

PDF Imager-LP

Version 1.9.1

株式会社トラスト・ソフトウェア・システム
2022 年 10 月

目次

1.0 はじめに	1
2.0 利用環境および PDF のバージョンと画像フォーマット	1
3.0 インストール	1
3.1 ファイルの概要	1
3.2 .NET インターフェース	2
3.3 C/C++ インターフェース	2
3.4 Python インターフェース	3
4.0 関数 - .NET、C/C++開発環境	3
4.1 インスタンス化および初期化	3
4.2 ライセンス情報の表示または取得	4
4.3 初期化ファイル（または初期化データ）を指定する	4
4.4 入力ファイル（PDF または画像データ）を開く	5
4.4.1 PDFファイルの許可フラグを無視して開く	5
4.4.2 PDFファイルを指定モードで開く	5
4.4.3 画像ファイルを開く	6
4.5 PDF 文書の情報	6
4.5.1 PDF文書のページ数取得	6
4.5.2 PDF文書のパーミッション（許可）フラグ	6
4.5.3 PDF文書の暗号化レビジョン	7
4.5.4 PDF文書のメタデータ	7
4.5.5 PDF文書内のフォント名	8
4.5.6 インターラクティブ・フォーム	9
4.6 画像ファイルに変換する関数群	11
4.6.1 指定された1ページを画像に変換（解像度と画像品質を指定しない）	11
4.6.2 指定された1ページを画像に変換（解像度と画像品質を指定する）	12
4.6.3 すべてのページを TIFF 形式画像に変換	12
4.6.4 ページ範囲を指定して TIFF 形式画像に変換	13
4.6.5 ページのリストを指定して TIFF 形式画像に変換	13
4.6.6 ページ範囲を指定して TIFF 形式画像に変換（解像度を指定する）	14
4.6.7 ページのリストを指定して TIFF 形式画像に変換（解像度を指定する）	15
4.7 画像（ピクセル）データに変換する関数群	16
4.7.1 指定されたページをピクセルデータに変換	16
4.7.2 連続する複数ページをピクセル画像に変換	16
4.7.3 不連続な複数ページをピクセル画像に変換	17
4.7.4 指定されたページをライブラリ内部データに変換	17

4.7.5 ライブラリ内部の画像データを取得	18
4.7.6 ライブラリ内部の画像データをインデックスで取得	18
4.7.7 ライブラリ内部の画像データをページ番号で取得	19
4.7.8 ライブラリ内部の画像データを削除	19
4.8 出力画像の色空間を指定する	20
4.8.1 RGB カラー画像を指定	20
4.8.2 CMYK カラー画像を指定	20
4.8.3 グレースケール画像を指定	20
4.8.4 白黒ディザ画像を指定	20
4.8.5 白黒2階調画像を指定	21
4.9 カラープロファイル	21
4.9.1 カラープロファイルの指定	21
4.9.2 色空間変換手順指定	22
4.10 画像のサイズ	23
4.10.1 ページの大きさ(境界)指定	23
4.10.2 画像解像度の設定	24
4.10.3 画像のピクセルサイズを取得	24
4.10.4 画像幅のピクセルサイズを取得	25
4.10.5 画像高さのピクセルサイズを取得	25
4.10.6 画像のピクセルサイズ指定	26
4.10.7 キャンバスサイズの設定	26
4.10.8 キャンバスサイズ変更の原点	27
4.10.9 画像の論理サイズ指定	28
4.10.10 ページの境界ボックスのサイズを取得	29
4.10.11 ページの回転角を取得	30
4.11 TIFF 形式画像の圧縮指定	30
4.11.1 圧縮しない	30
4.11.2 JPEG 圧縮指定	30
4.11.3 Deflate 圧縮指定	31
4.11.4 LZW 圧縮指定	31
4.12 JPEG 圧縮の品質指定	31
4.13 BMP 画像の形式を指定する	31
4.14 変換の詳細を指定する	32
4.14.1 塗りつぶしパスのアンチエイリアスを抑制する	32
4.14.2 塗りつぶしパスのアンチエイリアスを実施する	32
4.14.3 文字コード0(ゼロ)の表示	33
4.14.4 極細い線の描画	33
4.14.5 PDFのバージョン指定	34

4.15 代替フォント指定	34
4.15.1 既定フォントの指定	35
4.15.2 代替フォントの指定	35
4.16 OCG（レイヤー）	36
4.16.1 OCG の総数取得	36
4.16.2 OCG の状態取得	36
4.16.3 OCG の表示・非表示指定	37
4.17 透かし（ウォーターマーク）	38
4.17.1 透かしの設定	38
4.17.2 透かしの削除	39
4.17.3 文字色指定	39
4.17.4 透かしのエスケープ文字	40
4.18 コールバック関数	41
4.18.1 ページ解析終了（画像変換開始）のコールバック設定	41
4.18.2 画像生成進捗のコールバック設定	41
4.18.3 パスストローク描画時のコールバック設定	42
4.18.4 タイル画像定義時のコールバック設定	43
4.18.5 タイルマスク画像定義時のコールバック設定	44
4.18.6 タイル処理時のコールバック設定	45
4.19 コールバックの抑制	46
4.19.1 パスストローク描画時コールバックの抑制	46
4.19.2 タイル画像定義時コールバックの抑制	46
4.19.3 タイルマスク画像定義時コールバックの抑制	46
4.19.4 タイル処理時コールバックの抑制	47
4.20 変換された画像をパネルに描画	47
4.20.1 パネル作成	47
4.20.2 パネルをクリア	48
4.20.3 パネルにページ画像を貼り付ける	48
4.20.4 パネルにページ画像の境界を指定して貼り付ける	49
4.20.5 パネルに矩形を描画	49
4.20.6 パネルのデータを画像で取得	50
4.20.7 パネル削除	50
4.21 画像変換の並列処理	51
4.21.1 複数ページの並列処理の抑制	51
4.21.2 複数ページの並列処理の最大スレッド数	51
4.21.3 単一ページの並列処理の抑制	51
4.21.4 単一ページの並列処理の最大スレッド数	52
4.22 画像変換の結果	52

4.23	メタデータ	52
4.24	PDF 文書処理の終了	53
4.25	ライブラリの使用終了	53
5.0	XmpInterface インターフェース	54
5.1	XmpInterface の取得	54
5.2	XmpInterface の終了	54
5.3	メタデータを更新	54
5.4	PDF 文書をファイルに格納	55
5.5	プロパティの有無および削除	55
5.6	Simple (単純) プロパティ	56
5.6.1	プロパティの有無	56
5.6.2	プロパティ値の取得および設定	56
5.6.3	プロパティの削除	57
5.7	Array (配列) プロパティ	57
5.7.1	プロパティの有無	57
5.7.2	プロパティ、アイテムの追加	58
5.7.3	アイテム数	58
5.7.4	アイテム値の取得および設定	59
5.7.5	アイテムの削除	59
5.8	Structure (構造) プロパティ	60
5.8.1	プロパティ、フィールドの有無	60
5.8.2	フィールド値の取得および設定	60
5.8.3	フィールドの削除	61
5.9	日付のプロパティ値	61
5.10	独自名前空間	62
5.11	XMP データをダンプする	62
5.12	エラーの詳細	62
6.0	関数 – Python 開発環境	63
6.1	PdfImagerLP モジュール	63
6.2	インスタンス化および初期化	63
6.3	ライセンス情報の表示または取得	63
6.4	入力ファイル (PDF または画像データ) を開く	64
6.4.1	PDF ファイルの許可フラグを無視して開く	64
6.4.2	PDF ファイルを指定モードで開く	64
6.4.3	画像ファイルを開く	65
6.5	PDF 文書の情報	65
6.5.1	PDF 文書のページ数取得	65
6.5.2	PDF 文書のメタデータ	65

6.5.3 PDF文書内のフォント名	66
6.5.4 インターラクティブ・フォーム	66
6.6 画像ファイルに変換する関数群	67
6.6.1 指定された1ページを画像に変換(解像度と画像品質を指定しない)	67
6.6.2 指定された1ページを画像に変換(解像度と画像品質を指定する)	68
6.6.3 すべてのページを TIFF 形式画像に変換	68
6.6.4 ページ範囲を指定して TIFF 形式画像に変換	69
6.6.5 ページのリストを指定して TIFF 形式画像に変換	69
6.6.6 ページ範囲を指定して TIFF 形式画像に変換(解像度を指定する)	70
6.6.7 ページのリストを指定して TIFF 形式画像に変換(解像度を指定する)	71
6.7 画像(ピクセル)データに変換する関数群	71
6.8 出力画像の色空間を指定する	72
6.8.1 RGB カラー画像を指定	72
6.8.2 CMYK カラー画像を指定	72
6.8.3 グレースケール画像を指定	72
6.8.4 白黒ディザ画像を指定	73
6.8.5 白黒2階調画像を指定	73
6.9 カラープロファイル	74
6.9.1 カラープロファイルの指定	74
6.9.2 色空間変換手順指定	75
6.10 画像のサイズ	76
6.10.1 ページの大きさ	76
6.10.2 画像解像度の設定	77
6.10.3 画像のピクセルサイズを取得	77
6.10.4 画像幅のピクセルサイズを取得	78
6.10.6 画像のピクセルサイズ指定	78
6.10.7 キャンバスサイズの設定	79
6.10.8 キャンバスサイズ変更の原点	79
6.10.9 画像の論理サイズ指定	80
6.10.10 ページの境界ボックスのサイズを取得	80
6.10.11 ページの回転角を取得	80
6.11 TIFF 形式画像の圧縮指定	81
6.11.1 圧縮しない	81
6.11.2 JPEG 圧縮指定	81
6.11.3 Deflate 圧縮指定	81
6.11.4 LZW 圧縮指定	82
6.12 JPEG 圧縮の品質指定	82
6.13 BMP 画像の形式を指定する	82

6.14 変換の詳細を指定する	83
6.14.1 塗りつぶしパスのアンチエイリアスを抑制する	83
6.14.2 塗りつぶしパスのアンチエイリアスを実施する	83
6.14.3 文字コード0(ゼロ)の表示	83
6.14.4 極細い線の描画	84
6.14.5 PDFページの境界指定	84
6.14.6 PDFのバージョン指定	84
6.15 代替フォント指定	85
6.15.1 既定フォントの指定	85
6.15.2 代替フォントの指定	85
6.16 OCG (レイヤー)	86
6.16.1 OCG の総数取得	86
6.16.2 OCG の状態取得	86
6.16.3 OCG の表示・非表示指定	87
6.17 透かし (ウォーターマーク)	88
6.17.1 透かしの設定	88
6.17.2 透かしを削除	89
6.18 コールバック関数	89
6.19 変換された画像をパネルに描画	90
6.19.1 パネル作成	90
6.19.2 パネルをクリア	90
6.19.3 パネルにページ画像を貼り付ける	91
6.19.4 パネルにページ画像の境界を指定して貼り付ける	91
6.19.5 パネルに矩形を描画	92
6.19.6 パネルのデータを画像で取得	92
6.19.7 パネル削除	93
6.20 画像変換の並列処理	93
6.20.1 複数ページの並列処理の抑制	93
6.20.2 複数ページの並列処理の最大スレッド数	93
6.20.3 単一ページの並列処理の抑制	94
6.20.4 単一ページの並列処理の最大スレッド数	94
6.21 PDF 文書処理の終了	94
6.22 ライブラリの使用終了	94
7.0 PDF 文書の構成要素抽出関数	95
7.1 現在文書の xref	95
7.2 現在文書の trailer	95
7.3 現在文書の catalog	96
7.4 現在文書の document information	96

8.0 初期化ファイル(初期化データ)について	97
8.1 非埋め込みフォントの代替指定	97
8.2 埋め込みフォントの代替指定	98
8.3 代替フォントの太さとスタイル	99
9.0 文字列でのページ指定方法	100
10.0 著作権	101

1.0 はじめに

PDF Imager-LP は、PDF (Portable Document Format) 文書 (または画像データ) を画像に変換する機能をアプリケーションに追加するライブラリです。

PDFに指定されたカラーでの画像化だけではなく、グレースケール/二値画像やディザ変換した画像を生成できます。また、PDFで指定されたフォントを別のフォントに代替して変換できます。入力データに画像を指定した場合も同様です。

PDF Imager-LP は、PostScript タイプの XObject を画像に変換しません。

2.0 利用環境および PDF のバージョンと画像フォーマット

PDF Imager-LP は、以下の環境で利用できます。

利用環境	Windows 8、8.1、10、11(32ビットおよび64ビット) Windows Server 2012、2012 R2、2016、2019、2022 (32ビットおよび64ビット)
開発環境	C#、C/C++、VB.NET
画像フォーマット	PNG(Portable Network Graphics)、JPEG(Joint Photographic Experts Group)、TIFF (Tagged Image File Format) 形式、BMP(Device Independent Bitmap)形式 TIFF形式は、非圧縮、Deflate圧縮、JPEG 圧縮、LZW 圧縮を生成します。 BMP形式は非圧縮(WindowsまたはOS/2)を生成します。

PDFのバージョン PDF 1.4 から PDF 1.7 および PDF 2.0(全てではありません)を変換します。

PDF Imager-LP は、パスワードで暗号化されたPDF文書を画像に変換できます。(暗号化されたPDF文書を画像に変換するには、パスワードが必要です。)

PDF Imager-LP は入力データとしてPDF形式以外に種々の画像データを指定できますが、出力画像形式として指定できるのは上記の通りです。

なお、PDF Imager-LP は PostScript タイプの XObject を画像に変換しません。

3.0 インストール

PDF Imager-LP には、以下のフォルダーおよびファイルが含まれます。

doc	「PDF Imager-LP 説明書」および「使用許諾契約書」
include	C/C++で使用するためのヘッダファイル、他
lib	ライブラリ群
sample	C#/VB.NET、C/C++(Visual Studio プロジェクト) サンプル コード

開発環境に応じて適切なフォルダーでご利用ください。

なお、PDF Imager-LP を使用するためには、適切なライセンスキーが必要です。

3.1 ファイルの概要

PDF Imager-LP に含まれるファイルの概要です。

lib/x64/PdfImagerLP.dll	PDF画像変換のためのネイティブDLL(x64専用)です。
lib/Win32/PdfImagerLP.dll	PDF画像変換のためのネイティブDLL(win32専用)です。
lib/x64/PdfImagerLP.lib	C/C++(x64)開発環境の場合にリンクして使用します。
lib/Win32/PdfImagerLP.lib	C/C++(win32)開発環境の場合にリンクして使用します。
lib/PdfImagerLPNET.dll	PDF画像変換機能をC#(またはVB.NET)で利用するためのラッパーDLLです。

<code>include/ImagerLP.h</code>	C/C++用のヘッダファイルです。C/C++での開発時に使用します。
<code>sample/init.xml</code>	代替フォントを指定する初期化ファイルの例です。
<code>lib/x64/_ImagerLP.pyd</code>	Imager-LP ライブラリを Python で利用するためのネイティブDLL(x64専用)です。PdfImagerLP.dll と同じフォルダーに格納してください。
<code>lib/Win32/_ImagerLP.pyd</code>	Imager-LP ライブラリを Python で利用するためのネイティブDLL(win32専用)です。PdfImagerLP.dll と同じフォルダーに格納してください。
<code>Lib/x64/ImagerLP.py</code>	Pythonにimportするモジュールです。PdfImagerLP.dll と同じフォルダーに格納してください。
<code>Lib/Win32/ImagerLP.py</code>	Pythonにimportするモジュールです。PdfImagerLP.dll と同じフォルダーに格納してください。

3.2 .NET インターフェース

PDF画像変換ライブラリ(PdfImagerLP.dll)は、.NETアセンブリではありません。C#またはVB.NETから利用するための.NETアセンブリDLL(PdfImagerLPNET.dll)を参照して画像変換します。開発時にはPdfImagerLPNET.dllを「参照設定」に追加する必要があります。

これらのDLLは、コンパイル・実行時において適切に参照できるようにしてください。

ネイティブDLL(PdfImagerLP.dll)は32ビット環境用および64ビット環境用に最適化されています。必ず開発・利用環境に沿ったDLLを使用してください。

ネイティブDLL(PdfImagerLP.dll)の32ビット用と64ビット用をサブフォルダ「Win32」と「x64」に配置できます。これらをサブフォルダに配置すると、.NETアセンブリDLL(PdfImagerLPNET.dll)は利用環境に合った適切なネイティブDLLを動的に使用します。

名前空間：
 PDFTools.PdfImagerLP
クラス名：
 PdfImager

以下の手順で PdfImager をインスタンス化してください。
 PdfImager mlp = new PdfImager();

3.3 C/C++ インターフェース

ネイティブC/C++開発環境では、ヘッダファイル(ImagerLP.h)を利用できるようにし、ライブラリ(PdfImagerLP.lib)をリンクしてください。実行環境では PdfImagerLP.dll を適切なフォルダーに配置してください。

メソッドやプロパティと同じ機能の関数をC/C++開発環境で使用するために接頭辞「Mlp」が追加されて用意されています。

3.4 Python インターフェース

Python開発環境では、モジュール `ImagerLP` をインポートします。このモジュールはネイティブDLL (`_ImagerLP.pyd` および `ImagerLP.dll`) を参照しますので、適切なフォルダーに格納する、もしくはDLLの検索パスを適切に設定してください。

以下はPython DLLの検索パス設定手順の例です。

```
import sys
if sys.maxsize > 2**32:
    sys.path.insert(1, '<インストールFolder名>/lib/x64')
else:
    sys.path.insert(1, '<インストールFolder名>/lib/Win32')
```

<インストールFolder名>は `Imager-LP` をインストールしたフォルダー名です。

4.0 関数 – .NET、C/C++開発環境

C#/VB.NET、およびC/C++で利用する関数です。C/C++で利用する場合は、接頭辞「`Mlp`」の付いた関数名に読み替えてください。

4.1 インスタンス化および初期化

PDF Imager-LP ライブラリはインスタンス化およびライセンスキーを使った初期化が必要です。まず `PDFTool.PdfImagerLP.PdfImager` クラスをインスタンス化します。続いて、以下のメソッドで初期化します。ライブラリはその使用後に `Uninitialize` メソッドを使って開放します。

メソッド

```
int Initialize(string license)
void InitializeWException(string license)    // C#のみ
```

引数

license PDF Imager-LP を使用するためのライセンス文字列

戻り値

ライブラリ初期化に成功すると0(ゼロ)が戻ります。それ以外は、エラーコードです。
`InitializeWException` メソッドは失敗すると `Exception` をスローします。

使用例

```
using PDFTools.PdfImagerLP;

using (var mlp = new PdfImager())
{
    try
    {
        mlp.InitializeWException("ライセンスキー文字列");
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    ... // ここで変換などを実施します。
    mlp.Uninitialize();
}
```

4.2 ライセンス情報の表示または取得

PDF Imager-LP の初期化の後はライセンスキー内容を文字列で表示または取得ができます。
ShowLicenseInfo は結果をコンソールに出力し、LicenseInfoStr は結果を文字列で戻します。

メソッド

```
void ShowLicenseInfo()
```

引数

ありません

プロパティ (get)

```
string LicenseInfoString
```

戻り値

ShowLicenseInfo は、戻り値がありません。
LicenseInfoStr は、成功するとライセンスを説明する文字列が戻ります。

4.3 初期化ファイル(または初期化データ)を指定する

PDF Imager-LP は、PDF文書で指定されたフォントの代替などを初期化ファイルまたは初期化データで指定できます。初期化ファイル(初期化データ)はXML形式です。

初期化ファイル(初期化データ)でのフォントの代替などの詳細は、「8.0 初期化ファイル(初期化データ)について」を参照してください。

メソッド

```
int LoadInitFile(string filename)  
int LoadInitData(string data, int length)  
int LoadInitData(string data)
```

引数

filename	初期化ファイルのパス名
data	初期化データ(XML 形式データ)
length	初期化データのバイト数

戻り値

初期化ファイルを読み取ると0(ゼロ)が戻ります。それ以外の場合は、エラーコードです。

4.4 入力ファイル(PDFまたは画像データ)を開く

PDF 文書を開く際は、そのPDF 文書が印刷する場合の解像度を低解像度に指定されている場合や、印刷そのものが禁止されている場合があります。そのような場合PDF 文書はパスワードなどで暗号化されています。Imager-LP は、パスワードで暗号化されたPDF 文書を復号して画像に変換できます。また、開くモードを「表示」や「印刷」に指定して適切に開くことができます。

PDF Imager-LP は、PDF 文書以外に画像データを入力として開くことができます。入力画像はそのファイルの拡張子やデータの内容によって判別され適切に解釈されます。

4.4.1 PDFファイルの許可フラグを無視して開く

画像に変換するPDFファイルを開きます。

この関数はPDFファイルが暗号化されている場合でも、PDF 文書に指定された許可フラグを無視します。なお、パスワードは、PDFファイルがパスワードで暗号化されている場合のみ使用し、暗号化されていない場合は無視します。

メソッド

```
int OpenDoc(string filename, string ownerPassword, string userPassword)
```

引数

filename	PDF のファイルパス
ownerPassword	所有者パスワード ¹
userPassword	ユーザーパスワード ²

戻り値

成功すると0(ゼロ)が戻り、失敗した場合はエラーコードが戻ります。

4.4.2 PDFファイルを指定モードで開く

画像に変換するPDFファイルを開くモード(「表示」または「印刷」)に従って開きます。

この関数はPDFファイルが暗号化されている場合、PDF 文書に指定された許可フラグに従って開きます。そのため、印刷が許可されないPDFファイルを「印刷」モードで開くと失敗します。また、低解像度印刷指定の場合は失敗しませんので、適切に画像化解像度を指定するため許可フラグを確認する必要があります。(許可フラグは、「4.5.2 PDF文書のパーミッション(許可)フラグ」を参照してください。)

なお、パスワードは、PDFファイルがパスワードで暗号化されている場合のみ使用し、暗号化されていない場合は無視します。

メソッド

```
int OpenDocUsage(string filename, string password, int mode)
```

引数

filename	PDF文書のファイルパス名
password	パスワード オーナーパスワード、ユーザーパスワードのいずれかを指定します。
mode	1(表示)または、2(印刷)を指定します。

戻り値

成功すると0(ゼロ)が戻り、失敗した場合はエラーコードが戻ります。

¹ 所有者パスワードは「権限パスワード」と呼ばれることがあります。

² ユーザーパスワードは「開くパスワード」と呼ばれることがあります。

4.4.3 画像ファイルを開く

画像ファイルを開く場合は、OpenDoc または OpenDocUsage メソッドを利用します。ただし、ファイル名以外の引数は無視されます。また、画像形式はファイルの拡張子およびデータ内容から適切に解釈され内部データ(ピクセルデータ)に変換されます。

読み込まれた画像は、白色不透明な背景画像に描画されます。そのため、画像データにアルファチャンネル(不透明を指定したデータ)が含まれていても出力画像には現れません。

4.5 PDF文書の情報

PDF Imager-LP は、開いたPDF文書の情報を取得できます。

4.5.1 PDF文書のページ数取得

現在開いているPDF文書の総ページ数を取得します。

メソッド

```
int PageCount()
```

引数

ありません

戻り値

成功すると、PDF文書の総ページ数(0(ゼロ)の場合もあります)が戻り、失敗した場合はエラーコードが戻ります。なお、エラーコードは負数です。

4.5.2 PDF文書のパーミッション(許可)フラグ

PDF文書には、その文書を開いたユーザーに変更や印刷の可否を指定するフラグがあります。PDF Imager-LP はこの値を現在開いているPDF文書からパーミッション(許可)フラグ値として取得します。このフラグを評価するには、このPDF文書の暗号化レビジョンも必要です。暗号化レビジョンを取得するのは「4.5.3PDF文書の暗号化レビジョン」を参照してください

メソッド

```
int GetDocumentInfo(DocumentOpt.PERMISSION_FLAG, out int flag)
```

引数

flag パーミッション(許可)フラグ値

戻り値

成功すると、0(ゼロ)が戻り、失敗した場合はエラーコードが戻ります。

パーミッションフラグの詳細は、「PDF Reference」 3.5.2 Standard Security Handler を参照してください。

パーミッションフラグ値は、PDF文書の「Standard Security Handler」ディクショナリ内のPキーに指定された整数値です。

4.5.3 PDF文書の暗号化レビジョン

PDF文書が暗号化されている場合は、印刷の許可などのフラグ（パーミッションフラグ）を確認しなければならない場合があります。以下ではこのフラグの評価に必要な暗号化レビジョンを取得します。（パーミッションフラグの取得は「4.5.2 PDF文書のパーミッションフラグ」を参照）

メソッド

```
int GetDocumentInfo(DocumentOpt.ENCRYPT_REVISION, out int rev)
```

引数

rev	暗号化のレビジョン番号
-----	-------------

戻り値

成功すると、0（ゼロ）が戻り、失敗した場合はエラーコードが戻ります。

暗号化レビジョンの詳細は、「PDF Reference」 3.5.2 Standard Security Handler を参照してください。

暗号化レビジョンは、PDF文書の「Standard Security Handler」ディクショナリ内のRキーに指定された値です。

4.5.4 PDF文書のメタデータ

PDF文書に記載されたメタデータをバイトデータまたは文字列で取得します。メタデータはUTF-8文字コードで記載されています。

メタデータの詳細は、「PDF Reference」 10.2 Metadata を参照してください。

メソッド

```
int GetMetadata(out byte[] metadata)  
int GetMetadataString(out string metadataString)
```

引数

metadata	バイト列のメタデータ
metadataString	Unicode に変換されたメタデータ

戻り値

成功するとバイトサイズまたは文字数が戻り、失敗した場合はエラーコードが戻ります。

4.5.5 PDF文書内のフォント名

PDF文書に記載されたフォント名を取得します。記載されたとおりに取得しますので、フォントを代替する場合の名称として利用できます。(「4.15 代替フォント指定」参照)

メソッド(C#)

```
int SimpleFontListLength()  
int SimpleFontListGetName(int number, out string fontName [,out int flag])  
string[] SimpleFontListGetName()  
FontNameInfo SimpleFontListGetNameC()
```

関数(C/C++)

```
int MlpSimpleFontListGetName(int number, TCHAR **fontName, int *flag)
```

引数

number	フォント名に Imager-LP が検索順に付加した番号
fontName	検索されたフォントの名称
flag	2:フォントのサブセット埋め込まれている、1:フォント全体が埋め込まれている場合、4:フォントが標準14フォントの場合、0:それ以外の場合

戻り値(C#)

SimpleFontListLength および SimpleFontListGetName で引数を指定した場合は失敗すると負数のエラーコードを返します。それ以外は検索されたフォント名の総数を返します。SimpleFontListGetName で引数なしの場合は検索されたすべてのフォント名が格納された配列を返します。SimpleFontListGetNameC 検索されたフォント名と共に埋め込み・非埋め込みなどを示すフラグが格納されたクラスの配列インスタンスを返します。

戻り値(C/C++)

MilSimpleFontListGetName は失敗すると負数のエラーコードを返します。それ以外の場合は検索されたフォント名の総数を返します。

4.5.6 インターラクティブ・フォーム

PDF 文書に記載されたインターラクティブ・フォームの情報を取得します。インターラクティブ・フォームの詳細は「PDF Reference 1.7」 8.6Interactive Form を参照してください。

情報はPDF文書に記載された順に取得しますのでPDFのページに表示された見た目の順番とは違いますのでご注意ください。

フォームの情報が格納されたクラス(構造体)

```
class FontData
{
    int type;                // フォームタイプ
    string name;             // フォームの階層を含めた名称
    int valueLength;         // value配列の長さ
    string[] value;          // 表示文字列と取得値のペア
    string richText;         // テキスト文字列がリッチテキストの場合
    string[] GetValue(int);  // 表示文字列と取得値の1つのペアを取得
}
```

フォームのタイプ

インターラクティブ・フォームにはボタン、テキスト、項目選択がありますがそれぞれの動作に応じて更に詳細に分類できます。PDF Imager-LP はそれらに番号を付けて管理されます。

以下はその番号とフォームです。

- | 番号 | 内容 |
|----|---|
| 1 | プッシュボタン:
利用者がボタンを押下したときに何らかの動作をさせます。
そのため、ボタン自身は値を持ちません。 |
| 2 | チェックボタン:
利用者がマウスなどで操作したときにオンとオフの2つの状態を切り替えます。
オンの場合に定義した値を持ちます。オフの場合はヌル文字もしくは“Off”です。 |
| 3 | ラジオボタン:
1つ以上のボタンのグループで、オンとオフの2つの状態を持ちます。通常はグループ内のいずれかのボタンをオンにすると他のボタンがオフになります。 |
| 4 | テキストボックス(単一行):
単一行の文字列を扱います。 |
| 5 | テキストボックス(複数行):
1行以上の文字列を扱います。 |
| 6 | テキスト(ファイルパス):
ファイルのパスを値として持ちます。 |
| 7 | テキスト(リッチ・テキスト):
リッチ・テキスト文字列を値として保持します。 |
| 8 | パスワード:
パスワードを画面に表示することなく入力するためのフォームです。
パスワードそのものは値として保持しません。 |
| 9 | リストボックス(単一選択):
選択できる文字列をスクロール可能なボックスに格納したフォームで、ただ一つの項目を選択できます。 |
| 10 | リストボックス(複数選択):
選択できる文字列をスクロール可能なボックスに格納したフォームで、複数の項目を選択できます。 |
| 11 | コンボボックス:
ドロップダウンリストで選択項目を表示するフォームです。
選択項目だけでなく、利用者が値を入力することもできます。 |
| 20 | 電子署名用フォーム:電子署名の情報が格納されたフォーム(未実装) |

フォームの名前

各インタラクティブ・フォームを識別する階層を含めた名前です。

フォームの値

各インタラクティブ・フォームは選択するための取得値と表示値の2つをペアとして値を持ちます。

複数選択可能なフォームは、このペアを選択数だけ持ちます。

PDF Imager-LPはこのペアの総数を `valueLength` として保持し、抽出値を持たない場合は表示値を抽出値として保持します。また、利用者が選択値と違う値を入力した場合はそれを抽出文字列とし、表示値をヌル文字列とします。フォームタイプがテキストの場合も抽出文字列のみが格納されます。

複数の選択値

複数の選択された値を持つ場合は、`GetValue()` メソッドにインデックス(最初のペアはインデックス0です)を指定することで目的のペア(抽出値:要素0、表示値:要素1)を取得できます。

リッチ・テキスト

フォームがテキストでリッチ・テキストと指定された場合は、抽出値(リッチ・テキスト形式)を `richText` に格納します。

`value` にはPDF文書に記載された値が格納されます。

PDF 文書に記載されたインタラクティブフォームを以下のメソッドで取得します。まず `InteractiveFormFirstData` を使って最初に記載されたフォームのデータを取得します。次に、`InteractiveFormNextData` を使って残りフォームのデータを取得します。`FormData` の `type` が `Undefined(0)` になるまで繰り返しデータを取得することですべてのデータを取得できます。

メソッド

```
FontData InteractiveFormFirstData()  
FontData InteractiveFormNextData()
```

引数

ありません

戻り値

`FormData` クラス(または `MlpFormData` 構造体)が戻ります。

インタラクティブフォームが見つかった場合は、`FormData` の `type` 要素に1以上の値が戻ります。

この値が0(ゼロ)の場合はフォームがそれ以上ありません。

値が負数の場合はエラー(コード)です。

4.6 画像ファイルに変換する関数群

JPEG形式の画像およびPNG形式の画像の場合は、PDF文書の1つのページを1つの画像に変換します。TIFF形式の画像の場合は、PDF文書の1つのページを1つの画像に変換することや、PDF文書の複数のページを1つの画像ファイルに変換することができます。以下は、画像変換用の関数名および変換できる画像形式です。

関数名	変換できる画像形式
CreatePict	PNG、JPEG、TIFF、BMP
PageToPict	PNG、JPEG、TIFF、BMP
ConvertToTiff	TIFF
CreateTiffRange	TIFF
CreateTiffMulti	TIFF
RangeToTiff	TIFF
MultiPageToTiff	TIFF
CreateMem	ピクセルデータ

4.6.1 指定された1ページを画像に変換(解像度と画像品質を指定しない)

PDF文書の指定されたページを画像に変換します。

解像度と画像品質は、規定値または画像の変換に先立って指定された値が使用されます。画像の解像度や品質を指定する場合は、SetPictureResolution、SetPictureQuality、SetPicture関数などを利用して下さい。

ページ番号の指定では、最初のページを1とします。なお、ページ番号0はPDF文書の先頭ページ、負数のページ番号は最終ページとみなされます。

変換する画像ファイルの拡張子で画像の形式が自動で判断されます。

メソッド

```
int CreatePict(int pageNumber, string outputPath)
```

引数

pageNumber	画像に変換するPDF文書のページ番号(先頭のページ番号は1です。)
outputFilePath	画像のファイルパス
	画像形式はファイルの拡張子によって以下のように自動で選択されます。
	拡張子が“png”の場合、PNG形式の画像
	拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像
	拡張子が“tiff”または“tif”の場合、TIFF形式の画像
	拡張子が“bmp”の場合、BMP形式の画像

戻り値

成功すると0(ゼロ)が戻り、それ以外の場合はエラーコードが戻ります。

4.6.2 指定された1ページを画像に変換(解像度と画像品質を指定する)

PDF文書の指定されたページを画像に変換します。

画像に変換する際に、指定された解像度および画像品質を使用します。ページ番号は、先頭のページを1とします。

ページ番号に負数を指定しますと、PDF文書の最終ページが画像に変換されます。

変換する画像ファイルの拡張子で画像の形式が判断されます。

メソッド

```
int PageToPict(int pageNumber, int dpi, int quality, string outputPath)
```

引数

pageNumber	画像に変換するPDF文書のページ番号(先頭のページ番号は、1です。)
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstants.MIN_RESOLUTION</code> 以上かつ <code>MlpConstants.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstants.MIN_RESOLUTION</code> または <code>MlpConstants.MAX_RESOLUTION</code> に正規化されます。
quality	JPEG 圧縮の画像品質を <code>MlpConstants.MIN_QUALITY</code> 以上かつ <code>MlpConstants.MAX_QUALITY</code> までの数値で指定します。範囲外の値を指定すると <code>MlpConstants.MIN_QUALITY</code> または <code>MlpConstants.MAX_QUALITY</code> に正規化されます。
outputFilePath	PNG形式画像を指定した場合は無視されます。 画像のファイルパス 画像形式はファイルの拡張子によって以下のように自動で選択されます。 拡張子が“png”の場合、PNG形式の画像 拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像 拡張子が“tiff”または“tif”の場合、TIFF形式の画像 拡張子が“bmp”の場合、BMP形式の画像

戻り値

成功すると0(ゼロ)が戻り、それ以外の場合はエラーコードが戻ります。

4.6.3 すべてのページを TIFF 形式画像に変換

PDF文書のすべてのページを1つのTIFF形式画像ファイルに変換します。

メソッド

```
int ConvertToTiff (string outputPath)
```

引数

outputFilePath	画像のファイルパス TIFF形式の画像が作成されます。拡張子は「.tiff」を指定します。
----------------	--

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.6.4 ページ範囲を指定して TIFF 形式画像に変換

指定された範囲のPDFページをひとつのTIFF画像に変換します。(指定された範囲内のページをすべて含む1つのTIFF画像に変換されます)。ただし、PDF文書にないページを指定すると失敗します。

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

ページ番号0を指定すると最初のページ(ページ番号1)、ページ番号が負数の場合は最終ページとみなされます。したがって、開始ページ=1 終了ページ=-1 と指定すると、最初のページから最後のページまですべてのページをTIFF画像に変換します。

ページは逆順(大きいページ番号から小さいページ番号の順)でも指定できます。

メソッド

```
int CreateTiffRange(  
    int startPageNumber, int endPageNumber, string outputPath  
)
```

引数

startPageNumber 画像に変換する最初のPDF文書のページ番号(先頭のページ番号は、1です。)
endPageNumber 画像に変換する最後のPDF文書のページ番号
outputFilePath 画像のファイルパス
TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.6.5 ページのリストを指定して TIFF 形式画像に変換

リストで指定されたPDF文書のページをひとつのTIFF画像に変換します。

画像に変換するページを区切り文字(スペース、タブまたはコンマ)で区切ってページを指定すると、指定した順に画像に変換された複数ページで構成されたTIFF画像が生成されます。このとき、PDF文書に無いページを指定するとその指定は無視されます。成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

ページの指定の詳細は、「9.0 文字列でのページ指定方法」を参照してください。

メソッド

```
int CreateTiffMulti(string pageList, string outputPath)
```

引数

pageList 画像に変換するページのリスト。画像に変換するページを区切り文字で区切って1つ以上のページを指定します。PDF文書にないページは無視されます。
ページ番号0は最初のページ、ページ番号-1は最後のページに変換されます。
詳細は、「9.0 文字列でのページ指定方法」を参照してください。
outputFilePath 画像のファイルパス
TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.6.6 ページ範囲を指定して TIFF 形式画像に変換(解像度を指定する)

指定された範囲のPDFページをひとつのTIFF画像に変換します。指定された範囲内のページをすべて含む1つのTIFF画像に変換されます。ただし、PDF文書にないページを指定すると失敗します。

ページ番号0を指定すると最初のページ(ページ番号1)、ページ番号が負数の場合は最終ページとみなされます。したがって、開始ページ=1 終了ページ=-1 と指定すると、最初のページから最後のページまですべてのページをTIFF画像に変換します。

ページは逆順(大きいページから小さいページ番号の順)でも指定できます。

メソッド

```
int RangeToTiff(  
    int startPageNumber, int endPageNumber, int dpi, string outputPath  
)
```

引数

startPageNumber	画像に変換する最初のPDF文書のページ番号(最初のページは、1です。)
endPageNumber	画像に変換する最後のPDF文書のページ番号(最後のページを、-1 と指定できます。)
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstants.MIN_RESOLUTION</code> 以上かつ <code>MlpConstants.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstants.MIN_RESOLUTION</code> または <code>MlpConstants.MAX_RESOLUTION</code> に正規化されます。
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.6.7 ページのリストを指定して TIFF 形式画像に変換(解像度を指定する)

リストで指定されたPDFページをひとつのTIFF画像に変換します。

画像に変換するページを区切り文字(スペース、タブまたはコンマ)で区切ってページを指定すると、指定した順に画像に変換された複数ページで構成されたTIFF画像が生成されます。このとき、PDF文書に無いページを指定するとその指定は無視されます。

ページの指定の詳細は、「9.0 文字列でのページ指定方法」を参照してください。

メソッド

```
int MultiPageToTiff(string pageList, int dpi, string outputPath)
```

引数

pageList	画像に変換するページのリスト。画像に変換するページを区切り文字で区切って1つ以上のページを指定します。PDF文書にないページは無視されます。 ページ番号 0 は最初のページ、ページ番号-1 は最後のページに変換されます。 詳細は、「9.0 文字列でのページ指定方法」を参照してください。
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstants.MIN_RESOLUTION</code> 以上かつ <code>MlpConstants.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstants.MIN_RESOLUTION</code> または <code>MlpConstants.MAX_RESOLUTION</code> に正規化されます。
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.7 画像(ピクセル)データに変換する関数群

Imager-LPは画像形式への変換ばかりではなく、ピクセルデータへも変換します。

4.7.1 指定されたページをピクセルデータに変換

指定されたページをピクセルデータに変換し、そのピクセルデータを取得します。

変換されたピクセルデータにアルファチャネルは含みません。各ピクセルは RGB (Red, Green, Blue; カラーの場合) 値がこの順番でまたは、グレースケール(白黒の場合) 値が格納されます。

メソッド

```
int CreatePictBufPage(  
    int pageNumber, out byte[] pixels, out int width, out int height,  
    out int nOfColors  
)
```

引数

pageNumber	ピクセルデータに変換するページの番号。
pixels	ピクセルデータ(一次元配列)
width	画像に変換されたピクセルの幅
height	画像に変換されたピクセルの高さ
nOfColors	画像に変換されたピクセルの1画素あたりの色数

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.7.2 連続する複数ページをピクセル画像に変換

複数のページを連続して画像に変換します。

ページは、開始のページ番号と終了ページの番号を指定します。変換されたピクセルデータは、ライブラリ内部に格納されます。ピクセルデータは、インデックス番号(指定されたページ順に0から割り振られます)とページ番号で管理されます。格納されたピクセルデータは、GetPictFromMemIndex メソッドまたは GetPictFromMemPage メソッドを使って取得します。

CreatePictBufPageRange メソッドは、既定でページ画像への変換を並列に処理します。この並列処理は抑制することもできます。

メソッド

```
int CreatePictBufPageRange(int firstPageNumber, int lastPageNumber)
```

引数

firstPageNumber	ピクセルデータに変換を開始するページの番号。
lastPageNumber	ピクセルデータに変換を終了するページの番号。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.7.3 不連続な複数ページをピクセル画像に変換

連続していない複数のページを画像に変換します。

ページは、配列で指定します。変換されたピクセルデータは、ライブラリ内部に格納されます。ピクセルデータは、インデックス番号（指定されたページ順に0から割り振られます）とページ番号で管理されます。格納されたピクセルデータは、GetPictFromMemIndex メソッドまたは GetPictFromMemPage メソッドを使って取得します。

CreatePictBufPageArray メソッドは、ページ画像への変換を並列に処理します。この並列処理は抑制することもできます。

メソッド

```
int CreatePictBufPageArray(int[] pageNumberArray, int length)
```

引数

pageNumberArray ピクセルデータに変換するページの番号が格納された配列

length 配列に格納されたページの総数

戻り値

成功すると、0（ゼロ）が戻り、失敗するとエラーコードが戻ります。

4.7.4 指定されたページをライブラリ内部データに変換

指定されたページをライブラリ内部の画像データに変換します。

生成された画像は GetPictFromMem メソッドで取り出したり、MemPictToPanel メソッドや MemPictToPanelBBox メソッドでパネルに描画したりできます。

メソッド

```
int CreatePictMem(int pageNumber)
```

引数

pageNumber ピクセルデータに変換するページの番号。

戻り値

成功すると、0（ゼロ）が戻り、失敗するとエラーコードが戻ります。

4.7.5 ライブラリ内部の画像データを取得

ライブラリ内部に生成された画像データを取得します。この関数で取得できる画像データは、単一のページを画像に変換したデータです。複数のページを画像に変換した場合 (CreatePictBufPageRange メソッドを使った場合) の画像を取り出すには、GetPictFromMemIndex メソッドまたは GetPictMemFromPage メソッドを使用します。

変換されたピクセルデータにアルファチャネルは含みません。各ピクセルは RGB (Red, Green, Blue; カラーの場合) 値がこの順番でまたは、グレースケール (白黒の場合) 値が格納されます。

メソッド

```
int GetPictFromMem(  
    out byte[] pixels, out int width, out int height, out int nOfColors  
)
```

引数

pixels	ピクセルデータ (一次元配列) C/C++ 開発環境で利用する場合、このアドレスは一時的ですので後で使うことはできません。
width	画像に変換されたピクセルの幅
height	画像に変換されたピクセルの高さ
nOfColors	画像に変換されたピクセルの1画素あたりの色数

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.7.6 ライブラリ内部の画像データをインデックスで取得

CreatePictBufPageRange メソッドで作成されたピクセルデータをインデックスで取得します。

メソッド

```
int GetPictFromMemIndex(  
    int index, out byte[] pixels, out int width, out int height,  
    out int nOfColors, out int pageNumber  
)
```

引数

index	PDF 文書のページを連続して画像に変換した場合にそれぞれのページに自動で割り振られた 0 から始まる番号
pixels	ピクセルデータ (一次元配列) C/C++ 開発環境で利用する場合、このアドレスは一時的ですので後で使うことはできません。
width	画像に変換されたピクセルの幅
height	画像に変換されたピクセルの高さ
nOfColors	画像に変換されたピクセルの1画素あたりの色数
pageNumber	PDF 文書のページ番号

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.7.7 ライブラリ内部の画像データをページ番号で取得

CreatePictBufPageRange メソッドで作成されたピクセルデータをPDF文書のページ番号で取得します。

メソッド

```
int GetPictFromMemPage(  
    int pageNumber, out byte[] pixels, out int width, out int height,  
    out int nOfColors,  
)
```

引数

pageNumber	PDF文書のページ番号
pixels	ピクセルデータ(一次元配列) C/C++開発環境で利用する場合、このアドレスは一時的ですので後で使うことはできません。
width	画像に変換されたピクセルの幅
height	画像に変換されたピクセルの高さ
nOfColors	画像に変換されたピクセルの1画素あたりの色数

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.7.8 ライブラリ内部の画像データを削除

ライブラリ内部の画像データを削除し、そのメモリー領域を開放します。

この画像データ領域は、現在のPDF文書を閉じる際またはライブラリの使用終了で自動的に削除されるためメモリー容量等に問題がない場合は開放しなくともかまいません。

メソッド

```
int FreePictMem()
```

引数

ありません

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.8 出力画像の色空間を指定する

PDF文書がカラーで構成されている場合でも、グレースケール、白黒ディザ、白黒(2階調)画像に変換します。色空間の指定を省略すると、RGBカラーの画像が生成されます。

4.8.1 RGB カラー画像を指定

PDF文書をRGBカラーの画像に変換するよう設定します。

PDF文書がRGBカラーでない場合でも、画像データはRGBカラー(1画素あたり24ビットで構成された画像データ)になります。

RGBカラー画像指定は、既定値です。

メソッド

```
int SetPictureRGB()  
int SetPicture(PictureOpt.MLP_PICTURE_RGB)
```

引数

SetPictureRGB には、ありません。

SetPicture では、PictureOpt.PICTURE_RGB を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.8.2 CMYK カラー画像を指定

PDF文書をCMYKカラーの画像に変換するよう設定します。

メソッド

```
int SetPictureCMYK
```

引数

ありません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.8.3 グレースケール画像を指定

PDF文書をグレースケールの画像に変換するよう設定します。

メソッド

```
int SetPictureGray()  
int SetPicture(PictureOpt.PICTURE_GRAY)
```

引数

SetPictureGray には、ありません。

SetPicture では、PictureOpt.PICTURE_GRAY を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.8.4 白黒ディザ画像を指定

PDF文書を白黒ディザ(Floyd-Steinberg ディザリング)の画像に変換するよう設定します。

メソッド

```
int SetPictureDither()  
int SetPicture(PictureOpt.PICTURE_DITHER)
```

引数

SetPictureDither には、ありません。

SetPicture では、PictureOpt.PICTURE_DITHER を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.8.5 白黒2階調画像を指定

PDF文書を白黒2階調の画像に変換するよう設定します。

メソッド

```
int SetPictureBW()  
int SetPicture(PictureOpt.PICTURE_BW)
```

引数

SetPictureBW には、ありません。

SetPicture では、PictureOpt.PICTURE_BW を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.9 カラープロファイル

Imager-LPはRGBカラー画像に変換する場合に、PDF文書に指定されたCMYK色空間をRGB色空間に変換します。既定では、この変換を計算式に基づいて実施しますが、カラープロファイルを用いた変換を指定することができます。

4.9.1 カラープロファイルの指定

カラープロファイルを指定します。この指定をせずにカラープロファイルによる変換を指定しますと、既定のカラープロファイルが利用されます。UnuseColorProfile メソッドによってカラープロファイルの利用を解除できます。

カラープロファイルを使って色空間を変更する場合は SetColorConversion メソッドで変換手順を変更してください。

メソッド

```
int SetRGBProfile(string rgbpfn) /*RGBカラープロファイル指定*/  
int SetCMYKProfile(string cmykpfn) /*CMYKカラープロファイル指定*/  
int UnuseColorProfile() /*カラープロファイルの使用を解除*/
```

引数

rgbpfn	RGBカラープロファイルのファイル名
cmykpfn	CMYKカラープロファイルのファイル名

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

注意

現在利用可能なカラープロファイルは以下のとおりです。

RSWOP.icm (Agfa:Swop Standard) :既定のCMYKカラープロファイル
JapanColor2001Coated.icc (Japan Color 2001 Coated)
JapanColor2001Uncoated.icc (Japan Color 2001 Uncoated)
JapanColor2002Newspaper.icc (Japan Color 2002 Newspaper)
JapanWebCoated.icc (Japan Web Coated (Ad))
USWebCoatedSWOP.icc (U.S. Web Coated (SWOP) v2)
USWebUncoated.icc (U.S. Web Uncoated v2)
sRGB Color Space Profile.icm (sRGB IEC61966-2.1) :既定のRGBカラープロファイル
AdobeRGB1998.icc (Adobe RGB (1998))
AppleRGB.icc (Apple RGB)

4.9.2 色空間変換手順指定

CMYK色空間からRGB色空間への変換を指定します。

メソッド

```
int SetColorConversion(int direction, int flag)
```

引数

direction	CMYK色空間からRGB色空間へ変換するように以下を指定します。 <code>MlpConstants.CONVERT_CMYK_TO_RGB</code>
flag	色空間の変換手順を指定します。以下のいずれかを指定します。 <code>MlpConstants.CONVERT_BY_COLOR_PROFILE</code> カラープロファイルを使った変換を指定 <code>MlpConstants.CONVERT_FAST</code> 計算による変換を指定 (既定の手順) <code>MlpConstants.CONVERT_MOST_FAST</code> 簡易計算による変換を指定

4.10 画像のサイズ

PDF 文書では、1 ポイントを 1/72 インチとしています。PDF Imager-LP は、この 1 ポイントを指定された画像解像度に変換して画像を生成します。そのため、大きな値の画像解像度を指定すると大きなピクセルサイズの画像が生成され、小さな画像解像度では小さなピクセルサイズの画像が生成されます。

キャンバスとは、画像を配置する領域です。生成された画像は、このキャンバスに貼り付けられます。キャンバスの領域が生成された画像よりも小さい場合は画像の一部が切り取られ、逆に大きな領域を指定すると余白の付加された画像になります。

生成された画像は、画像の解像度から算出された画像の論理サイズを持っています。論理サイズを指定のサイズに設定できます、がこの指定によって画像のピクセルサイズは変わりません。論理サイズの既定値は、PDF 文書に記載されたページの大きさです。

4.10.1 ページの大きさ(境界)指定

PDF 文書では、ページごとにその大きさを持っています。さらに、このページの大きさ(詳細は、『PDF Reference』を参照してください。)を表すために5種類の大きさ(MediaBox、CropBox、BleedBox、TrimBox、ArtBox)が用意されています。PDF Imager-LP は、ページの大きさを決定する際の境界を指定できます。

メソッド

```
int SetPicture(PictureOpt opt)
```

引数

opt	利用する大きさの種類を指定します。
PICTURE_USE_MEDIABOX	MediaBox で指定された大きさを使います。(既定値)
PICTURE_USE_CROPBOX	CropBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、MediaBox を使います。
PICTURE_USE_BLEEDBOX	BleedBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。
PICTURE_USE_TRIMBOX	TrimBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。
PICTURE_USE_ARTBOX	ArtBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.10.2 画像解像度の設定

PDF文書を画像に変換する際の 1 ポイントを指定のピクセル数で構成するよう指定します。このときに指定する数値を画像解像度(または、解像度)といいます。既定の解像度は、150DPI(Dot Per Inch)です。

解像度は、X方向、Y方向で同じ値が設定されます。設定する値が大きい場合は、処理に必要なメモリが多くなり、処理時間も増大します。

既定では 150DPI で画像が生成されます。

メソッド

```
int SetPictureResolution(int dpi)
int SetPicture(PictureOpt.RESOLUTION_DPI, int dpi)
```

引数

dpi	解像度を DPI(dot per inch) 単位で指定します。 解 像 度 は 、 <code>MlpConstants.MIN_RESOLUTION</code> 以 上 か つ <code>MlpConstants.MAX_RESOLUTION</code> 以下を指定します。範囲外の値を指定 す る と 、 <code>MlpConstants.MIN_RESOLUTION</code> ま た は <code>MlpConstants.MAX_RESOLUTION</code> に正規化されます。
-----	--

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.10.3 画像のピクセルサイズを取得

指定された解像度で生成される画像のピクセルサイズを取得します。

メソッド

```
int GetPicturePixel(int pageNum, out int x, out int y)
```

引数

pageNum	サイズを取得するページの番号
x	画像の幅(ピクセル単位)
y	画像の高さ(ピクセル単位)

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

なお、PDF文書では、ページごとにその物理的な大きさが”MediaBox”(PDF Reference 3.6.2 Page Tree 参照)として記載されています。PDF文書での解像度 72DPI を指定して所得する画像のピクセルサイズは、この”Media Box”のサイズです(既定の場合)。ページサイズの種類は変更できます。「4.10.1 ページの大きさ」を参照してください。

4.10.4 画像幅のピクセルサイズを取得

指定された解像度で生成される画像幅のピクセルサイズを取得します。

メソッド

```
int GetPicturePixelX(int pageNum)
```

引数

pageNum	サイズを取得するページの番号
---------	----------------

戻り値

成功すると画像の幅がピクセル数で戻り、失敗するとエラーコード(負数)が戻ります。

なお、PDF文書では、ページごとにその物理的な大きさが”MediaBox”(PDF Reference 3.6.2 Page Tree 参照)として記載されています。PDF文書での解像度 72DPI を指定して所得する画像のピクセルサイズは、この”Media Box”のサイズです(既定の場合)。ページサイズの種類は変更できます。「4.10.1 ページの大きさ」を参照してください。

4.10.5 画像高さのピクセルサイズを取得

指定された解像度で生成される画像高さのピクセルサイズを取得します。

メソッド

```
int GetPicturePixelY(int pageNum)
```

引数

pageNum	サイズを取得するページの番号
---------	----------------

戻り値

成功すると画像の高さがピクセル数で戻り、失敗するとエラーコード(負数)が戻ります。

なお、PDF文書では、ページごとにその物理的な大きさが”MediaBox”(PDF Reference 3.6.2 Page Tree 参照)として記載されています。PDF文書での解像度 72DPI を指定して所得する画像のピクセルサイズは、この”Media Box”のサイズです(既定の場合)。ページサイズの種類は変更できます。「4.10.1 ページの大きさ」を参照してください。

4.10.6 画像のピクセルサイズ指定

生成される画像の大きさをピクセル単位で指定します。ただし、指定できるのは幅(X方向)または、高さ(Y方向)のみです。

メソッド

```
int SetPicturePixelX(int x)
int SetPicturePixelY(int y)
```

引数

x	画像の幅をピクセル単位で指定します。 0(ゼロ)を指定すると、既定の大きさになります。 高さ(Y方向)は、自動で計算されます。
y	画像の高さをピクセル単位で指定します。 0(ゼロ)を指定すると、既定の大きさになります。 幅(X方向)は、自動で計算されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.10.7 キャンバスサイズの設定

PDF文書から変換された画像を配置する仮想の領域(キャンバス)のサイズを指定します。

キャンバスサイズを変更しても画像の解像度は変化しません。そのため、変換された画像よりも小さな領域を指定すると画像が切り取られ、大きな領域を指定すると画像に余白が付加されます。

生成された画像は、キャンバスの左上に配置され、余白は白色で塗りつぶされます。

指定を省略すると、生成される画像のピクセルサイズとキャンバスのサイズは同じになります。

メソッド

```
int SetCanvasSize(int x, int y)
```

引数

x	キャンバスの幅をピクセル単位で指定します。 0(ゼロ)を指定すると、変換された画像の幅と同じになります。
y	キャンバスの高さをピクセル単位で指定します。 0(ゼロ)を指定すると、変換された画像の高さと同じになります。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.10.8 キャンバスサイズ変更の原点

PDF文書から変換された画像のキャンバスサイズを変更(「4.10.7 キャンバスサイズの変更」参照)する際の原点位置を指定します。

原点を指定することで切り取または余白が追加される画像の辺を限定できます。

指定を省略すると、原点は画像の中心になります。そのため、画像の四辺で切り取りまたは余白の追加が実施されます。なお、原点が中央の場合に上下左右の辺の切り取りピクセル数または追加される余白のピクセル数は必ずしも同じではありません。

メソッド

```
int SetCanvasOrigin(int org)
```

引数

org	キャンバスサイズを変更する際の原点を指定します。 0(ゼロまたは <code>CanvasOrigin.CORIGIN_CENTER</code>): 画像の中心を原点とします。 画像の四辺が変更対象になります。 1(または <code>CanvasOrigin.CORIGIN_LEFT_TOP</code>): 画像の左上を原点とします。 画像の右辺、下辺のみが変更対象になります。 2(または <code>CanvasOrigin.CORIGIN_LEFT_BOTTOM</code>): 画像の左下を原点とします。 画像の右辺、上辺のみが変更対象になります。 3(または <code>CanvasOrigin.CORIGIN_RIGHT_TOP</code>): 画像の右上を原点とします。 画像の左辺、下辺のみが変更対象になります。 4(または <code>CanvasOrigin.CORIGIN_RIGHT_BOTTOM</code>): 画像の右下を原点とします。 画像の左辺、上辺のみが変更対象になります。
-----	---

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.10.9 画像の論理サイズ指定

PDF Imager-LP で生成する画像データ (TIFF 形式および JPEG 形式画像) は、そのピクセルサイズとは別に、論理的な大きさを持っています。論理的な大きさは、画像データのプロパティに含まれる解像度と画像のピクセルサイズから算出されます。PDF Imager-LP では、この算出された画像の大きさを画像の論理サイズといいます。

次の関数では、この論理サイズを指定します。ただし、記載される解像度の丸め誤差によって、生成された画像の論理サイズにも誤差が生じます。そのため、画像の形式にもよりますが、画像への変換時に指定した論理サイズと、生成された画像の論理サイズに差異が生じます。

メソッド

```
int SetPictureSize(int x, int y, PictureUnit unit)
```

引数

x	論理サイズの幅を指定します。
y	論理サイズの高さを指定します。
unit	指定した論理サイズの単位を指定します。 以下の値を指定します。

<code>PictureUnit.PICTURE_UNIT_PERCENT</code>	値をパーセントで指定
<code>PictureUnit.PICTURE_UNIT_INCH</code>	値をインチで指定
<code>PictureUnit.PICTURE_UNIT_MM</code>	値をミリメートルで指定

論理サイズは、幅もしくは高さのいずれかのみを設定できます。両方を指定した場合は、幅が優先されます。高さを指定する場合は、幅を0(ゼロ)にします。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

注意

指定のサイズから算出される論理的な画像解像度が小さすぎる場合に正しい画像データが生成されない場合があります。

4.10.10 ページの境界ボックスのサイズを取得

PDF 文書では、そのページのサイズを表すのに、Media box、Crop box、Bleed box、Trim box および Art box を使用します。なお、Media box は必須で指定されますが、他は省略される場合があります。

次の関数では、この境界ボックスのサイズを所得します。

GetPageBoundary メソッドは各境界ボックスの値を既定値で補完した値を取得しますが、GetPageRawBoundary メソッドは PDF 文書から読み取った値をそのまま取得します。

メソッド

```
int GetPageBoundary(  
    int pageNum, BoundaryBox kind, out float userUnit, out float left,  
    out float bottom, out float right, out float top  
)  
int GetPageRawBoundary(  
    int pageNum, BoundaryBox kind, out float userUnit, out float left,  
    out float bottom, out float right, out float top  
)
```

引数

pageNum	値を取得するページの番号を指定します。
kind	取得する境界ボックスの種類を指定します。 以下の値を指定します。 BoundaryBox.MEDIA_BOX Media boxのサイズを取得するよう指定 BoundaryBox.CROP_BOX Crop box のサイズを取得するよう指定 BoundaryBox.BLEED_BOX Bleed box のサイズを取得するよう指定 BoundaryBox.TRIM_BOX Trim boxのサイズを取得するよう指定 BoundaryBox.ART_BOX Art box のサイズを取得するよう指定
userUnit	UserUnit を取得します。(既定値は 1.0)
left	左辺のx座標位置を取得
bottom	下辺のy座標位置を取得
right	右辺のx座標位置を取得
top	上辺のy座標位置を取得

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

注意:

各座標位置は userUnit 倍されることなく取得します。そのため、この値が宣言されたPDF文書を取り扱う場合にはご注意ください。

4.10.11 ページの回転角を取得

PDF 文書では、ページが回転されている場合があります。
以下のメソッドで、その回転角を取得します。
なお、回転角は0°、90°、180°、270°のいずれかです。

メソッド

```
int GetPageRotation(int pageNum, out int rotate)
```

引数

pageNum	値を取得するページの番号を指定します。
rotate	回転の角度を取得します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.11 TIFF 形式画像の圧縮指定

TIFF 画像を生成する場合の圧縮方法を指定します。

4.11.1 圧縮しない

TIFF形式画像を生成する場合に圧縮をしません。

メソッド

```
int SetTiffCompress(TiffCompressMode.TIFF_COMPRESS_NONE)  
int SetPicture(PictureOpt.COMPRESS_NONE)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.11.2 JPEG 圧縮指定

TIFF形式画像を生成する場合にJPEG圧縮をします。

2番目の指定形式では、JPEG 圧縮の品質を同時に指定します。ここで指定する品質は、画像形式にJPEGを選択した場合にも有効になります。既定の解像度は 75%です。

メソッド

```
int SetTiffCompress (PictureOpt.COMPRESS_JPEG)  
int SetPicture(PictureOpt.COMPRESS_JPEG, int quality)
```

引数

quality	JPEG 圧縮画像の品質を <code>MlpConstants.MIN_QUALITY</code> 以上かつ <code>MlpConstants.MAX_QUALITY</code> 以下で指定します。 範囲外の値を指定した場合は、 <code>MlpConstants.MIN_QUALITY</code> または <code>MlpConstants.MAX_QUALITY</code> に正規化されます。
---------	--

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.11.3 Deflate 圧縮指定

TIFF形式画像を生成する場合にDeflate圧縮をします。

メソッド

```
int SetTiffCompress(TiffCompressMode.TIFF_COMPRESS_DEFLATE)
int SetPicture(PictureOpt.COMPRESS_DEFLATE)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.11.4 LZW 圧縮指定

TIFF形式画像を生成する場合にLZW圧縮をします。

メソッド

```
int SetTiffCompress(TiffCompressMode.TIFF_COMPRESS_LZW)
int SetPicture(PictureOpt.COMPRESS_LZW)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.12 JPEG 圧縮の品質指定

JPEG圧縮が指定された場合の画像品質を指定します。画像データの圧縮が指定されない場合やJPEG圧縮以外の圧縮が指定された場合は、この指定は無視されます。

メソッド

```
int SetPictureQuality(int quality)
int SetPicture(PictureOpt.JPEG_QUALITY, int quality)
```

引数

quality	JPEG圧縮の際の品質を <code>MlpConstants.MIN_QUALITY</code> 以上かつ <code>MlpConstants.MAX_QUALITY</code> 以下の値で指定します。範囲外の値を指定すると、 <code>MlpConstants.MIN_QUALITY</code> または <code>MlpConstants.MAX_QUALITY</code> に正規化されます。
---------	---

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.13 BMP画像の形式を指定する

BMP画像を生成する際に Windows 形式または OS/2形式を指定できます。

メソッド

```
int SetPicture(PictureOpt.BITMAP_IMG_FORMAT, int format)
```

引数

format	0(Windows 形式)または、1(OS/2 形式)を指定します。
--------	------------------------------------

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.14 変換の詳細を指定する

4.14.1 塗りつぶしパスのアンチエイリアスを抑制する

PDF Imager-LP は、パスや文字描画の際にその境界を滑らかにするため、アンチエイリアス処理を施しています。そのため、小さなパスを組み合わせると大きなパス画像になる場合などで、その下地となる画像や文字が透けてしまう場合があります。このような場合は、塗りつぶしパスのアンチエイリアスを抑制して下地が透けて見えるのを防ぐことができます。

なお既定では、塗りつぶしパスのアンチエイリアスは実施します。

メソッド

```
int SetFillPathAADisable()  
int SetPicture(PictureOpt.FILL_PATH_AA, 0)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

プロパティ (set)

```
bool FillPathAA
```

4.14.2 塗りつぶしパスのアンチエイリアスを実施する

塗りつぶしパスのアンチエイリアスを抑制すると、それ以降の変換で抑制されますので、以下でその抑制を中止します。

メソッド

```
int SetFillPathAAEnable();  
int SetPicture(PictureOpt.FILL_PATH_AA, 1);
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

プロパティ (set)

```
bool FillPathAA
```


4.14.3 文字コード0(ゼロ)の表示

PDF Imager-LP は、表示文字のコードが0(ゼロ)の場合には、未定義グリフ³の代わりにスペースが表示されます。

この動作を以下の手順で変更およびその設定状態を取得できます。

メソッド

```
int SetCid0GlyphMode(Cid0GlyphMode mode)           /*設定*/  
Cid0GlyphMode GetCid0GlyphMode()                   /*取得*/  
int SetPicture(PictureOpt.MLP_CID_0_GLYPH, int mode) /*設定*/
```

引数

mode	以下のグリフ表示モード(Cid0GlyphMode)を指定します。
SHOW_NOTDEF_GLYPH	未定義グリフを表示
REPLACE_TO_BLANK_GLYPH	空白(Space)グリフを表示(既定)
IGNORE_CID_0_GLYPH	CID=0 の文字表示を無視します。

戻り値

設定の場合は、成功すると0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

取得の場合は、成功すると設定されている値が戻り、失敗するとエラーコードが戻ります。

プロパティ (set,get)

```
Cid0GlyphMode Cid0GlyphMode
```

4.14.4 極細い線の描画

PDF Imager-LP はPDF文書を指定された解像度で画像に変換するため、解像度に比較して極端に細い線を期待したとおりに変換できないことがあります。このような場合に、その細い線をより太い線で代替描画することを指定します。

以下の関数で、描画する最小の太さ(線幅)を指定します。これにより、指定された線幅より小さな線幅は最小線幅に代替して描画されます。

メソッド

```
int SetPicture(PictureOpt.STROKEPATH_MINIMUM_LINE_WIDTH, float width)
```

引数

width	最小の線幅をポイント(1 ポイント=1/72 インチ)単位で指定します。 0(ゼロ)または負数を指定すると、最小線幅の指定がなくなります。
-------	--

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

ご注意ください。

この関数で変更できる線は、PDFコマンドのパスストロークによって描画される線です。

パスストロークの詳細は、「PDF Reference」を参照してください。

³ フォントなどの文字コードに対するグリフ(字形)が未定義の場合に表示されるグリフ

4.14.5 PDFのバージョン指定

PDF Imager-LP はPDF1.4 からPDF1.7 までおよびPDF2.0 (一部) 規格の文書を画像に変換できます。しかし、これ以外の規格のPDF 文書であっても可能な限り画像に変換しようとしします。規格外のPDF 文書を変換しない設定にできます。

これを指定すると、PDF 文書のオープンは成功しますが許容されないバージョンのPDFの画像化に失敗します。PDF 文書のオープンは成功しますので、その文書のバージョンを取得できますので、画像化の失敗を回避できます。

メソッド

```
int SetPicture(PictureOpt command)
```

引数

command 以下のいずれかを指定します。

ALLOW_PDF_ALL	すべてのバージョンのPDFを許容します。(既定)
ALLOW_PDF_14_TO_17	バージョン 1.4 から 1.7 のPDFのみを許容します。
ALLOW_PDF_10_TO_17	バージョン 1.0 から 1.7 のPDFのみを許容します。
ALLOW_PDF_14_TO_20	バージョン 1.4 から 2.0 のPDFのみを許容します。

戻り値

成功すると、0 (ゼロ) が戻ります。失敗すると、エラーコードが戻ります。

4.15 代替フォント指定

PDF Imager-LP はPDF 文書に指定されたフォントを別のフォントに替えて画像に変換することができます。

指定フォントを代替する場合は、PDF 文書に指定されているフォント名を使わなければなりません。このフォント名を取得するには、「4.5.5 PDF 文書内のフォント名」を利用します。

ご注意

PDF 文書を画像に変換した後に代替フォントを指定しなおして同じページを画像に変換しても新たに指定したフォントに代替されません。同じページを別のフォントに代替する場合はライブラリを終了する手順を実行してください。

4.15.1 既定フォントの指定

PDF文書に指定されたフォントや代替指定されたフォントが検索できない場合のフォントを指定します。
既定では、「MS ゴシック」または「MS 明朝」フォントを使って画像変換されます。

メソッド

```
int SetDefaultFont(string defaultFont)
```

引数

defaultFont 既定で使用するフォントの名前

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

4.15.2 代替フォントの指定

PDF文書に指定されたフォントに替えて画像化するフォントを指定できます。

メソッド

```
int SetAlternateFont(string originalFont, string alternateFont, int always)
int SetAlternateFont2(
    string originalFont, string alternateFont, string fontWeight,
    string fontStyle, int always, int replace
)
```

引数

originalFont	PDF文書に指定されたフォントの名前
alternateFont	代替するフォントの名前
fontWeight	代替するフォントの太さ “normal”、“bold”または、“100”、“200”、“300”、“400”、“500”、“600”、“700”、“800”、“900”のいずれかを指定します。なお、数値の300以下は“normal”と同じで、400以上は“bold”と同じになります。
fontStyle	“normal”、“italic”または、“oblique”のいずれかを指定します。なお、“italic”と“oblique”は同じです。
always	1(真)を指定すると、originalFont フォントがシステムにインストールされている場合でも alternateFont で代替されます。0(偽)を指定すると、originalFont フォントがシステムにインストールされている場合は代替されません。
replace	1(真)を指定すると、originalFont フォントがPDF文書に埋め込まれている場合でも alternateFont で代替されます。0(偽)を指定すると、originalFont フォントがPDF文書に埋め込まれている場合は代替されません。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

4.16 OCG(レイヤー)

PDF文書は、レイヤーを持つことができます。PDF Imager-LP は、OCGを初期状態で画像に変換することや各レイヤーの表示・非表示を指定して画像に変換することができます。

4.16.1 OCG の総数取得

PDF文書に記載されたレイヤーの総数を取得します。

メソッド

```
int GetOcgCount()
```

引数

ありません

戻り値

成功すると、0(ゼロ)以上の正数が戻ります。負数はエラーコードです。

4.16.2 OCG の状態取得

OCGのID、名前、階層、表示・非表示の状態を取得します。取得には、MlpOcgData 型(構造体またはクラス)を使います。

C/C++開発環境で使用する場合は、MlpOcgData 型のポインターを取得しますが、このポインターは一時的なもので、保存して後で利用することはできません。また、このポインターを開放してはいけません。

```
public class OcgData
{
    public int      id;
    public string   name;
    public int      level;
    public int      visible;
}
```

メンバー

id	各OCGにつけられたユニークな認識番号
name	PDF文書で示されたname
level	OCGの階層(ゼロがトップレベル)、-1の場合はそのOCGは階層に含まれません。
visible	0:OCGが非表示に設定されている、0以外:表示に設定されている

メソッド

```
OcgData[] GetOcgData()
```

引数

ありません

戻り値

成功すると OcgData[] が戻ります。

4.16.3 OCG の表示・非表示指定

OCGの表示・非表示を指定します。設定には、MlpOcgData 型(構造体またはクラス)を使います。詳細は、「4.15.2 OCGの状態」を参照してください。

メソッド

```
int SetOcgState(OcgData[] data)
```

引数

data	OCGのID、名前、階層、表示・非表示の状態を示す構造体(またはクラス) ただし、id および visible 以外は無視されます。
------	---

戻り値

成功すると、0(ゼロ)以上の正数が戻ります。失敗すると、エラーコードが戻ります。

4.17 透かし(ウォーターマーク)

Imager-LP は変換した画像に透かし(ウォーターマーク)を追加描画できます。

透かしは、画像や文字列それぞれを最大1つページ画像に追加描画できます。描画するZ軸位置としてページ画像の前面もしくは背面を選択できます。画像および文字列共に指定された場合で同じZ軸位置が指定された場合は文字列がより前面に描画されます。

4.17.1 透かしの設定

透かしの内容を設定します。

透かしとして文字を描画する場合は、エスケープ文字列を使ってPDF文書のタイトルや作成日時などを描画できます(「4.17.4 透かしのエスケープ文字」)参照。

PDF文書から変換された画像の原点は左上です。また、Y軸は画像の上から下が正方向です。

メソッド

```
int SetImageWatermark(          /* 画像を描画指定 */
    string fileName, int x_position, int y_position, int z_position,
    float x_scale, float y_scale, float alpha
)
int SetFillTextWatermark(       /* 塗りつぶしたテキストを描画指定 */
    string text, string fontName, int x_position, int y_position,
    int z_position, float fontSize, string color, float alpha
)
int SetStrokeTextWatermark(     /* 枠線で描画されたテキストを描画指定 */
    string text, string fontName, int x_position, int y_position,
    int z_position, float fontSize, string color, float alpha
)
```

引数

fileName	透かしとして描画する画像のファイル名(パス名) 貼り付ける画像の原点は、左上です。
text	透かしとして描画する文字列
fontName	文字列のフォント名
x_position	透かしを描画するX座標
y_position	透かしを描画するY座標
z_position	透かしを描画するZ軸位置(前面または背面) 0: 透かしは描画されません。 正数: 透かしはページ画像の前面に描画されます。 負数: 透かしはページ画像の背面に描画されます。
x_scale	画像をX方向に拡大する倍率です。(1.0超は拡大、1.0未満は縮小です。)
y_scale	画像をY方向に拡大する倍率です。(1.0超は拡大、1.0未満は縮小です。)
fontSize	文字列のサイズ(単位はポイント;1ポイント=1/72インチ)
color	文字列の色を16進数または色名で指定します。(「4.17.3 文字色指定」参照)
alpha	アルファチャネル値(不透明度) 0.0 から 1.0 の値を指定します。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

4.17.2 透かしを削除

既に指定した透かしを削除します。

通常は、PDF文書を閉じる処理(「4.24 PDF文書処理の終了」参照)によって透かし(の描画情報)は削除されます。それ以前に削除する場合に以下を使います。削除後に変換されたページ画像には透かしが描画されません。

メソッド

```
int ClearWatermarkData()
```

引数

ありません。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

4.17.3 文字色指定

透かしとして描画する文字列の色は16進数または色名で指定できます。

16進数で指定する場合は、以下の形式で指定します。

#rrggbb または #rgb

必ず7文字または4文字で指定します。r、g、bはそれぞれ16進数1文字を表します。

大文字と小文字は区別しません。

rr(r)は赤色(red)値で 0x00 から 0xFF までの値を指定します。

gg(g)は緑色(green)値で 0x00 から 0xFF までの値を指定します。

bb(b)は青色(blue)値で 0x00 から 0xFF までの値を指定します。

色名で指定する場合は、CSS(Cascading Style Sheets)に準じた色名を指定できます。指定できる色名は「11.色名称一覧」を参照してください。

4.17.4 透かしのエスケープ文字

透かし文字に以下のようなエスケープ文字列を含めると現在開いているPDF文書の情報に置き換わります。

文字列	内容	表示形式
%CDT	現在PDF文書の作成日時	“YYYY/MM/DD hh:mm:ss”
%CDt	現在PDF文書の作成年月日	“YYYY/MM/DD”
%Cdy	現在PDF文書の作成年	“YYYY”
%Cdm	現在PDF文書の作成月	“MM”
%Cdd	現在PDF文書の作成日	“DD”
%CTm	現在PDF文書の作成時刻	“hh:mm:ss”
%Cth	現在PDF文書の作成時	“hh”
%Ctm	現在PDF文書の作成分	“mm”
%Cts	現在PDF文書の作成秒	“ss”
%MDT	記載されている最新の更新日時	“YYYY/MM/DD hh:mm:ss”
%MDt	記載されている最新の更新年月日	“YYYY/MM/DD”
%Mdy	記載されている最新の更新年	“YYYY”
%Mdm	記載されている最新の更新月	“MM”
%Mdd	記載されている最新の更新日	“DD”
%MTm	記載されている最新の更新時刻	“hh:mm:ss”
%Mth	記載されている最新の更新時	“hh”
%Mtm	記載されている最新の更新分	“mm”
%Mts	記載されている最新の更新秒	“ss”
%Pgn	現在PDF文書の対象ページ番号	数値 (10進数)
%Pgc	現在PDF文書の総ページ数	数値 (10進数)
%Ttl	現在PDF文書のタイトル	(記載された文字列)
%%		“%”

4.18 コールバック関数

コールバック関数は、サイズの大きなPDF文書または複雑なPDF文書を画像に変換する場合や、解像度の高い画像を生成する際に、コンピュータの能力によってその変換時間が長い場合などで使用します。コールバック関数を指定すると、その進捗状況を知ることができます。なお、コールバック関数は、ライブラリと同じスレッドで実行されますので注意してください。

コールバック関数の定義は C/C++開発環境でのみ必要です。C#開発環境では既に定義されていますので、不要です。C#開発環境ではコールバックが抑制されていますので、それを解除してから利用します。解除の手順は「4.19 コールバックの抑制」を参照してください。

以下の関数で、コールバック関数を登録します。

```
void MlpSetCallbackFunc(int mode, int opt, void *fnc)
```

引数

mode	進捗を知らせるモードを指定します。指定できるモードは、「4.18.1 ページ解析終了のコールバック」および「4.18.2 画像生成進捗のコールバック」を参照してください。
opt	関数をコールする際のオプション(モードごとに意味が違います)
fnc	コールバックメソッド(モードごとに形式が違います)

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.18.1 ページ解析終了(画像変換開始)のコールバック設定

指定されたページを画像に変換する場合はまずそのページを解析し、その後に画像に変換します。

以下のモードとコールバック関数を指定すると、ページの解析後にその関数がコールされます。

mode	MLP_RENDERING_CB
opt	無視されます。整数の0(ゼロ)を指定してください。
fnc	<code>void __stdcall rendering_cb(int pageNum)</code> を指定します。 pageNum には、画像に変換されるページ番号が渡ります。

4.18.2 画像生成進捗のコールバック設定

指定されたライン数(ラインは、画像の横幅に相当します)を処理することに関数がコールされます。

なお、変換される画像形式ごとに処理が違いますので、関数がコールされるタイミングは必ずしも一定ではありません。

mode	MLP_PROGRESS_CB
opt	ライン数を指定します。このライン数処理終了後に関数がコールされます。
fnc	<code>void __stdcall progress_cb(int lines, int totalLines)</code> を指定します。 lines には処理したライン数が渡り、totalLines には生成される画像の全ライン数が渡ります。

4.18.3 パスストローク描画時のコールバック設定

PDF Imager-LP においてPDF文書のパスストロークを描画する場合は、指定された解像度とアンチエイリアスの関係から期待したとおりの画像に変換されない場合があります。このような場合に、PDF Imager-LP が生成した画像に替えてユーザーアプリケーションで作成した画像を貼り付けることができます。

指定された関数は PDF Imager-LP がパスストロークを画像に変換した後で、ページ画像 (生成する画像) へ描画する前にコールされます。

なお、パスストロークの描画コマンドを画像に変換する処理は、ユーザーアプリケーションで行います。

パスストロークや描画コマンドなどの詳細は、「PDF Reference」を参照してください。

```
mode      MLP_DRAW_STROKE_PATH_CB
opt        無視されます。整数の0(ゼロ)を指定してください。
fnc        int __stdcall draw_stroke_path_cb(StrokePathAttr *pa, unsigned
           char *pixels)を指定します。
           paには、以下のパスストローク属性が渡ります。
           typedef struct StrokePathAttr_s {
               float   lineWidth;           //パスの線幅
               int      lineCap;             //線の始点・終点形状
               int      lineJoin;            //線接続点の形状
               float    miterLimit;          //とがった線接続点の形状
               int      dashLen;             //点線定義配列の長さ
               float    dashPhase;           //最初の点線の長さ
               float    dashData[32];        //点線定義配列
               float    colorfv[5];          //線描画色
               char     *pathString;         //パスストローク コマンド文字列
               int      x, y, width, height; //変換された画像の位置とサイズ
               int      colors;              //描画色の色数
           } StrokePathAttr;
```

戻り値 コールバック関数からの戻り値によって貼り付けられる画像が変わります。
0(ゼロ) 画像を貼り付けません。
1 ユーザーアプリケーションによって変更された pixel 値のパス画像を描画します。
2 PDF Imager-LP が生成した画像を貼り付けます。

その他 線描画色はカラーの場合 RGBA (Red, Green, Blue, Alpha 値) の順で格納されグレースケール (白黒二値も含みます) の場合はグレースケール値, Alpha 値の順で格納されています。ピクセルデータは、画像の色数が3の場合は3バイトでひとつの画素を表し RGB (Red, Green, Blue) の順で格納されています。画像の色数が1の場合は、1バイトでひとつの画素を表し、グレースケール値です。色数が4色および2色の場合は、各画素値に Alpha 値が追加されます。さらに、画素データは行 (横方向スキャン) がライン (縦方向) 数隙間無く格納されたデータです。画像の原点は、左上です。

ご注意:

このコールバック関数を利用すると、生成される画像の解像度によってはピクセルデータのコピーなどに大きな時間を要する場合があります。

以下の関数を使うと、コールバックを抑制したり実行したりできます。

```
int MlpSetPicture(MLP_STROKEPATH_ENABLE_CALLBACK, int flag);
flag を 1 (真) にすると、コールバック関数が実行され、0 (偽) にすると実行されません。
```

コールバックを C#開発環境で利用する場合は、既定でコールバックが抑制されています。

4.18.4 タイル画像定義時のコールバック設定

タイルやタイルを敷き詰めた画像をユーザーアプリケーションで作成したものと替えるために、タイルの画像定義時にユーザー関数をコールバックします。

ただし、コールバックされるのはタイルが画像だけで構成されている場合に限りです。

```
mode    MLP_DEFINE_IMAGE_TILE_CB
opt      無視されます。整数の0(ゼロ)を指定してください。
fnc      void __stdcall define_image_tile_cb(ImageTileAttr *ta, unsigned
          char *pixels)を指定します。
          pa には、以下のタイル属性が渡ります。
          typedef struct ImageTileAttr_s {
              int      width, height, colors;    //タイルのサイズ、画像の色数
              float    xstep, ystep;            //タイル敷詰め縦横ステップ
              Matrix    ctm;
          } ImageTileAttr;
```

なお、MatrixはPDF文書に定義されたタイルをユーザー指定空間に写像する行列式の係数です。

戻り値 ありません。

その他 ピクセルデータは、画像の色数が3の場合は3バイトでひとつの画素を表し RGB (Red, Green, Blue) の順格納されています。画像の色数が1の場合は、1バイトでひとつの画素を表し、グレースケール値です。色数が4色および2色の場合は、格画素値に Alpha 値が追加されます。さらに、画素データは行(横方向スキャン)がライン(縦方向)数隙間無く格納されたデータです。
画像の原点は、左上です。

ご注意:

このコールバック関数を利用すると、生成される画像の解像度によってはピクセルデータのコピーなどに大きな時間を要する場合があります。

以下の関数を使うと、コールバックを抑制したり実行したりできます。

```
int MlpSetPicture(MLP_TILE_IMAGE_ENABLE_CALLBACK, int flag);
```

flag を 1 (真) にすると、コールバック関数が実行され、0 (偽) にすると実行されません。

コールバックを C#開発環境で利用する場合は、既定でコールバックが抑制されます。

4.18.5 タイルマスク画像定義時のコールバック設定

タイルやタイルを敷き詰めた画像をユーザーアプリケーションで作成したものと替えるために、タイルのマスク画像定義時にユーザー関数をコールバックします。

ただし、コールバックされるのはタイルが画像だけで構成されている場合に限りです。

```
mode    MLP_DEFINE_MASK_IMAGE_TILE_CB
opt      無視されます。整数の0(ゼロ)を指定してください。
fnc      void __stdcall define_image_image_tile_cb(ImageTileAttr *ta,
            unsigned char *pixels)を指定します。
pa には、以下のタイル属性が渡ります。
        typedef struct ImageTileAttr_s {
            int      width, height, colors; //タイルのサイズ、画像の色数
            float     xstep, ystep;         //タイル敷詰め時の縦横ステップ
            Matrix     ctm;
        } ImageTileAttr;
```

なお、MatrixはPDF文書に定義されたタイルをユーザー指定空間に写像する行列式の係数です。

戻り値 ありません。

その他 マスクのピクセルデータは、画像の色数が1となります。1バイトでひとつの画素を表し、グレースケール値です。Alpha 値は含まれません。
縦横ステップの値は意味がありません。
画素データは行(横方向スキャン)がライン(縦方向)数隙間無く格納されたデータです。
画像の原点は、左上です。

ご注意:

このコールバック関数を利用すると、生成される画像の解像度によってはピクセルデータのコピーなどに大きな時間を要する場合があります。

以下の関数を使うと、コールバックを抑制したり実行したりできます。

```
int MlpSetPicture(MLP_TILE_MASK_IMG_ENABLE_CALLBACK, int flag);
flag を 1 (真) にすると、コールバック関数が実行され、0 (偽) にすると実行されません。
```

コールバックを C#開発環境で利用する場合は、既定でコールバックが抑制されます。

4.18.6 タイル処理時のコールバック設定

ユーザーアプリケーションではタイル画像やタイルのマスク画像を元にして、それを敷き詰めた画像を作成できます。PDF Imager-LP は、ユーザーアプリケーションで作成した画像をページ画像 (PDF 文書を変換した画像) に貼り付けられるようにユーザー関数をコールバックします。

ただし、コールバックされるのはタイルが画像だけで構成されている場合に限ります。

```
mode    MLP_DRAW_TILED_IMAGE_CB
opt      無視されます。整数の0(ゼロ)を指定してください。
fnc      int __stdcall define_image_image_tile_cb(ImageTileAttr *ta1,
          unsigned char *pixels1, ImageTileAttr *ta2, unsigned char
          *pixels2)を指定します。
          ta1, ta2 には、以下のタイル属性が渡ります。
          typedef struct ImageTileAttr_s {
              int      width, height, colors;  //画像のサイズ、画像の色数
              float    xstep, ystep;
              Matrix    ctm;
          } ImageTileAttr;
```

なお、ta1, pixel1 はタイルの画像で、ta2, pixel2 はタイルを敷き詰めた画像です。

また、xstep、ystep、ctm の値は0(ゼロ)となり意味がありません。

戻り値 コールバック関数からの戻り値によって貼り付けられる画像が変わります。

0(ゼロ) 画像を貼り付けません。

1 ユーザーアプリケーションによって変更された pixel2 値のタイルを敷き詰画像をページ画像 (PDF 文書を変換した画像) に描画します。

2 ユーザーアプリケーションによって変更されたタイル画像 pixel1 を使い、それを敷き詰めた画像をページ画像 (PDF 文書を変換した画像) に描画します。

3 PDF Imager-LP が生成した、タイルを敷き詰めた画像をページ画像に貼り付けます。

その他 ピクセルデータは、画像の色数が3の場合は3バイトでひとつの画素を表し RGB (Red, Green, Blue) の順格納されています。画像の色数が1の場合は、1バイトでひとつの画像をあらわし、グレースケール値です。色数が4色および2色の場合は、格画素値に Alpha 値が追加されます。さらに、画素データは行 (横方向スキャン) がライン (縦方向) 数隙間無く格納されたデータです。

画像の原点は、左上です。

ご注意:

このコールバック関数を利用すると、生成される画像の解像度によってはピクセルデータのコピーなどに大きな時間を要する場合があります。

以下の関数を使うと、コールバックを抑制したり実行したりできます。

```
int MlpSetPicture(MLP_TILING_ENABLE_CALLBACK, int flag);
```

flag を 1 (真) にすると、コールバック関数が実行され、0 (偽) にすると実行されません。

コールバックを C# 開発環境で利用する場合は、既定でコールバックが抑制されます。

4.19 コールバックの抑制

C/C++開発環境では、コールバック関数を定義することで、パスストロークやタイル処理時にコールバックされるようになります。しかし C#開発環境では、コールバック関数が既定で登録されていてその動作が抑制されています。そのため、C#開発環境でこれらのコールバックを利用する場合は、その動作が抑制されないように指定しなければなりません。

4.19.1 パスストローク描画時コールバックの抑制

パスストローク描画時のコールバックを抑制します。
以下の関数を使いコールバックを抑制したり実行したりします。

メソッド

```
int SetPicture(PictureOpt.STROKEPATH_ENABLE_CALLBACK, int flag)
```

引数

flag	1 (真) を指定するとコールバック関数が実行され、0 (偽) にするとコールバックは抑制されます。
------	--

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.19.2 タイル画像定義時コールバックの抑制

タイル画像が定義された際のコールバックを抑制します。
以下の関数を使いコールバックを抑制したり実行したりします。

メソッド

```
int SetPicture(PictureOpt.TILE_IMAGE_ENABLE_CALLBACK, int flag)
```

引数

flag	1 (真) を指定するとコールバック関数が実行され、0 (偽) にするとコールバックは抑制されます。
------	--

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.19.3 タイルマスク画像定義時コールバックの抑制

タイルマスク画像が定義された際のコールバックを抑制します。
以下の関数を使いコールバックを抑制したり実行したりします。

メソッド

```
int SetPicture(PictureOpt.TILE_MASK_IMG_ENABLE_CALLBACK, int flag)
```

引数

flag	1 (真) を指定するとコールバック関数が実行され、0 (偽) にするとコールバックは抑制されます。
------	--

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.19.4 タイル処理時コールバックの抑制

タイル処理時のコールバックを抑制します。
以下の関数を使いコールバックを抑制したり実行したりします。

メソッド

```
int SetPicture(PictureOpt.TILING_ENABLE_CALLBACK, int flag)
```

引数

flag	1 (真) を指定するとコールバック関数が実行され、0 (偽) にするとコールバックは抑制されます。
------	--

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.20 変換された画像をパネルに描画

PDF Imager-LP は、その内部に特別な画像を生成できます。この画像を PDF Imager-LP では「パネル」と呼びます。

パネルにはPDF文書を変換したページ画像を1つ以上貼り付けることができます。さらに、ページ画像は別のPDF文書を変換した画像でも貼り付けられます。作成されたパネルは指定された画像形式で取得できます。

一般に以下の手順で複数のページ画像をパネルに貼り付けます。

1. ライブラリを初期化 (Initialize)
2. パネルを生成 (CreatePanel)
3. PDF文書を開く (OpenDoc)
4. 変換する画像の色空間や解像度などを指定
(SetPictureRGB、SetPictureResolution など)
5. PDF文書のページを指定して内部の画像データに変換 (CreatePictMem)
6. 3、4、5の手順を必要に応じて繰り返します。
7. パネルのデータを画像形式で取得 (PanelToFile)
8. PDF文書を閉じる (CloseDoc)
9. ライブラリを終了 (Uninitialize)

4.20.1 パネル作成

複数のページ画像を貼り付けられるパネルを生成します。生成されたパネルは不透明白色でクリアされています。

パネルはライブラリ内部に1つだけ作成されます。既に作成されている場合は、そのパネルを削除して新たなパネルが作成されます。作成されたパネルは、FreePictMem で削除できます。

メソッド

```
int CreatePanel(int width, int height, int colorspace)
```

引数

width	パネルの横方向のピクセル数
height	パネルの縦方向のピクセル数
colorspace	色空間 (MLP_PICTURE_RGB または MLP_PICTURE_GRAY)

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.20.2 パネルをクリア

複数のページ画像を貼り付けられるパネル全体を不透明白色で塗りつぶします。

メソッド

```
int ClearPanel();
```

引数

ありません

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.20.3 パネルにページ画像を貼り付ける

メモリー上のページ画像をパネルに貼り付けます。

メモリーにはあらかじめ MlpCreatePictMem を使ってページ画像を作成しておかなければなりません(メモリー上に画像がない場合などは貼り付けができません)。

メモリー上の画像はパネルの色空間と同じ色空間に自動で変換されます。ページ画像をパネルに貼り付ける際に画像のサイズや解像度は変更されません。ページ画像にアルファチャンネルはありませんので、後から描画したページ画像でそれまでのページ画像に上書きされます。ページ画像の背景は不透明白色ですので注意してください。

パネル画像やページ画像の原点 (x=0, y=0) は、左上です。

メソッド

```
int MemPictToPanel(int xPos, int yPos, float alpha)
```

引数

xPos	ページ画像を貼り付けるパネルの X 座標
yPos	ページ画像を貼り付けるパネルの Y 座標
alpha	貼り付けるページ画像の不透明度(0.0 から 1.0 の間で指定)

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.20.4 パネルにページ画像の境界を指定して貼り付ける

メモリー上のページ画像の境界を指定してパネルに貼り付けます。境界を指定すると、画像を切り取ることなどができます。

指定された領域の内部がパネルに貼り付けられます。指定された領域にページ画像がない部分はパネルに描画されません。

メモリーにはあらかじめ CreatePictMem を使ってページ画像を作成しておかなければなりません(メモリー上に画像がない場合などは貼り付けができません)。

メモリー上の画像はパネルの色空間と同じ色空間に自動で変換されます。ページ画像をパネルに貼り付ける際に画像のサイズや解像度は変更されません。ページ画像にアルファチャネルはありませんので、後から描画したページ画像でそれまでのページ画像に上書きされます。ページ画像の背景は不透明白色ですので注意してください。

パネル画像やページ画像の原点 (x=0, y=0) は、左上です。

メソッド

```
int MemPictToPanelBBox(  
    int xPos, int yPos, float alpha, int bboxX, int bboxY,  
    int bboxWidth, int bboxHeight  
)
```

引数

xPos	ページ画像を貼り付けるパネルの X 座標
yPos	ページ画像を貼り付けるパネルの Y 座標
alpha	貼り付けるページ画像の不透明度 (0.0 から 1.0 の間で指定)
bboxX	貼り付けるページ画像から切り取る領域の X 座標 (ページ画像上の座標)
bboxY	貼り付けるページ画像から切り取る領域の Y 座標 (ページ画像上の座標)
bboxWidth	貼り付けるページ画像から切り取る領域の幅 (ピクセル値)
bboxHeight	貼り付けるページ画像から切り取る領域の高さ (ピクセル値)

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.20.5 パネルに矩形を描画

パネルに矩形を描画します。

ページ画像を切り取った場合など、その領域を明示したい場合などに利用します。

パネル画像の原点 (x=0, y=0) は、左上です。矩形は不透明な指定色で描画されます。

メソッド

```
int PaintRectToPanel(  
    int xPos, int yPos, int width, int height, int lineWidth, byte[] color)  
int PaintRectToPanel2(  
    int xPos, int yPos, int width, int height, int lineWidth, string colorStr)
```

引数

xPos	矩形を描画するパネルの X 座標
yPos	矩形を描画するパネルの Y 座標
width	描画する矩形の幅 (ピクセル値)
height	描画する矩形の高さ (ピクセル値)
lineWidth	矩形の線幅 (ピクセル値)
color	矩形線の色配列 (カラーの場合はRGBの3色、グレースケールの場合は1色) 各色は0~255の値で指定
colorStr	色名

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.20.6 パネルのデータを画像で取得

パネルのデータを画像に変換しファイルで取得します。

メソッド

```
int PanelToFile(string fileName)
```

引数

fileName

画像ファイル名

画像形式はファイルの拡張子によって以下のように自動で選択されます。

拡張子が“png”の場合、PNG形式の画像

拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像

拡張子が“tiff”または“tif”の場合、TIFF形式の画像

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.20.7 パネル削除

作成されたパネルを削除し、そのメモリー領域を開放します。

パネル領域は、ライブラリの使用終了で自動的に削除されるためメモリー領域に問題がない場合は開放しなくともかまいません。

メソッド

```
int FreePanel()
```

引数

ありません

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

4.21 画像変換の並列処理

PDF Imager-LP は、複数ページを画像に変換する場合に、それぞれをスレッドで並列に処理します。また、各ページを画像に変換する場合でも、そのページをより小さな領域に分割して並列処理します。

規定では、単一のページを分割してスレッドで並列に処理し、ページごとには並列処理しません。なお、これらの動作は抑制することができ、実行するスレッドの最大数を指定できます。

4.21.1 複数ページの並列処理の抑制

複数ページを画像変換する場合にスレッドによる並列処理を抑制します。

メソッド

```
int SetPicture(PictureOpt.THREAD_EACH_PAGE, int flag)
```

引数

flag 0(ゼロ)を指定すると、並列処理は抑制されます。それ以外の場合は抑制されません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.21.2 複数ページの並列処理の最大スレッド数

複数ページを画像変換する場合の最大スレッド数(並列処理数)を変更します。

メソッド

```
int SetPicture(PictureOpt.THREAD_EACH_PAGE_COUNT, int num)
```

引数

num 実行されるスレッドの総数を指定します。省略した場合は、使用しているコンピュータの論理CPU数と同じ数が使用されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.21.3 単一ページの並列処理の抑制

各ページを画像変換する場合の並列処理を抑制します。

メソッド

```
int SetPicture(PictureOpt.THREAD_SPLIT_PAGE, int flag)
```

引数

flag 0(ゼロ)を指定すると、並列処理は抑制されます。それ以外の場合は抑制されません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.21.4 単一ページの並列処理の最大スレッド数

各ページを画像変換する場合の並列処理の最大スレッド数(並列処理数)を変更します。

メソッド

```
int SetPicture(PictureOpt.THREAD_SPLIT_PAGE_COUNT, int num)
```

引数

num 実行されるスレッドの総数を指定します。省略した場合は、使用しているコンピュータの論理CPU数と同じ数が使用されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

4.22 画像変換の結果

PDF Imager-LP はPDF文書にフォントが埋め込まれていない場合に代替フォントを使って文字を画像に変換します。この際に、PDF文書で指定された文字コードのグリフ(字形)が代替されたフォントに存在しない場合があります。この場合は、一般にグリフが存在しない場合の特殊な字形(未定義グリフ)が表示されます。

このように文字が「未定義グリフ」に変換されたことを、画像生成後に知ることができます。

以下の関数を利用して、未定義グリフに変換された文字数を取得できます。

メソッド

```
int GetDocumentInfo(PictureOpt.GLYPH_NOT_FOUND_COUNT, out int count)
```

引数

count 未定義グリフに変換された文字の総数。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

ご注意ください。

この関数で取得できる総数は、「正しい文字(グリフ)に変換されなかった文字の総数」ではありません。画像への変換時に指定されたフォントに、対照のPDF文書で指定されたグリフ(字形)が存在しておらず、未定義グリフに変換された文字の総数です。

この値を読み出すと結果は0(ゼロ)にリセットされます。

4.23 メタデータ

PDF文書のメタデータをXMP(Extensible Metadata Platform)として解析および変更します。

メタデータに記載されたプロパティの取得および変更は XmpInterface クラス(C#の場合)または XMP_TK_HANDLE を介して実行します。

C#開発環境の場合:

```
Imr = new PdfImager();  
XmpInterface xmp = imr.GetXmpInterface();
```

C/C++開発環境の場合:

```
XMP_TK_HANDLE handle = imr.GetXmpInterface();
```

戻り値

0はエラーです。それ以外は値が取得できていますので、取得や編集の処理を行えます。

4.24 PDF文書処理の終了

PDF文書の画像変換処理を終了します。

メソッド

```
void CloseDoc()
```

引数

ありません。

戻り値

ありません。

4.25 ライブラリの使用終了

ライブラリの使用を終了します。

メソッド

```
void Uninitialize()
```

引数

ありません。

戻り値

ありません。

5.0 XmpInterface インターフェース

PDF 文書のメタデータはXML (Extensible Markup Language) で記載され、その内容はXMP (Extensible Metadata Platform) に従って記載されることが推奨されています。

PDF Imager-LP このようなメタデータを解析または変更します。なお、解析・変更はPDF文書に記載されたメタデータをシリアル化した後に実施します。

XMPメタデータは XmpInterface クラスを介して解析変更します。

XMP規格は、ISO 16684-1 を参照してください。

5.1 XmpInterface の取得

メソッド

<code>XmpInterface</code> GetXmpInterface ()	C # の場合
<code>XMP_TK_HANDLE</code> MlpGetXmpInterface ()	C / C ++ の場合

引数

ありません。

戻り値

0	エラー
0 以外	成功

エラーの多くは XMPCode.dll が参照できない場合に起こります。このDLLを参照可能なフォルダーに格納してください。

5.2 XmpInterface の終了

XmpInterface は現在文書をクローズすると終了しますが、以下の手順でも終了させることができます。

メソッド

`void` CloseInterface ()

引数

ありません。

戻り値

ありません。

5.3 メタデータを更新

プロパティが変更されたメタデータでPDF文書を更新します。

更新されたPDFファイルを作成するためにはこの処理が必要です。

メソッド

`int` UpdateDocument ()

引数

ありません

戻り値

負数	エラーコード (指定された名前空間URIが不正な場合など)
0	成功

5.4 PDF文書をファイルに格納

現在のPDF文書をファイルに格納します。

格納にはPDFファイルを作成できるライセンスが必要です。このライセンスがない場合には、格納されたPDF文書に透かしが挿入され、格納の後にページを画像化した場合にも透かしが挿入されます。

メソッド

```
int DocumentToFile(string fileName)
```

引数

fileName	作成するファイルのパス名
----------	--------------

戻り値

負数	エラーコード（指定された名前空間URIが不正な場合など）
0	成功

5.5 プロパティの有無および削除

プロパティがメタデータに存在するかを確認または削除します。確認の場合はプロパティ値の有無は無関係です。削除の場合、プロパティは、その種類に関係なく、必ず削除されます。

Simple(単純)プロパティ、Array(配列)プロパティ、Structure(構造)プロパティに限定する場合はそれぞれ、「5.3Simple プロパティ」、「5.4Array プロパティ」、「5.5Structure プロパティ」を参照して下さい。

メソッド

```
int DoesPropertyExist(string uri, string property)  
int DeleteProperty(string uri, string property)
```

引数

uri	名前空間 URI
property	プロパティ名

戻り値

負数	エラーコード（指定された名前空間URIが不正な場合など）
0	正しく削除された、または property が存在しない
1 以上	property が存在する

5.6 Simple(単純)プロパティ

5.6.1 プロパティの有無

指定されたプロパティが「Simple(単純)プロパティ」として存在するかを確認します。プロパティの値の有無は無関係です。

メソッド

```
int DoesSimplePropertyExist(string uri, string property)
```

引数

uri	名前空間 URI
property	プロパティ名

戻り値

負数	エラーコード (指定された名前空間URIが不正な場合など)
0	名前空間URIがSimpleプロパティではない、またはSimpleプロパティに property が存在しない。
1 以上	property がSimpleプロパティに存在する

5.6.2 プロパティ値の取得および設定

Simpleプロパティの値を取得または設定します。設定する場合にそのプロパティが存在しない場合は新たに追加されますが、独自プロパティの場合は名前空間URIをXmpInterfaceへ登録しなければなりません、「5.7 独自名前空間」を参照してください。

メソッド

```
string GetSimpleProperty(string uri, string property)  
int GetSimpleProperty(string uri, string property, out string value)  
int SetSimpleProperty(string uri, string property, string value)
```

引数

uri	名前空間 URI
property	プロパティ名
value	プロパティ値

戻り値

負の整数値	エラーコード
正の整数値	property が存在し、値を取得できた場合に文字列のバイト数
0	property が存在し、値を設定した
文字列	property がSimpleプロパティの場合のプロパティ値

5.6.3 プロパティの削除

Simple(単純)プロパティを削除します。

メソッド

```
int DeleteSimpleProperty(string uri, string property)
```

引数

uri	名前空間 URI
property	プロパティ名

戻り値

負数	エラーコード
0	削除に成功、プロパティが存在しない

5.7 Array(配列)プロパティ

5.7.1 プロパティの有無

指定されたプロパティが「Array(配列)プロパティ」として存在するか、またはアイテムが存在するかを確認します。値の有無は無関係です。

メソッド

```
int DoesArrayPropertyExist(string uri, string property)  
int DoesArrayItemExist(string uri, string property, int index)
```

引数

uri	名前空間 URI
property	プロパティ名
index	アイテム番号(先頭の番号は1)

戻り値

負数	エラーコード(指定された名前空間URIが不正な場合など)
0	Arrayプロパティではない、property がArrayプロパティに存在しない、もしくは index 番号のアイテムがない。
1 以上	指定された property または index のアイテムが存在する。

5.7.2 プロパティ、アイテムの追加

新たな Array (配列) プロパティを追加し、アイテム値を設定します。
既にプロパティが存在している場合は、値が最後のアイテムとして追加されます。

メソッド

```
int AppendArrayItem(string uri, string property, long option, string value)
```

引数

uri	名前空間 URI
property	プロパティ名
option	新たにプロパティを追加する場合のタイプ、既存の場合は無視されます。 以下のいずれかをしています。 kXMP_PropArrayIsOrdered kXMP_PropArrayIsAlternate kXMP_PropArrayIsAltText
value	作成された Array プロパティのアイテム値

戻り値

負数	エラーコード
0	成功

5.7.3 アイテム数

Array (配列) プロパティのアイテム数を取得します。

メソッド

```
int CountArrayItems(string uri, string property)
```

引数

uri	名前空間 URI
property	プロパティ名

戻り値

負数	エラーコード
正数	Array (配列) プロパティのアイテム数

5.7.4 アイテム値の取得および設定

Array(配列)プロパティのアイテム値を取得または設定します。

メソッド

```
string GetArrayItems(string uri, string property)
int GetArrayItems(string uri, string property, int index, out string value)
int SetArrayItems(string uri, string property, int index, string value)
```

引数

uri	名前空間 URI
property	プロパティ名
index	Array プロパティのアイテムのインデックス番号 GetArrayItems ではアイテム数 + 1 とすると、最後のアイテムとして追加されます。
value	アイテム値

戻り値

負の整数値	エラーコード
正の整数値	property が存在し、値が取得できた場合に文字列のバイト数
0	property が存在し、値を設定できた
文字列	property が Array プロパティの場合のプロパティ値

5.7.5 アイテムの削除

Array(配列)プロパティのアイテムを削除します。

メソッド

```
int DeleteArrayItem(string uri, string property, int index)
```

引数

uri	名前空間 URI
property	プロパティ名
index	アイテム番号 (non-zero ベース)

戻り値

負数	エラーコード
0	成功、プロパティまたはアイテムが存在しない

5.8 Structure(構造)プロパティ

5.8.1 プロパティ、フィールドの有無

指定されたプロパティやフィールドが「Structure(構造)プロパティ」に存在するかを確認します。プロパティの値の有無は無関係です。

メソッド

```
int DoesStructPropertyExist(string uri, string property)
int DoesStructFieldExist(string uri, string property,
                        string fieldUri, string fieldName)
```

引数

uri	名前空間 URI
property	プロパティ名
fieldUri	Structure プロパティのフィールド名 名前空間 URI
fieldName	Structure プロパティのフィールド名

戻り値

負数	エラーコード
0	property または fieldName が存在しない
1 以上	property または fieldName が存在する

5.8.2 フィールド値の取得および設定

Structure プロパティのフィールド値を取得または設定します。
プロパティが無い場合やフィールドが無い場合は新たに作成されます。

メソッド

```
string GetStructField(string uri, string property, string fieldUri,
                    string fieldName)
int GetStructField(string uri, string property, string fieldUri,
                 string fieldName, out string fieldValue)
int SetStructField(string uri, string property, string fieldUri,
                 string fieldName, string fieldValue)
```

引数

uri	名前空間 URI
property	プロパティ名
fieldUri	フィールド名 名前空間 URI
fieldName	フィールド名
fieldValue	フィールド値

戻り値

負数	エラーコード
0	フィールド値の設定成功
1 以上	取得したフィールド値のバイト数
文字列	取得したフィールド値

5.8.3 フィールドの削除

Structure プロパティのフィールドを削除します。

メソッド

```
int DeleteStructField(string uri, string property,  
                     string fieldUri, string fieldName)
```

引数

uri	名前空間 URI
property	プロパティ名
fieldUri	フィールド名前空間 URI
fieldName	フィールド名

戻り値

負数	エラーコード
0	削除成功、またはプロパティやフィールドが存在しない

5.9 日付のプロパティ値

プロパティ値をローカルの日時データとして取得または設定します。

メソッド

```
DateTime GetPropertyDate(string uri, string property)  
int GetPropertyDate(string uri, string property, out DateTime datetime)  
int SetSimplePropertyDate(string uri, string property, DateTime datetime)  
int SetSimplePropertyDate(string uri, string property, int year, int month,  
                          int day, int hour, int minute, int second)  
int SetSimplePropertyCurrentDate(string uri, string property)
```

引数

uri	名前空間 URI
property	プロパティ名
datetime	取得または設定する日時データ
year	年
month	月
day	日
hour	時
minute	分
second	秒

戻り値

負数	エラーコード
0	成功
DateTime	日時データ

5.10 独自名前空間

独自の名前空間を使用する場合は、XmpInterface への登録が必要です。

メソッド

```
string RegisterNamespace (string uri, string property)
int RegisterNamespace (string uri, string suggested, out string registered)
```

引数

uri	名前空間 URI
suggested	プロパティのプリフィックス
registered	プロパティの登録されたプリフィックス

戻り値

負数	エラーコード
0	成功
文字列	名前空間 URI に関連付けられたプロパティのプリフィックス

5.11 XMP データをダンプする

現在文書のXMPデータをダンプします。

メソッド

```
int DumpObjectToConsole ()
int DumpObjectToFile (string fileName)
int DumpObjectToChar (out string data)
```

引数

fileName	出力ファイル名
data	出力文字列

戻り値

負数	エラーコード
0	成功

5.12 エラーの詳細

エラーコードが戻された場合に、より詳細なエラー内容を取得できる場合があります。
このメソッドで取得できるメッセージは必ずしも直前のものではありません。

メソッド

```
string GetLastErrorMessage ()
int GetLastErrorMessage (out string message)
```

引数

message	エラーのより詳細な内容
---------	-------------

戻り値

負数	エラーコード
0	成功

6.0 関数 – Python 開発環境

Python開発環境での実行手順と利用できる関数(メソッド)です。
PdfImagerLPモジュールの文字コードは UTF-8 です。他の文字コードでは機能しない関数があります。

6.1 PdfImagerLP モジュール

PdfImagerLP モジュールはネイティブDLL(_PdfImagerLP.pyd および PdfImagerLP.dll)を参照します。このモジュールに定義されている Mlp クラスを介して画像化等の処理を実施します。

上記のネイティブDLLは PYTHONPATH に配置するか、もしくはDLLの検索パスを追加して利用できるようにします。

以下は検索パスを追加してモジュールをインポートする場合の手順です。

```
#DLL の検索パスを変更する場合は以下を実行します。  
# なお、<Imager-LP インストールパス>は Imager-LP をインストールしたパスに読み替えてください。  
import sys  
if sys.maxsize > 2**32:  
    sys.path.insert(1, '<Imager-LP インストールパス>/lib/x64')  
else:  
    sys.path.insert(1, '<Imager-LP インストールパス>/lib/Win32')  
import PdfImagerLP
```

6.2 インスタンス化および初期化

Imager-LP クラスはインスタンス化およびライセンスキーを使った初期化が必要です。
以下手順でインスタンス化と初期化を行います。

インスタンス化(例)

```
mlp = PdfImagerLP.Mlp()
```

メソッド

```
int Initialize(licensekey)
```

引数

Licensekey PDF Imager-LP を使用するためのライセンス文字列

戻り値

ライブラリの初期化に成功すると0(ゼロ)が戻ります。それ以外はエラーコードです。

6.3 ライセンス情報の表示または取得

PDF Imager-LP の初期化の後はライセンスキー内容を文字列で表示または取得ができます。

メソッド

```
void ShowLicenseInfo()  
string LicenseInfoStr()
```

引数

ありません

戻り値

ShowLicenseInfo は戻り値がありません

LicenseInfoStr は成功すると、ライセンス内容の文字列が戻ります。

6.4 入力ファイル(PDFまたは画像データ)を開く

PDF 文書を開く際は、そのPDF 文書が印刷する場合の解像度を低解像度に指定されている場合や、印刷そのものが禁止されている場合があります。そのような場合PDF 文書はパスワードなどで暗号化されています。Imager-LP は、パスワードで暗号化されたPDF 文書を復号して画像に変換できます。また、開くモードを「表示」や「印刷」に指定して適切に開くことができます。

PDF Imager-LP は、PDF 文書以外に画像データを入力として開くことができます。入力画像はそのファイルの拡張子やデータの内容によって判別され適切に解釈されます。

6.4.1 PDFファイルの許可フラグを無視して開く

画像に変換するPDFファイルを開きます。

この関数はPDFファイルが暗号化されている場合でも、PDF 文書に指定された許可フラグを無視します。なお、パスワードは、PDFファイルがパスワードで暗号化されている場合のみ使用し、暗号化されていない場合は無視します。

メソッド

```
int OpenDoc(string filename, string ownerPassword, string userPassword)
```

引数

filename	PDF のファイルパス
ownerPassword	所有者パスワード ⁴
userPassword	ユーザーパスワード ⁵

戻り値

成功すると0(ゼロ)が戻り、失敗した場合はエラーコードが戻ります。

6.4.2 PDFファイルを指定モードで開く

画像に変換するPDFファイルを開くモード(「表示」または「印刷」)に従って開きます。

この関数はPDFファイルが暗号化されている場合、PDF 文書に指定された許可フラグに従って開きます。そのため、印刷が許可されないPDFファイルを「印刷」モードで開くと失敗します。また、低解像度印刷指定の場合は失敗しませんので、適切に画像化解像度を指定するため許可フラグを確認する必要があります。(許可フラグは、「4.5.2 PDF文書のパーミッション(許可)フラグ」を参照してください。)

なお、パスワードは、PDFファイルがパスワードで暗号化されている場合のみ使用し、暗号化されていない場合は無視します。

メソッド

```
int OpenDocUsage(string filename, string password, int mode)
```

引数

filename	PDF文書のファイルパス名
password	パスワード オーナーパスワード、ユーザーパスワードのいずれかを指定します。
mode	1(表示)または、2(印刷)を指定します。

戻り値

成功すると0(ゼロ)が戻り、失敗した場合はエラーコードが戻ります。

⁴ 所有者パスワードは「権限パスワード」と呼ばれることがあります。

⁵ ユーザーパスワードは「開くパスワード」と呼ばれることがあります。

6.4.3 画像ファイルを開く

画像ファイルを開く場合は、OpenDoc または OpenDocUsage メソッドを利用します。ただし、ファイル名以外の引数は無視されます。また、画像形式はファイルの拡張子およびデータ内容から適切に解釈され内部データ(ピクセルデータ)に変換されます。

読み込まれた画像は、白色不透明な背景画像に描画されます。そのため、画像データにアルファチャンネル(不透明を指定したデータ)が含まれていても出力画像には現れません。

6.5 PDF文書の情報

PDF Imager-LP は、開いたPDF文書の情報を取得できます。

6.5.1 PDF文書のページ数取得

現在開いているPDF文書の総ページ数を取得します。

メソッド

```
int PageCount()
```

引数

ありません

戻り値

成功すると、PDF文書の総ページ数(0(ゼロ)の場合もあります)が戻り、失敗した場合はエラーコードが戻ります。なお、エラーコードは負数です。

6.5.2 PDF文書のメタデータ

PDF文書に記載されたメタデータをバイトデータまたは文字列で取得します。メタデータにはUTF-8文字コードで記載されています。

メタデータの詳細は、「PDF Reference」 10.2 Metadata を参照してください。

メソッド

```
string GetMetadataString()
```

引数

ありません

戻り値

成功すると現在文書のメタデータ文字が戻り、失敗した場合はNull文字列が戻ります。

6.5.3 PDF文書内のフォント名

PDF文書に記載されたフォント名を取得します。記載されたとおりに取得しますので、フォントを代替する場合の名称として利用できます。(「6.15 代替フォント指定」参照)

メソッド

```
int SimpleFontListLength()  
[ string, int ] SimpleFontListGetName(int number)  
FontNameInfo SimpleFontListGetNameC()
```

引数

number	フォント名に Imager-LP が検索順に付加した番号
fontName	検索されたフォントの名称

戻り値(C#)

SimpleFontListLength は失敗すると負数のエラーコードをもどします。それ以外は検索されたフォント名の総数を戻します。SimpleFontListGetName 指定された番号のフォント名と共に埋め込み・非埋め込みなどを示すフラグのリストを戻します。SimpleFontListGetNameC 検索されたフォント名と共に埋め込み・非埋め込みなどを示すフラグが格納されたクラスの配列インスタンスを戻します。

6.5.4 インターラクティブ・フォーム

PDF文書に記載されたインターラクティブ・フォームの情報を取得します。インターラクティブ・フォームの詳細は「PDF Reference 1.7」 8.6Interactive Form を参照してください。

情報はPDF文書に記載された順に取得しますのでPDFのページに表示された見た目の順番とは違いますのでご注意ください。取得されるデータの詳細は、「4.5.6 インターラクティブ・フォーム」を参照してください。

PDF文書に記載されたインターラクティブフォームを以下のメソッドで取得します。まず InteractiveFormFirstData を使って最初に記載されたフォームのデータを取得します。次に、InteractiveFormNextData を使って残りフォームのデータを取得します。FormData の type が Undefined(0) になるまで繰り返しデータを取得することですべてのデータを取得できます。

メソッド

```
int InteractiveFormFirstData()  
int InteractiveFormNextData()
```

引数

ありません

戻り値

FormData クラス(または MlpFormData 構造体)が戻ります。
インターラクティブフォームが見つかった場合は、type 要素に1以上の値が戻り、0(ゼロ)の場合はフォームがそれ以上なく、負数の場合はエラーです。

6.6 画像ファイルに変換する関数群

JPEG形式の画像およびPNG形式の画像の場合は、PDF文書の1つのページを1つの画像に変換します。TIFF形式の画像の場合は、PDF文書の1つのページを1つの画像に変換することや、PDF文書の複数のページを1つの画像ファイルに変換することができます。以下は、画像変換用の関数名および変換できる画像形式です。

関数名	変換できる画像形式
CreatePict	PNG、JPEG、TIFF、BMP
PageToPict	PNG、JPEG、TIFF、BMP
ConvertToTiff	TIFF
CreateTiffRange	TIFF
CreateTiffMulti	TIFF
RangeToTiff	TIFF
MultiPageToTiff	TIFF

6.6.1 指定された1ページを画像に変換(解像度と画像品質を指定しない)

PDF文書の指定されたページを画像に変換します。

解像度と画像品質は、既定値(デフォルト値)または画像の変換に先立って指定された値が使用されます。画像の解像度や品質を指定する場合は、SetPictureResolution、SetPictureQuality関数などを利用して下さい。

ページ番号の指定では、最初のページを1とします。なお、ページ番号0はPDF文書の先頭ページ、負数のページ番号は最終ページとみなされます。

変換する画像ファイルの拡張子で画像の形式が自動で判断されます。

メソッド

```
int CreatePict(int pageNumber, string outputPath)
```

引数

pageNumber	画像に変換するPDF文書のページ番号(先頭のページ番号は1です。)
outputFilePath	画像のファイルパス 画像形式はファイルの拡張子によって以下のように自動で選択されます。 拡張子が“png”の場合、PNG形式の画像 拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像 拡張子が“tiff”または“tif”の場合、TIFF形式の画像 拡張子が“bmp”の場合、BMP形式の画像

戻り値

成功すると0(ゼロ)が戻り、それ以外の場合はエラーコードが戻ります。

6.6.2 指定された1ページを画像に変換(解像度と画像品質を指定する)

PDF文書の指定されたページを画像に変換します。

画像に変換する際に、指定された解像度および画像品質を使用します。ページ番号は、先頭のページを1とします。

ページ番号に負数を指定しますと、PDF文書の最終ページが画像に変換されます。

変換する画像ファイルの拡張子で画像の形式が判断されます。

メソッド

```
int PageToPict(int pageNumber, int dpi, int quality, string outputPath)
```

引数

pageNumber	画像に変換するPDF文書のページ番号(先頭のページ番号は、1です。)
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstant.MIN_RESOLUTION</code> 以上かつ <code>MlpConstant.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstant.MIN_RESOLUTION</code> または <code>MlpConstant.MAX_RESOLUTION</code> に正規化されます。
quality	JPEG 圧縮の画像品質を <code>MlpConstant.MIN_QUALITY</code> 以上かつ <code>MlpConstant.MAX_QUALITY</code> までの数値で指定します。範囲外の値を指定すると <code>MlpConstant.MIN_QUALITY</code> または <code>MlpConstant.MAX_QUALITY</code> に正規化されます。
outputFilePath	PNG形式画像を指定した場合は無視されます。 画像のファイルパス 画像形式はファイルの拡張子によって以下のように自動で選択されます。 拡張子が“png”の場合、PNG形式の画像 拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像 拡張子が“tiff”または“tif”の場合、TIFF形式の画像 拡張子が“bmp”の場合、BMP形式の画像

戻り値

成功すると0(ゼロ)が戻り、それ以外の場合はエラーコードが戻ります。

6.6.3 すべてのページを TIFF 形式画像に変換

PDF文書のすべてのページを1つのTIFF形式画像ファイルに変換します。

メソッド

```
int ConvertToTiff (string outputPath)
```

引数

outputFilePath	画像のファイルパス TIFF形式の画像が作成されます。拡張子は「.tiff」を指定します。
----------------	--

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.6.4 ページ範囲を指定して TIFF 形式画像に変換

指定された範囲のPDFページをひとつのTIFF画像に変換します。(指定された範囲内のページをすべて含む1つのTIFF画像に変換されます)。ただし、PDF文書にないページを指定すると失敗します。

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

ページ番号0を指定すると最初のページ(ページ番号1)、ページ番号が負数の場合は最終ページとみなされます。したがって、開始ページ=1 終了ページ=-1 と指定すると、最初のページから最後のページまですべてのページをTIFF画像に変換します。

ページは逆順(大きいページ番号から小さいページ番号の順)でも指定できます。

メソッド

```
int CreateTiffRange(  
    int startPageNumber, int endPageNumber, string outputPath  
)
```

引数

startPageNumber	画像に変換する最初のPDF文書のページ番号(先頭のページ番号は、1です。)
endPageNumber	画像に変換する最後のPDF文書のページ番号
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.6.5 ページのリストを指定して TIFF 形式画像に変換

リストで指定されたPDF文書のページをひとつのTIFF画像に変換します。

画像に変換するページを区切り文字(スペース、タブまたはコンマ)で区切ってページを指定すると、指定した順に画像に変換された複数ページで構成されたTIFF画像が生成されます。このとき、PDF文書に無いページを指定するとその指定は無視されます。成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

ページの指定の詳細は、「9.0 文字列でのページ指定方法」を参照してください。

メソッド

```
int CreateTiffMulti(string pageList, string outputPath)
```

引数

pageList	画像に変換するページのリスト。画像に変換するページを区切り文字で区切って1つ以上のページを指定します。PDF文書にないページは無視されます。 ページ番号 0 は最初のページ、ページ番号 -1 は最後のページに変換されます。 詳細は、「9.0 文字列でのページ指定方法」を参照してください。
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.6.6 ページ範囲を指定して TIFF 形式画像に変換(解像度を指定する)

指定された範囲のPDFページをひとつのTIFF画像に変換します。指定された範囲内のページをすべて含む1つのTIFF画像に変換されます。ただし、PDF文書にないページを指定すると失敗します。

ページ番号0を指定すると最初のページ(ページ番号1)、ページ番号が負数の場合は最終ページとみなされます。したがって、開始ページ=1 終了ページ=-1 と指定すると、最初のページから最後のページまですべてのページをTIFF画像に変換します。

ページは逆順(大きいページから小さいページ番号の順)でも指定できます。

メソッド

```
int RangeToTiff(  
    int startPageNumber, int endPageNumber, int dpi, string outputPath  
)  
int RangeToTiff(  
    int startPageNumber, int endPageNumber, string outputPath  
)
```

引数

startPageNumber	画像に変換する最初のPDF文書のページ番号(最初のページは、1です。)
endPageNumber	画像に変換する最後のPDF文書のページ番号(最後のページを、-1 と指定できます。)
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstant.MIN_RESOLUTION</code> 以上かつ <code>MlpConstant.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstant.MIN_RESOLUTION</code> または <code>MlpConstant.MAX_RESOLUTION</code> に正規化されます。 (省略可能)
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.6.7 ページのリストを指定して TIFF 形式画像に変換(解像度を指定する)

リストで指定されたPDFページをひとつのTIFF画像に変換します。

画像に変換するページを区切り文字(スペース、タブまたはコンマ)で区切ってページを指定すると、指定した順に画像に変換された複数ページで構成されたTIFF画像が生成されます。このとき、PDF文書に無いページを指定するとその指定は無視されます。

ページの指定の詳細は、「9.0 文字列でのページ指定方法」を参照してください。

メソッド

```
int MultiPageToTiff(string pageList, int dpi=0, string outputPath)  
int MultiPageToTiff(string pageList, string outputPath)
```

引数

pageList	画像に変換するページのリスト。画像に変換するページを区切り文字で区切って1つ以上のページを指定します。PDF文書にないページは無視されます。 ページ番号 0 は最初のページ、ページ番号-1 は最後のページに変換されます。 詳細は、「9.0 文字列でのページ指定方法」を参照してください。
dpi	生成される画像の 1 インチあたりの画素数(生成される画像の解像度)を <code>MlpConstant.MIN_RESOLUTION</code> 以上かつ <code>MlpConstant.MAX_RESOLUTION</code> 以下で指定します。範囲外の値を指定すると <code>MlpConstant.MIN_RESOLUTION</code> または <code>MlpConstant.MAX_RESOLUTION</code> に正規化されます。 既定値は <code>MlpConstant.DEFAULT_RESOLUTION</code> です。
outputFilePath	画像のファイルパス TIFF形式の画像のみが作成されます。拡張子は「.tiff」を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.7 画像(ピクセル)データに変換する関数群

Imager-LPは画像形式への変換ばかりではなく、ピクセルデータへも変換しますが、Python開発環境では利用できません。

6.8 出力画像の色空間を指定する

PDF文書がカラーで構成されている場合でも、グレースケール、白黒ディザ、白黒(2階調)画像に変換します。
色空間の指定を省略すると、RGBカラーの画像が生成されます。

6.8.1 RGB カラー画像を指定

PDF文書をRGBカラーの画像に変換するよう設定します。

PDF文書がRGBカラーでない場合でも、画像データはRGBカラー(1画素あたり24ビットで構成された画像データ)になります。

RGBカラー画像指定は、既定値です。

メソッド

```
int SetPictureRGB()  
int SetPicture(MlpConstant.MLP_PICTURE_RGB)
```

引数

SetPictureRGB には、ありません。

SetPicture では、`MlpConstant.PICTURE_RGB` を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.8.2 CMYK カラー画像を指定

PDF文書をCMYKカラーの画像に変換するよう設定します。

メソッド

```
int SetPictureCMYK
```

引数

ありません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.8.3 グレースケール画像を指定

PDF文書をグレースケールの画像に変換するよう設定します。

メソッド

```
int SetPictureGray()  
int SetPicture(MlpConstant.PICTURE_GRAY)
```

引数

SetPictureGray には、ありません。

SetPicture では、`MlpConstant.PICTURE_GRAY` を指定します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.8.4 白黒ディザ画像を指定

PDF文書を白黒ディザ (Floyd-Steinberg ディザリング) の画像に変換するよう設定します。

メソッド

```
int SetPictureDither()  
int SetPicture(MlpConstant.PICTURE_DITHER)
```

引数

SetPictureDither には、ありません。

SetPicture では、MlpConstant.PICTURE_DITHER を指定します。

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.8.5 白黒2階調画像を指定

PDF文書を白黒2階調の画像に変換するよう設定します。

メソッド

```
int SetPictureBW()  
int SetPicture(MlpConstant.PICTURE_BW)
```

引数

SetPictureBW には、ありません。

SetPicture では、MlpConstant.PICTURE_BW を指定します。

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.9 カラープロファイル

Imager-LP はRGBカラー画像に変換する場合に、PDF文書に指定された CMYK 色空間を RGB 色空間に変換します。既定では、この変換を計算式に基づいて実施しますが、カラープロファイルを用いた変換を指定することができます。

6.9.1 カラープロファイルの指定

カラープロファイルを指定します。この指定をせずにカラープロファイルによる変換を指定すると、既定のカラープロファイルが利用されます。UnuseColorProfile メソッドによってカラープロファイルの利用を解除できます。

カラープロファイルを使って色空間を変更する場合は SetColorConversion メソッドで変換手順を変更してください。

メソッド

```
int SetRGBProfile(string rgbpfn)           /*RGBカラープロファイル指定*/  
int SetCMYKProfile(string cmykpfn)        /*CMYKカラープロファイル指定*/  
int UnuseColorProfile()                   /*カラープロファイルの使用を解除*/
```

引数

rgbpfn	RGBカラープロファイルのファイル名
cmykpfn	CMYKカラープロファイルのファイル名

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

注意

現在利用可能なカラープロファイルは以下のとおりです。

RSWOP.icm (Agfa: Swop Standard) : 既定のCMYKカラープロファイル
JapanColor2001Coated.icc (Japan Color 2001 Coated)
JapanColor2001Uncoated.icc (Japan Color 2001 Uncoated)
JapanColor2002Newspaper.icc (Japan Color 2002 Newspaper)
JapanWebCoated.icc (Japan Web Coated (Ad))
USWebCoatedSWOP.icc (U.S. Web Coated (SWOP) v2)
USWebUncoated.icc (U.S. Web Uncoated v2)
sRGB Color Space Profile.icm (sRGB IEC61966-2.1) : 既定のRGBカラープロファイル
AdobeRGB1998.icc (Adobe RGB (1998))
AppleRGB.icc (Apple RGB)

6.9.2 色空間変換手順指定

CMYK色空間からRGB色空間への変換を指定します。

メソッド

```
int SetColorConversion(int direction, int flag)
```

引数

direction	CMYK色空間からRGB色空間へ変換するように以下を指定します。 <code>MlpConstant.CONVERT_CMYK_TO_RGB</code>
flag	色空間の変換手順を指定します。以下のいずれかを指定します。 <code>MlpConstant.CONVERT_BY_COLOR_PROFILE</code> カラープロファイルを使った変換を指定 <code>MlpConstant.CONVERT_FAST</code> 計算による変換を指定 (既定の手順) <code>MlpConstant.CONVERT_MOST_FAST</code> 簡易計算による変換を指定

6.10 画像のサイズ

PDF 文書では、1 ポイントを 1/72 インチとしています。PDF Imager-LP は、この 1 ポイントを指定された画像解像度に変換して画像を生成します。そのため、大きな値の画像解像度を指定すると大きなピクセルサイズの画像が生成され、小さな画像解像度では小さなピクセルサイズの画像が生成されます。

キャンバスとは、画像を配置する領域です。生成された画像は、このキャンバスに貼り付けられます。キャンバスの領域が生成された画像よりも小さい場合は画像の一部が切り取られ、逆に大きな領域を指定すると余白の付加された画像になります。

生成された画像は、画像の解像度から算出された画像の論理サイズを持っています。論理サイズを指定のサイズに設定できます、がこの指定によって画像のピクセルサイズは変わりません。論理サイズの既定値は、PDF 文書に記載されたページの大きさです。

6.10.1 ページの大きさ

PDF 文書では、ページごとにその大きさを持っています。さらに、このページの大きさ(詳細は、『PDF Reference』を参照してください。)を表すために5種類の大きさ(MediaBox、CropBox、BleedBox、TrimBox、ArtBox)が用意されています。PDF Imager-LP は、ページの大きさを決定する際の境界を指定できます。

メソッド

```
int SetPicture(int opt)
```

引数

opt	利用する大きさの種類を指定します。 <code>MlpConstant.PICTURE_USE_MEDIABOX</code> MediaBox で指定された大きさを使います。(既定値) <code>MlpConstant.PICTURE_USE_CROPBOX</code> CropBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、MediaBox を使います。 <code>MlpConstant.PICTURE_USE_BLEEDBOX</code> BleedBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。 <code>MlpConstant.PICTURE_USE_TRIMBOX</code> TrimBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。 <code>MlpConstant.PICTURE_USE_ARTBOX</code> ArtBox で指定された大きさを使います。 PDF 文書に指定されていない場合は、CropBox を使います。
-----	--

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.10.2 画像解像度の設定

PDF文書を画像に変換する際の 1 ポイントを指定のピクセル数で構成するよう指定します。このときに指定する数値を画像解像度(または、解像度)といいます。既定の解像度は、150DPI(Dot Per Inch)です。

解像度は、X方向、Y方向で同じ値が設定されます。設定する値が大きい場合は、処理に必要なメモリが多くなり、処理時間も増大します。

既定では 150DPI で画像が生成されます。

メソッド

```
int SetPictureResolution(int dpi)
int SetPicture(MlpConstant.RESOLUTION_DPI, int dpi)
```

引数

dpi	解像度を DPI(dot per inch) 単位で指定します。 解 像 度 は 、 <code>MlpConstant.MIN_RESOLUTION</code> 以 上 か つ <code>MlpConstant.MAX_RESOLUTION</code> 以下を指定します。範囲外の値を指定する と 、 <code>MlpConstant.MIN_RESOLUTION</code> ま た は <code>MlpConstant.MAX_RESOLUTION</code> に正規化されます。
-----	---

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.10.3 画像のピクセルサイズを取得

指定された解像度で生成される画像のピクセルサイズを取得します。

メソッド

```
BoundingBox GetPicturePixel(int pageNum)
```

引数

pageNum	サイズを取得するページの番号
<code>BoundingBox.x1</code>	画像の幅(ピクセル単位)
<code>BoundingBox.y1</code>	画像の高さ(ピクセル単位)

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

なお、PDF文書では、ページごとにその物理的な大きさが”MediaBox”(PDF Reference 3.6.2 Page Tree 参照)として記載されています。PDF文書での解像度 72DPI を指定して所得する画像のピクセルサイズは、この”Media Box”のサイズです(既定の場合)。ページサイズの種類は変更できます。「6.10.1 ページの大きさ」を参照してください。

6.10.4 画像幅のピクセルサイズを取得

指定された解像度で生成される画像幅のピクセルサイズを取得します。

メソッド

```
int GetPicturePixelX(int pageNum)
```

引数

pageNum	サイズを取得するページの番号
---------	----------------

戻り値

成功すると画像の幅がピクセル数で戻り、失敗するとエラーコード(負数)が戻ります。

なお、PDF文書では、ページごとにその物理的な大きさが”MediaBox”(PDF Reference 3.6.2 Page Tree 参照)として記載されています。PDF文書での解像度 72DPI を指定して所得する画像のピクセルサイズは、この”Media Box”のサイズです(既定の場合)。ページサイズの種類は変更できます。「6.10.1 ページの大きさ」を参照してください。

6.10.6 画像のピクセルサイズ指定

生成される画像の大きさをピクセル単位で指定します。ただし、指定できるのは幅(X方向)または、高さ(Y方向)のみです。

メソッド

```
int SetPicturePixelX(int x)  
int SetPicturePixelY(int y)
```

引数

x	画像の幅をピクセル単位で指定します。 0(ゼロ)を指定すると、既定の大きさになります。 高さ(Y方向)は、自動で計算されます。
y	画像の高さをピクセル単位で指定します。 0(ゼロ)を指定すると、既定の大きさになります。 幅(X方向)は、自動で計算されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.10.7 キャンバスサイズの設定

PDF 文書から変換された画像を配置する仮想の領域(キャンバス)のサイズを指定します。

キャンバスサイズを変更しても画像の解像度は変化しません。そのため、変換された画像よりも小さな領域を指定すると画像が切り取られ、大きな領域を指定すると画像に余白が付加されます。

生成された画像は、キャンバスの左上に配置され、余白は白色で塗りつぶされます。

指定を省略すると、生成される画像のピクセルサイズとキャンバスのサイズは同じになります。

メソッド

```
int SetCanvasSize(int x, int y)
```

引数

x	キャンバスの幅をピクセル単位で指定します。 0(ゼロ)を指定すると、変換された画像の幅と同じになります。
y	キャンバスの高さをピクセル単位で指定します。 0(ゼロ)を指定すると、変換された画像の高さと同じになります。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.10.8 キャンバスサイズ変更の原点

PDF 文書から変換された画像のキャンバスサイズを変更(「6.10.7 キャンバスサイズの変更」参照)する際の原点位置を指定します。

原点を指定することで切り取または余白が追加される画像の辺を限定できます。

指定を省略すると、原点は画像の中心になります。そのため、画像の四辺で切り取りまたは余白の追加が実施されます。なお、原点が中央の場合に上下左右の辺の切り取りピクセル数または追加される余白のピクセル数は必ずしも同じではありません。

メソッド

```
int SetCanvasOrigin(int org)
```

引数

org	キャンバスサイズを変更する際の原点を指定します。 0(ゼロまたは <code>MlpConstant.CORIGIN_CENTER</code>): 画像の中心を原点とします。 画像の四辺が変更対象になります。 1(または <code>MlpConstant.CORIGIN_LEFT_TOP</code>): 画像の左上を原点とします。 画像の右辺、下辺のみが変更対象になります。 2(または <code>MlpConstant.CORIGIN_LEFT_BOTTOM</code>): 画像の左下を原点とします。 画像の右辺、上辺のみが変更対象になります。 3(または <code>MlpConstant.CORIGIN_RIGHT_TOP</code>): 画像の右上を原点とします。 画像の左辺、下辺のみが変更対象になります。 4(または <code>MlpConstant.CORIGIN_RIGHT_BOTTOM</code>): 画像の右下を原点とします。 画像の左辺、上辺のみが変更対象になります。
-----	--

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.10.9 画像の論理サイズ指定

PDF Imager-LP で生成する画像データ(TIFF 形式および JPEG 形式画像)は、そのピクセルサイズとは別に、論理的な大きさを持っています。論理的な大きさは、画像データのプロパティに含まれる解像度と画像のピクセルサイズから算出されます。PDF Imager-LP では、この算出された画像の大きさを画像の論理サイズといいます。

次の関数では、この論理サイズを指定します。ただし、記載される解像度の丸め誤差によって、生成された画像の論理サイズにも誤差が生じます。そのため、画像の形式にもよりますが、画像への変換時に指定した論理サイズと、生成された画像の論理サイズに差異が生じます。

メソッド

```
int SetPictureSize(int x, int y, int unit)
```

引数

x	論理サイズの幅を指定します。
y	論理サイズの高さを指定します。
unit	指定した論理サイズの単位を指定します。 以下の値を指定します。

<code>MlpConstant.PICTURE_UNIT_PERCENT</code>	値をパーセントで指定
<code>MlpConstant.PICTURE_UNIT_INCH</code>	値をインチで指定
<code>MlpConstant.PICTURE_UNIT_MM</code>	値をミリメートルで指定

論理サイズは、幅もしくは高さのいずれかのみを設定できます。両方を指定した場合は、幅が優先されます。高さを指定する場合は、幅を0(ゼロ)にします。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

注意

指定のサイズから算出される論理的な画像解像度が小さすぎる場合に正しい画像データが生成されない場合があります。

6.10.10 ページの境界ボックスのサイズを取得

Imager-LP は PDF 文書に記述されたMedia box、Crop box、Bleed box、Trim boxおよびArt box の値を取得できますが、Python 開発環境では取得できません。

6.10.11 ページの回転角を取得

PDF 文書では、ページが回転されている場合があります。

以下のメソッドで、その回転角を取得します。

なお、回転角は0°、90°、180°、270°のいずれかです。

メソッド

```
int GetPageRotation(int pageNum, out int rotate)
```

引数

pageNum	値を取得するページの番号を指定します。
rotate	回転の角度を取得します。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.11 TIFF 形式画像の圧縮指定

TIFF 画像を生成する場合の圧縮方法を指定します。

6.11.1 圧縮しない

TIFF 形式画像を生成する場合に圧縮をしません。

メソッド

```
int SetTiffCompress (MlpConstant.TIFF_COMPRESS_NONE)  
int SetPicture (MlpConstant.COMPRESS_NONE)
```

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.11.2 JPEG 圧縮指定

TIFF 形式画像を生成する場合に JPEG 圧縮をします。

2 番目の指定形式では、JPEG 圧縮の品質を同時に指定します。ここで指定する品質は、画像形式に JPEG を選択した場合にも有効になります。既定の解像度は 75% です。

メソッド

```
int SetTiffCompress (MlpConstant.COMPRESS_JPEG)  
int SetPicture (MlpConstant.COMPRESS_JPEG, int quality)
```

引数

quality JPEG 圧縮画像の品質を `MlpConstant.MIN_QUALITY` 以上かつ `MlpConstant.MAX_QUALITY` 以下で指定します。
範囲外の値を指定した場合は、`MlpConstant.MIN_QUALITY` または `MlpConstant.MAX_QUALITY` に正規化されます。

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.11.3 Deflate 圧縮指定

TIFF 形式画像を生成する場合に Deflate 圧縮をします。

メソッド

```
int SetTiffCompress (MlpConstant.TIFF_COMPRESS_DEFLATE)  
int SetPicture (MlpConstant.COMPRESS_DEFLATE)
```

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.11.4 LZW 圧縮指定

TIFF形式画像を生成する場合にLZW圧縮をします。

メソッド

```
int SetTiffCompress(MlpConstant.TIFF_COMPRESS_LZW)
int SetPicture(MlpConstant.COMPRESS_LZW)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.12 JPEG 圧縮の品質指定

JPEG圧縮が指定された場合の画像品質を指定します。画像データの圧縮が指定されない場合やJPEG圧縮以外の圧縮が指定された場合は、この指定は無視されます。

メソッド

```
int SetPictureQuality(int quality)
int SetPicture(MlpConstant.JPEG_QUALITY, int quality)
```

引数

quality	JPEG圧縮の際の品質を <code>MlpConstant.MIN_QUALITY</code> 以上かつ <code>MlpConstant.MAX_QUALITY</code> 以下の値で指定します。範囲外の値を指定すると、 <code>MlpConstant.MIN_QUALITY</code> または <code>MlpConstant.MAX_QUALITY</code> に正規化されます。
---------	---

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.13 BMP画像の形式を指定する

BMP画像を生成する際に Windows 形式または OS/2形式を指定できます。

メソッド

```
int SetPicture(MlpConstant.BITMAP_IMG_FORMAT, int format)
```

引数

format	0(Windows 形式) または、1(OS/2 形式)を指定します。
--------	-------------------------------------

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.14 変換の詳細を指定する

6.14.1 塗りつぶしパスのアンチエイリアスを抑制する

PDF Imager-LP は、パスや文字描画の際にその境界を滑らかにするため、アンチエイリアス処理を施しています。そのため、小さなパスを組み合わせて大きなパス画像にする場合などで、その下地となる画像や文字が透けてしまう場合があります。このような場合は、塗りつぶしパスのアンチエイリアスを抑制して下地が透けて見えるのを防ぐことができます。

なお既定では、塗りつぶしパスのアンチエイリアスは実施します。

メソッド

```
int SetFillPathAADisable()  
int SetPicture(MlpConstant.FILL_PATH_AA, 0)
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.14.2 塗りつぶしパスのアンチエイリアスを実施する

塗りつぶしパスのアンチエイリアスを抑制すると、それ以降の変換で抑制されますので、以下でその抑制を中止します。

メソッド

```
int SetFillPathAAEnable();  
int SetPicture(MlpConstant.FILL_PATH_AA, 1);
```

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.14.3 文字コード0(ゼロ)の表示

PDF Imager-LP は、表示文字のコードが0(ゼロ)の場合には、未定義グリフ⁶の代わりにスペースが表示されます。

この動作を以下の手順で変更およびその設定状態を取得できます。

メソッド

```
int SetCid0GlyphMode(int mode) /*設定*/  
int SetPicture(MlpConstant.MLP_CID_0_GLYPH, int mode) /*設定*/
```

引数

mode	以下のグリフ表示モードを指定します。
	<code>MlpConstant.HOW_NOTDEF_GLYPH</code> 未定義グリフを表示
	<code>MlpConstant.REPLACE_TO_BLANK_GLYPH</code> 空白(Space)グリフを表示(既定)
	<code>MlpConstant.IGNORE_CID_0_GLYPH</code> CID=0 の文字表示を無視します。

戻り値

設定の場合は、成功すると0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

取得の場合は、成功すると設定されている値が戻り、失敗するとエラーコードが戻ります。

⁶ フォントなどの文字コードに対するグリフ(字形)が未定義の場合に表示されるグリフ

6.14.4 極細い線の描画

PDF Imager-LP はPDF文書を指定された解像度で画像に変換するため、解像度に比較して極端に細い線を期待したとおりに変換できないことがあります。このような場合に、その細い線をより太い線で代替描画することを指定します。

以下の関数で、描画する最小の太さ(線幅)を指定します。これにより、指定された線幅より小さな線幅は最小線幅に代替して描画されます。

メソッド

```
int SetPicture(MlpConstant.STROKEPATH_MINIMUM_LINE_WIDTH, float width)
```

引数

width 最小の線幅をポイント(1 ポイント=1/72 インチ)単位で指定します。
0(ゼロ)または負数を指定すると、最小線幅の指定がなくなります。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

ご注意ください。

この関数で変更できる線は、PDFコマンドのパスストロークによって描画される線です。
パスストロークの詳細は、「PDF Reference」を参照してください。

6.14.5 PDFページの境界指定

PDFのページ境界には種類があります。PDF Imager-LPはページ境界のひとつであるMediaBoxに従って画像に変換します。それとは違う業界を使って画像に変換するためには以下の設定が必要です。

メソッド

```
int SetPicture(int command)
```

引数

command	以下のいずれかを指定します。	
<code>MlpConstant.PICTURE_USE_MEDIABOX</code>		MediaBox をページ境界とします。(既定)
<code>MlpConstant.PICTURE_USE_CROPBOX</code>		CropBox をページ境界とします。
<code>MlpConstant.PICTURE_USE_BLEEDBOX</code>		BleedBox をページ境界とします。
<code>MlpConstant.PICTURE_USE_TRIMBOX</code>		TrimBox をページ境界とします。
<code>MlpConstant.PICTURE_USE_ARTBOX</code>		ArtBox をページ境界とします。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

6.14.6 PDFのバージョン指定

PDF Imager-LP はPDF1.4 からPDF1.7 までおよびPDF2.0(一部)規格の文書を画像に変換できます。しかし、これ以外の規格のPDF文書であっても可能な限り画像に変換しようとしています。規格外のPDF文書を変換しない設定にできますが、Python 開発環境では指定できません。

6.15 代替フォント指定

PDF Imager-LP はPDF文書に指定されたフォントを別のフォントに替えて画像に変換することができます。

指定フォントを代替する場合は、PDF文書に指定されているフォント名を使わなければなりません。このフォント名を取得するには、「6.5.3 PDF文書内のフォント名」を利用します。

ご注意

PDF文書を画像に変換した後に代替フォントを指定しなおして同じページを画像に変換しても新たに指定したフォントに代替されません。同じページを別のフォントに代替する場合はライブラリを終了する手順を実行してください。

6.15.1 既定フォントの指定

PDF文書に指定されたフォントや代替指定されたフォントが検索できない場合のフォントを指定します。

既定では、「MS ゴシック」または「MS 明朝」フォントを使って画像変換されます。

メソッド

```
int SetDefaultFont(string defaultFont)
```

引数

defaultFont 既定で使用するフォントの名前

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

6.15.2 代替フォントの指定

PDF文書に指定されたフォントでそれが埋め込まれていない場合は、そのフォントに替えて使用するフォントを指定できます。

メソッド

```
int SetAlternateFont(string origFont, string altFont, bool always)  
int SetAlternateFont2(  
    string origFont, string altFont, string weight, string style,  
    bool always, bool replace)
```

引数

origFont	PDF文書に指定されたフォントの名前
altFont	代替するフォントの名前
weight	代替するフォントの太さ “normal”、“bold”または、“100”、“200”、“300”、“400”、“500”、“600”、“700”、“800”、“900”のいずれかを指定します。なお、数値の300以下は“normal”と同じで、400以上は“bold”と同じになります。
style	“normal”、“italic”または、“oblique”のいずれかを指定します。なお、“italic”と“oblique”は同じです。
always	真(True)を指定すると、origFont フォントがシステムにインストールされている場合でも altFont で代替されます。偽(False)を指定すると、origFont フォントがシステムにインストールされている場合は代替されません。
replace	真(True)を指定すると、origFont フォントがPDF文書に埋め込まれている場合でも altFont で代替されます。偽(False)を指定すると、origFont フォントがPDF文書に埋め込まれている場合は代替されません。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

6.16 OCG(レイヤー)

PDF文書は、レイヤーを持つことができます。PDF Imager-LP は、OCGを初期状態で画像に変換することや各レイヤーの表示・非表示を指定して画像に変換することができます。

6.16.1 OCG の総数取得

PDF文書に記載されたレイヤーの総数を取得します。

メソッド

```
int GetOcgCount ()
```

引数

ありません

戻り値

成功すると、0(ゼロ)以上の正数が戻ります。負数はエラーコードです。

6.16.2 OCG の状態取得

OCGのID、名前、階層、表示・非表示の状態を取得します。取得には、MlpOcgData クラスを使います。

C/C++開発環境で使用する場合は、MlpOcgData 型のポインターを取得しますが、このポインターは一時的なもので、保存して後で利用することはできません。また、このポインターを開放してはいけません。

```
class OcgData
{
    int          id;
    string       name;
    int          level;
    bool         visible;
}
```

メンバー

id	各OCGにつけられたユニークな認識番号
name	PDF文書で示されたname
level	OCGの階層(ゼロがトップレベル)、-1の場合はそのOCGは階層に含まれません。
visible	False:OCGが非表示に設定されている、True:表示に設定されている

メソッド

```
OcgData[] GetOcgData ()
```

引数

ありません

戻り値

成功すると OcgData の配列が戻ります。

6.16.3 OCG の表示・非表示指定

OCGの表示・非表示を指定します。設定には、MlpOcgData クラスを使います。詳細は、「6.15.2 OCGの状態」を参照してください。

メソッド

```
int SetOcgState(OcgData[] data)
```

引数

data	OCGのID、名前、階層、表示・非表示の状態を示す構造体の配列 ただし、id および visible 以外は無視されます。
------	--

戻り値

成功すると、0(ゼロ)以上の正数が戻ります。失敗すると、エラーコードが戻ります。

6.17 透かし(ウォーターマーク)

Imager-LP は変換した画像に透かし(ウォーターマーク)を追加描画できます。

透かしは、画像や文字列それぞれを最大1つページ画像に追加描画できます。描画するZ軸位置としてページ画像の前面もしくは背面を選択できます。画像および文字列共に指定された場合で同じZ軸位置が指定された場合は文字列がより前面に描画されます。

6.17.1 透かしの設定

透かしの内容を設定します。

透かしとして文字を描画する場合は、エスケープ文字列を使ってPDF文書のタイトルや作成日時などを描画できます(「4.17.4 透かしのエスケープ文字」)参照。

PDF文書から変換された画像の原点は左上です。また、Y軸は画像の上から下が正方向です。

メソッド

```
int SetImageWatermark(          /* 画像を描画指定 */
    string fileName, int x_position, int y_position, int z_position,
    float x_scale, float y_scale, float alpha
)
int SetFillTextWatermark(       /* 塗りつぶしたテキストを描画指定 */
    string text, string fontName, int x_position, int y_position,
    int z_position, float fontSize, string color, float alpha
)
int SetStrokeTextWatermark(     /* 枠線で描画されたテキストを描画指定 */
    string text, string fontName, int x_position, int y_position,
    int z_position, float fontSize, string color, float alpha
)
```

引数

fileName	透かしとして描画する画像のファイル名(パス名) 貼り付ける画像の原点は、左上です。
text	透かしとして描画する文字列
fontName	文字列のフォント名
x_position	透かしを描画するX座標
y_position	透かしを描画するY座標
z_position	透かしを描画するZ軸位置(前面または背面) 0: 透かしは描画されません。 正数: 透かしはページ画像の前面に描画されます。 負数: 透かしはページ画像の背面に描画されます。
x_scale	画像をX方向に拡大する倍率です。(1.0超は拡大、1.0未満は縮小です。)
y_scale	画像をY方向に拡大する倍率です。(1.0超は拡大、1.0未満は縮小です。)
fontSize	文字列のサイズ(単位はポイント;1ポイント=1/72インチ)
color	文字列の色を16進数または色名で指定します。(「4.17.3 文字色指定」参照)
alpha	アルファチャネル値(不透明度) 0.0 から 1.0 の値を指定します。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

6.17.2 透かしを削除

既に指定した透かしを削除します。

通常は、PDF文書を閉じる処理(「6.24 PDF文書処理の終了」参照)によって透かし(の描画情報)は削除されます。それ以前に削除する場合に以下を使います。削除後に変換されたページ画像には透かしが描画されません。

メソッド

```
int ClearWatermarkData()
```

引数

ありません。

戻り値

成功すると、0(ゼロ)が戻ります。失敗すると、エラーコードが戻ります。

6.18 コールバック関数

Python 開発環境では実装されていません。

6.19 変換された画像をパネルに描画

PDF Imager-LP は、その内部に特別な画像を生成できます。この画像を PDF Imager-LP では「パネル」と呼びます。

パネルにはPDF文書を変換したページ画像を1つ以上貼り付けることができます。さらに、ページ画像は別のPDF文書を変換した画像でも貼り付けられます。作成されたパネルは指定された画像形式で取得できます。

一般に以下の手順で複数のページ画像をパネルに貼り付けます。

1. ライブラリを初期化 (Initialize)
2. パネルを生成 (CreatePanel)
3. PDF文書を開く (OpenDoc)
4. 変換する画像の色空間や解像度などを指定
(SetPictureRGB、SetPictureResolution など)
5. PDF文書のページを指定して内部の画像データに変換 (CreatePictMem)
6. 3、4、5の手順を必要に応じて繰り返します。
7. パネルのデータを画像形式で取得 (PanelToFile)
8. PDF文書を閉じる (CloseDoc)
9. ライブラリを終了 (Uninitialize)

6.19.1 パネル作成

複数のページ画像を貼り付けられるパネルを生成します。生成されたパネルは不透明白色でクリアされています。

パネルはライブラリ内部に1つだけ作成されます。既に作成されている場合は、そのパネルを削除して新たなパネルが作成されます。作成されたパネルは、FreePictMem で削除できます。

メソッド

```
int CreatePanel(int width, int height, int colorspace)
```

引数

width	パネルの横方向のピクセル数
height	パネルの縦方向のピクセル数
colorspace	色 空 間 (MlpConstant.PICTURE_RGB または MlpConstant.PICTURE_GRAY)

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.19.2 パネルをクリア

複数のページ画像を貼り付けられるパネル全体を不透明白色で塗りつぶします。

メソッド

```
int ClearPanel();
```

引数

ありません

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.19.3 パネルにページ画像を貼り付ける

メモリー上のページ画像をパネルに貼り付けます。

メモリーにはあらかじめ MlpCreatePictMem を使ってページ画像を作成しておかなければなりません(メモリー上に画像がない場合などは貼り付けができません)。

メモリー上の画像はパネルの色空間と同じ色空間に自動で変換されます。ページ画像をパネルに貼り付ける際に画像のサイズや解像度は変更されません。ページ画像にアルファチャネルはありませんので、後から描画したページ画像でそれまでのページ画像に上書きされます。ページ画像の背景は不透明白色ですので注意してください。

パネル画像やページ画像の原点 (x=0, y=0) は、左上です。

メソッド

```
int MlpPictToPanel(int xPos, int yPos, float alpha)
```

引数

xPos	ページ画像を貼り付けるパネルの X 座標
yPos	ページ画像を貼り付けるパネルの Y 座標
alpha	貼り付けるページ画像の不透明度 (0.0 から 1.0 の間で指定)

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.19.4 パネルにページ画像の境界を指定して貼り付ける

メモリー上のページ画像の境界を指定してパネルに貼り付けます。境界を指定すると、画像を切り取ることなどができます。

指定された領域の内部がパネルに貼り付けられます。指定された領域にページ画像がない部分はパネルに描画されません。

メモリーにはあらかじめ CreatePictMem を使ってページ画像を作成しておかなければなりません(メモリー上に画像がない場合などは貼り付けができません)。

メモリー上の画像はパネルの色空間と同じ色空間に自動で変換されます。ページ画像をパネルに貼り付ける際に画像のサイズや解像度は変更されません。ページ画像にアルファチャネルはありませんので、後から描画したページ画像でそれまでのページ画像に上書きされます。ページ画像の背景は不透明白色ですので注意してください。

パネル画像やページ画像の原点 (x=0, y=0) は、左上です。

メソッド

```
int MlpMemPictToPanelBBox(  
    int xPos, int yPos, float alpha, int bboxX, int bboxY,  
    int bboxWidth, int bboxHeight  
)
```

引数

xPos	ページ画像を貼り付けるパネルの X 座標
yPos	ページ画像を貼り付けるパネルの Y 座標
alpha	貼り付けるページ画像の不透明度 (0.0 から 1.0 の間で指定)
bboxX	貼り付けるページ画像から切り取る領域の X 座標 (ページ画像上の座標)
bboxY	貼り付けるページ画像から切り取る領域の Y 座標 (ページ画像上の座標)
bboxWidth	貼り付けるページ画像から切り取る領域の幅 (ピクセル値)
bboxHeight	貼り付けるページ画像から切り取る領域の高さ (ピクセル値)

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.19.5 パネルに矩形を描画

パネルに矩形を描画します。

ページ画像を切り取った場合など、その領域を明示したい場合などに利用します。

パネル画像の原点 (x=0, y=0) は、左上です。矩形は不透明な指定色で描画されます。

メソッド

```
int PaintRectToPanel(  
    int xPos, int yPos, int width, int height, int lineWidth,  
    int color1=0, int color2=0, int color3=0, int color4=0)  
int PaintRectToPanel(  
    int xPos, int yPos, int width, int height, int lineWidth, string colorStr)
```

引数

xPos	矩形を描画するパネルの x 座標
yPos	矩形を描画するパネルの y 座標
width	描画する矩形の幅 (ピクセル値)
height	描画する矩形の高さ (ピクセル値)
lineWidth	矩形の線幅 (ピクセル値)
color	矩形線の色配列 (カラーの場合はRGBの3色、グレースケールの場合は1色) 各色は0～255の値で指定 色指定を省略した場合は黒色
colorStr	色名

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.19.6 パネルのデータを画像で取得

パネルのデータを画像に変換しファイルで取得します。

メソッド

```
int PanelToFile(string fileName)
```

引数

fileName	画像ファイル名 画像形式はファイルの拡張子によって以下のように自動で選択されます。 拡張子が“png”の場合、PNG形式の画像 拡張子が“jpeg”または“jpg”の場合、JPEG形式の画像 拡張子が“tiff”または“tif”の場合、TIFF形式の画像
----------	---

戻り値

成功すると、0 (ゼロ) が戻り、失敗するとエラーコードが戻ります。

6.19.7 パネル削除

作成されたパネルを削除し、そのメモリー領域を開放します。

パネル領域は、ライブラリの使用終了で自動的に削除されるためメモリー領域に問題がない場合は開放しなくともかまいません。

メソッド

```
int FreePanel()
```

引数

ありません

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.20 画像変換の並列処理

PDF Imager-LP は、複数ページを画像に変換する場合に、それぞれをスレッドで並列に処理します。また、各ページを画像に変換する場合でも、そのページをより小さな領域に分割して並列処理します。

規定では、単一のページを分割してスレッドで並列に処理し、ページごとには並列処理しません。なお、これらの動作は抑制することができ、実行するスレッドの最大数を指定できます。

6.20.1 複数ページの並列処理の抑制

複数ページを画像変換する場合にスレッドによる並列処理を抑制します。

メソッド

```
int SetPicture(MlpConstant.THREAD_EACH_PAGE, int flag)
```

引数

flag 0(ゼロ)を指定すると、並列処理は抑制されます。それ以外の場合は抑制されません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.20.2 複数ページの並列処理の最大スレッド数

複数ページを画像変換する場合の最大スレッド数(並列処理数)を変更します。

メソッド

```
int SetPicture(MlpConstant.THREAD_EACH_PAGE_COUNT, int num)
```

引数

num 実行されるスレッドの総数を指定します。省略した場合は、使用しているコンピュータの論理CPU数と同じ数が使用されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.20.3 単一ページの並列処理の抑制

各ページを画像変換する場合の並列処理を抑制します。

メソッド

```
int SetPicture(MlpConstant.THREAD_SPLIT_PAGE, int flag)
```

引数

flag 0(ゼロ)を指定すると、並列処理は抑制されます。それ以外の場合は抑制されません。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.20.4 単一ページの並列処理の最大スレッド数

各ページを画像変換する場合の並列処理の最大スレッド数(並列処理数)を変更します。

メソッド

```
int SetPicture(MlpConstant.THREAD_SPLIT_PAGE_COUNT, int num)
```

引数

num 実行されるスレッドの総数を指定します。省略した場合は、使用しているコンピュータの論理CPU数と同じ数が使用されます。

戻り値

成功すると、0(ゼロ)が戻り、失敗するとエラーコードが戻ります。

6.21 PDF文書処理の終了

PDF文書の画像変換処理を終了します。

メソッド

```
void CloseDoc()
```

引数

ありません。

戻り値

ありません。

6.22 ライブラリの使用終了

ライブラリの使用を終了します。

メソッド

```
void Uninitialize()
```

引数

ありません。

戻り値

ありません。

7.0 PDF 文書の構成要素抽出関数

PDF Imager-LP にはPDF文書からデータを直接抽出できる関数が試験的に追加されています。
ここに記載された関数群は将来その動作が変更される場合があります。利用には注意をお願いします。

各関数には、コンソールに出力、文字列として抽出、オブジェクトハンドルとして戻すものがあります。なお、C/C++開発環境で利用する場合は接頭辞Mlpを加えた名称に読み替えてください。Python開発環境でも同様に利用できます。

- ・コンソールに出力する関数

Show...(...)

- ・文字列を戻す関数

String...(...)

string を戻します。(C/C++では char**を戻します。)

- ・ObjectHandle を戻す関数

Object...(...)

ObjectHandle クラスのメソッドが利用ます。(C/C++ではハンドルとしての意味を持ちます。)

7.1 現在文書の xref

現在オープンしているPDF文書から xref 情報を抽出します。

xref の詳細は PDF Reference, version1.7 の「3.4.3 Cross-Reference Table」を参照してください。

```
void ShowXref() //コンソールに出力
int StringXref(out string data) //文字列を戻す
int GetXrefOffset() //xref のオフセット値
```

戻り値

int 型の0(ゼロ)または正数は成功です。それ以外はエラーコードです。

7.2 現在文書の trailer

現在オープンしているPDF文書から trailer 情報を抽出します。

trailer の詳細は PDF Reference, version1.7 の「3.4.4 File Trailer」を参照してください。

```
int ShowTrailer(bool pretty) //コンソールに出力
int StringTrailer(bool pretty, out string data) //文字列を戻す
ObjectHandle StringTrailer()
```

引数

pretty PDF オブジェクトを表示する際に改行などで見やすくします。

戻り値

int 型の0(ゼロ)は成功です。それ以外はエラーコードです。

ObjectHandle を戻す場合では、失敗すると不明な型(UNKNOWN_OBJ 型)が戻ります。

7.3 現在文書の catalog

現在オープンしているPDF文書から catalog 情報を抽出します。

catalog の詳細は PDF Reference, version1.7 の「3.6.1 Document Catalog」を参照してください。

```
int ShowCatalog(bool pretty); //コンソールに出力
int StringCatalog(bool pretty, out string data); //文字列を戻す
```

7.4 現在文書の document information

現在オープンしているPDF文書から document information 情報を抽出します。

document information の詳細は PDF Reference, version1.7 の「10.2.1 Document Information Dictionary」を参照してください。

```
int ShowDocumentInformation(bool pretty);
int StringDocumentInformation (bool pretty, out string data);
```


8.0 初期化ファイル(初期化データ)について

PDF Imager-LP は初期化ファイル(または初期化データ)を使うことでPDF文書に指定されたフォント⁷(埋め込み・非埋め込みフォントにかかわらず)を指定したフォントに置き換えて変換することができます。

初期化ファイル(または初期化データ)はXML形式です。

初期化ファイル(または初期化データ)の指定手順は、「4.3 初期化ファイル(初期化データ)を指定する」を参照してください。

8.1 非埋め込みフォントの代替指定

PDF文書で使用しているフォントがその文書に埋め込まれていない場合は、システムにインストールされている同じ名前のフォントを検索して利用します。もし、必要なフォントがシステムで検索できない場合は、「MS ゴシック」または「MS 明朝」フォントで代替します。

属性値 `always=true` を指定すると、PDF文書で指定されたフォントがシステムにインストールされている場合でも代替指定されたフォントが使用されます。なお、`always` の既定値は `false`(偽)です。省略すると、システムにインストールされたフォントが使用されます。

代替フォントは関数で指定することもできます。「4.12 代替フォント指定」を参照してください。

この指定ファイルまたはデータの指定は、「4.3 初期ファイル(初期化データ)を指定する」を参照してください。

フォントの指定は、以下のようにXML形式で指定します。

```
<?xml version="1.0" encoding="shift_jis" ?>
<init>
  <fonts default="Font0">
    <font org="Font11" alt="Font12" />
    <font org="Font21" alt="Font22" />
    <font always="true" org="Font31" alt="Font32" />
  </fonts>
</init>
```

属性

<code>org</code>	PDF文書に指定されたフォント名指定
<code>alt</code>	代替するフォント名指定
<code>always</code>	値に <code>true</code> (真)を指定すると、 <code>org</code> で指定されたフォントは <code>alt</code> で指定されたフォントに代替されます。 <code>false</code> (偽)を指定すると、 <code>org</code> で指定されたフォントがシステムにインストールされていない場合にのみ <code>alt</code> で指定されたフォントに代替されます。 既定値は、 <code>false</code>
<code>default</code>	指定のフォントがシステムにインストールされていない場合に既定で利用されるフォント名 既定値は「MS ゴシック」および「MS 明朝」

値

<code>Font0</code>	PDF文書で指定されたフォントや代替フォントが検索できない場合に利用されるフォント名
<code>Font11</code> , <code>Font21</code>	PDF文書で指定されたフォントのPostScript名 属性「 <code>org</code> 」で指定します。
<code>Font12</code> , <code>Font22</code>	代替するシステムにインストールされたフォント名 属性「 <code>alt</code> 」で指定します。
<code>Font31</code> は <code>Font32</code>	常に代替されます。ただし、 <code>Font32</code> がシステムにインストールされている場合に

⁷ Type0 フォントに限ります。詳細は、PDF Reference を参照してください。

限ります。

代替フォントは<init>およびの子要素として指定します。

この初期化ファイルはShift_JIS文字コードで記述してください。それ以外の文字コードは利用できません。

8.2 埋め込みフォントの代替指定

PDF Imager-LP はPDF文書にフォントが埋め込まれている場合でもフォントを代替指定できます。代替フォントは関数で指定することもできます。「4.15 代替フォント指定」を参照してください。

この指定ファイルまたはデータの指定は、「4.3 初期ファイル(初期化データ)を指定する」を参照してください。

フォントの指定は、以下のようにXML形式で指定します。

```
<?xml version="1.0" encoding="shift_jis" ?>
<init>
  <font default="Font0">
    <font replace="true" org="Font11" alt="Font12" />
    <font replace="true" org="Font21" alt="Font22" />
    <font always="true" replace="true" org="Font31" alt="Font32" />
  </font>
</init>
```

属性

org	PDF文書に指定されたフォント名指定
alt	代替するフォント名指定
replace	値に true(真)を指定すると、org で指定されたPDF文書の埋め込みフォントが利用されなくなります。
always	値に false(偽)を指定すると、org で指定されたフォントがシステムにインストールされているフォントに代替使用されます。システムにインストールされていない場合は alt に指定されたフォントが使用されます。 true(真)を指定すると、org で指定されたフォントに替わって alt で指定されたフォントが使用されます。 既定値は、false です。
replace	値に true(真)を指定すると、org フォントがPDF文書に埋め込まれている場合でも always が真であれば alt フォントに代替され、always が偽であれば org フォントがシステムにインストールされていない場合に alt フォントに代替されます。 false(偽)を指定すると、org で指定されたフォントがPDF文書に埋め込まれている場合代替されません。 既定値は、false
default	指定のフォントがシステムにインストールされていない場合に既定で利用されるフォント名です。既定値は「MS ゴシック」および「MS 明朝」です。

値

Font0	PDF文書で指定されたフォントや代替フォントが検索できない場合に利用されるフォント名
Font11、Font21	PDF文書で指定されたフォントのPostScript名 属性「org」で指定します。
Font12、Font22	代替するシステムにインストールされたフォント名 属性「alt」で指定します。
Font31	Font31 がPDF文書に埋め込まれている場合でも Font32 に代替されます。なお、always を false (偽)とするとFont31 がシステムにインストールされている場合にはインストールされているフォントが使

用されます。

代替フォントは<init>およびの子要素として指定します。

この初期化ファイルはShift_JIS文字コードで記述してください。それ以外の文字コードは利用できません。

8.3 代替フォントの太さとスタイル

初期化ファイルまたは初期化データでフォントを代替する場合は、指定するフォントの太さ(weight)やスタイル(style)を指定できます。ただし、代替のフォントが日本語フォントの場合のみです。

代替フォントは関数で指定することもできます。「4.15 代替フォント指定」を参照してください。

この指定ファイルまたはデータの指定は、「4.3 初期ファイル(初期化データ)を指定する」を参照してください。

フォントの指定は、以下のようにXML形式で指定します。

```
<?xml version="1.0" encoding="shift_jis" ?>
<init>
  <font>
    <font org="Font11" alt="Font12" weight="bold" />
    <font org="Font21" alt="Font22" weight="normal" style="italic" />
  </font>
</init>
```

属性

weight	代替フォントの太さ指定 値は、normal、bold または 100、200、300、400、500、600、700、800、900 のいずれかを指定できます。 既定値は、normal です。この属性は省略できます。
style	代替フォントのスタイル指定 値は、normal、italic、oblique のいずれかを指定できます。 なお、italic と oblique は同じ斜体です。 既定値は、normal です。この属性は省略できます。

この初期化ファイルはShift_JIS文字コードで記述してください。それ以外の文字コードは利用できません。

9.0 文字列でのページ指定方法

複数のページで構成されたTIFF画像に変換する場合、ページを文字列で指定できます（「4.6 画像ファイルに変換する関数群」参照）。ページの指定方法は、以下のとおりです。

複数のページを指定する場合の区切り文字は、スペース、タブまたはコンマのいずれかです。

PDF文書に存在しないページ番号は、無視されます。

最初のページをページ番号「0」と指定できます。

最後のページをページ番号「-1」と指定できます。

連続したページをマイナス文字（“-”）で区切って初めのページ番号、終わりのページ番号の順に指定します。

指定の例：

"0,3,5,7"	ページ番号 1, 3, 5, 7 をこの順で画像に変換します。0 は、ページ番号「1」に変換されます。
"2,5,3,9,-1"	ページ番号 2, 5, 3, 9 及び最終のページをこの順で画像に変換します。5ページで構成された画像が作成されます。
"2,2,3,3"	ページ番号 2 を2回そしてページ番号 3 を2回この順で画像に変換します。4ページで構成された画像が作成されます。
"1,3-7,9"	ページ番号 1, 3, 4, 5, 6, 7, 9 をこの順で画像に変換します。
"7-4,10"	ページ番号 7, 6, 5, 4, 10 をこの順で画像に変換します。
"1-3,6-5"	ページ番号 1, 2, 3, 6, 5 をこの順で画像に変換します。

上記の例を組み合わせてページを指定できます。

いずれの場合でも、PDF文書に存在しないページ番号は無視されます（単独で現れる「0」と「-1」を除きます）。

10.0 著作権

本書「PDF Imager-LP 説明書」は株式会社トラスト・ソフトウェア・システムの著作物です。他媒体への転載を禁止します。