：　受信したい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。
  指定できる最長時間は 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

受信したデータの合計バイト数が返されます。

本 API を複数回実行した場合、受信データはすべて加算されます。

例外

処理に失敗した場合、以下の HResult を返す Exception が発生します。

HResult	説明
HR_E_INVALIDARG (0x80070057)	不正なパラメーターが渡された。
HR_E_PENDING (0x8000000A)	処理を実行できなかった。
HR_E_OUTOFMEMORY (0x8007000E)	処理に必要なメモリーが確保できなかった。
HR_E_ACCESSDENIED (0x80070005)	通信エラーが発生した。
HR_E_FAIL (0x80004005)	• オープンしていない状態で本 API が呼び出された。 • その他のエラーが発生した。

本 API が同一オブジェクトから同時に実行された場合、排他制御により所有権を持たない処理は HR_E_PENDING の例外が発生します。

付録

プリンターの仕様

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

	58 mm 仕様
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	カット位置への紙送り
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

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddFeedPosition	118
AddImageAsync (画像圧縮)	75	AddImageAsync	79
AddLogo	83	AddBarcode	85
AddSymbol	91	AddPageBegin	98
AddPageEnd	100	AddPageArea	102
AddPageDirection	104	AddPagePosition	106
AddPageLine	108	AddPageRectangle	110
AddCut	112	AddSound	116
AddLayout	120	AddCommand	123
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetBatteryLowEventCallback	159	SetBatteryOkEventCallback	161
SetBatteryStatusChangeEventCallback	163	GetPrinterStatus	165

コマンド送受信 API はすべてサポートしています。

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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddFeedPosition	118
AddImageAsync (画像圧縮)	75	AddImageAsync	79
AddLogo	83	AddBarcode	85
AddSymbol	91	AddPageBegin	98
AddPageEnd	100	AddPageArea	102
AddPageDirection	104	AddPagePosition	106
AddPageLine	108	AddPageRectangle	110
AddCut	112	AddSound	116
AddLayout	120	AddCommand	123
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetBatteryLowEventCallback	159	SetBatteryOkEventCallback	161
SetBatteryStatusChangeEventCallback	163	GetPrinterStatus	165

コマンド送受信 API はすべてサポートしています。

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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddImageAsync (画像圧縮)	75
AddImageAsync	79	AddLogo	83
AddBarcode	85	AddSymbol	91
AddPageBegin	98	AddPageEnd	100
AddPageArea	102	AddPageDirection	104
AddPagePosition	106	AddCut	112
AddPulse	114	AddSound	116
AddCommand	123		
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetDrawerClosedEventCallback	155	SetDrawerOpenEventCallback	157
GetPrinterStatus	165		

コマンド送受信 API はすべてサポートしています。

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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddImageAsync (画像圧縮)	75
AddImageAsync	79	AddLogo	83
AddBarcode	85	AddSymbol	91
AddPageBegin	98	AddPageEnd	100
AddPageArea	102	AddPageDirection	104
AddPagePosition	106	AddCut	112
AddPulse	114	AddCommand	123
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetDrawerClosedEventCallback	155	SetDrawerOpenEventCallback	157
GetPrinterStatus	165		

コマンド送受信 API はすべてサポートしています。

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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddImageAsync (画像圧縮)	75
AddImageAsync	79	AddLogo	83
AddBarcode	85	AddSymbol	91
AddPageBegin	98	AddPageEnd	100
AddPageArea	102	AddPageDirection	104
AddPagePosition	106	AddCut	112
AddPulse	114	AddSound	116
AddCommand	123		
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetDrawerClosedEventCallback	155	SetDrawerOpenEventCallback	157
GetPrinterStatus	165		

コマンド送受信 API はすべてサポートしています。

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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddImageAsync (画像圧縮)	75
AddImageAsync	79	AddLogo	83
AddBarcode	85	AddSymbol	91
AddPageBegin	98	AddPageEnd	100
AddPageArea	102	AddPageDirection	104
AddPagePosition	106	AddCut	112
AddPulse	114	AddSound	116
AddCommand	123		
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetCoverOpenEventCallback	147
SetPaperOkEventCallback	149	SetPaperEndEventCallback	153
SetDrawerClosedEventCallback	155	SetDrawerOpenEventCallback	157
GetPrinterStatus	165		

コマンド送受信 API はすべてサポートしています。

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, 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	

ePOS-Print API サポート一覧

ePOS-Print API	ページ	ePOS-Print API	ページ
Builder クラス			
コンストラクター	46	ClearCommandBuffer	48
AddTextAlign	49	AddTextLineSpace	51
AddTextRotate	53	AddText	55
AddTextLang	57	AddTextFont	59
AddTextSmooth	61	AddTextDouble	63
AddTextSize	65	AddTextStyle	67
AddTextPosition	69	AddFeedUnit	71
AddFeedLine	73	AddImageAsync (画像圧縮)	75
AddImageAsync	79	AddLogo	83
AddBarcode	85	AddSymbol	91
AddPageBegin	98	AddPageEnd	100
AddPageArea	102	AddPageDirection	104
AddPagePosition	106	AddCut	112
AddPulse	114	AddCommand	123
Print クラス			
コンストラクター	125	OpenPrinterAsync	126
OpenPrinterAsync(旧フォーマット)	129	ClosePrinterAsync	132
SendDataAsync	134	SetStatusChangeEventCallback	137
SetOnlineEventCallback	139	SetOfflineEventCallback	141
SetPowerOffEventCallback	143	SetCoverOkEventCallback	145
SetCoverOpenEventCallback	147	SetPaperOkEventCallback	149
SetPaperNearEndEventCallback	151	SetPaperEndEventCallback	153
SetDrawerClosedEventCallback	155	SetDrawerOpenEventCallback	157
GetPrinterStatus	165		

コマンド送受信 API はすべてサポートしています。

注意事項

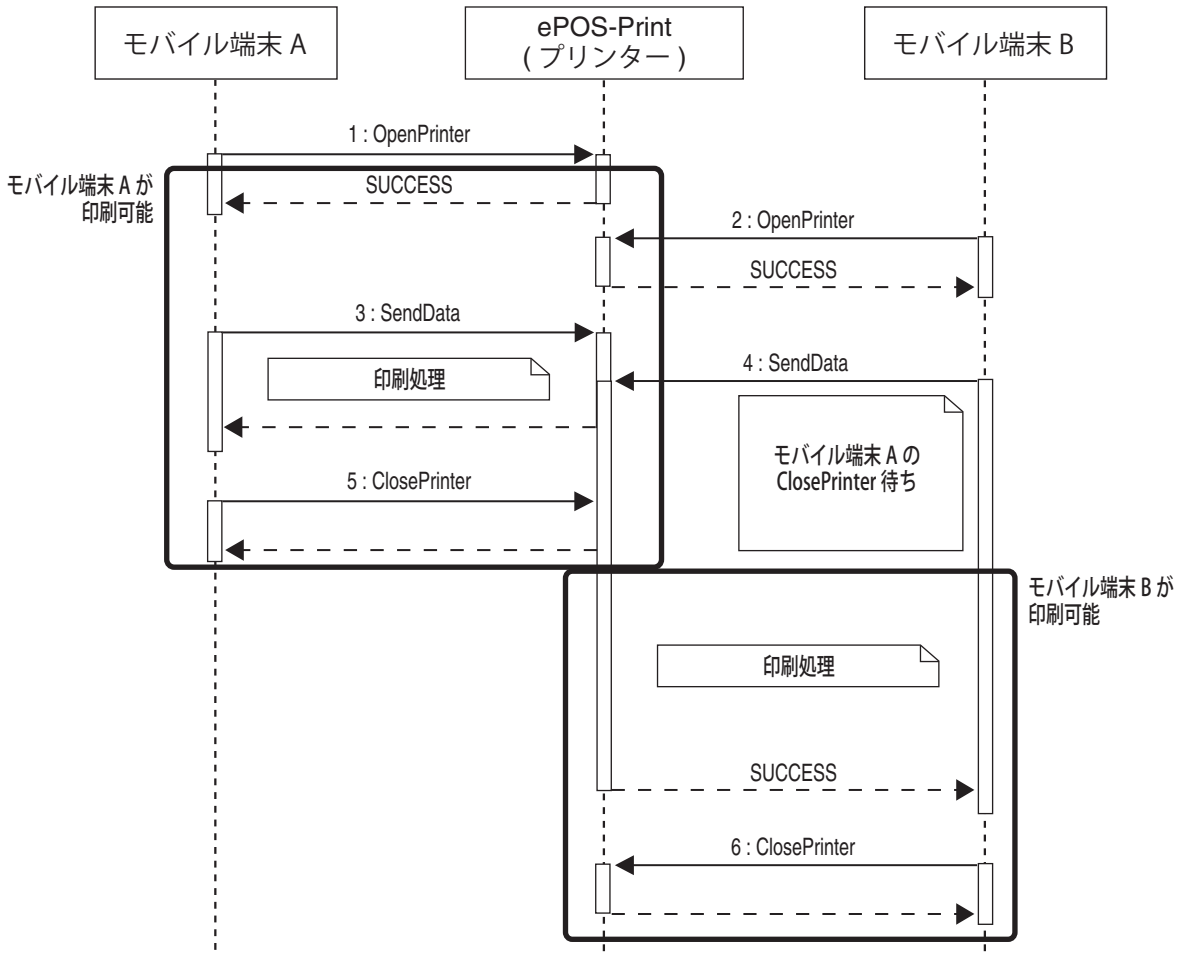
一台のプリンターを複数のモバイル端末から使用する場合

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。