Antenna House PDF Viewer SDK V5 / PDF Viewer API インタフェース仕様

目次

1. 概要

本書では「PDF Viewer API」のインタフェース仕様について説明します。「PDF Viewer API」は PDF を表示するための C++/C/.NET/.NET Framework 向けのライブラリです。

2. 動作環境

2.1. 対応 OS

対応 OS	Windows 11 / 10　64 ビット 日本語版 Windows Server 2022 / 2019 / 2016　日本語版

※　Microsoft Visual C++ 2019 再頒布可能パッケージが必要です。

※　.NET Framework 4.6.2 以降が必要です。（.NET Framework 利用の場合）

※　.NET6.0 以降が必要です。（.NET6 利用の場合）Windows Server ではデスクトップエクスペリエンスが必要です。Server Core / Nano Server はサポートされません。

※　Microsoft 社のメインストリームサポートが終了したプラットフォームはサポート対象外です。

2.2. 仮想環境

仮想環境（VMware や Hyper-V など）で実行する場合、実環境上と同じ動作が保証されているのであれば機能制限はありません。

3. 制限事項

3.1. 対象 PDF ファイル

- PDF 仕様（PDF32000-2）に準拠した PDF1.3 ～ PDF2.0
- オープンできる PDF のファイルサイズは 2GB までとなります。
- ページ数は明示的な制限はしておりません。

3.2. レンダリング品質および性能について

- 対象 PDF であっても一部正しく表示できない項目があります。詳しくは「PDFViewerSDK 表示仕様.pdf」をご覧ください。
- PDF ファイルや実行環境など条件によっては、表示仕様に記載されていない未知の制限が存在する場合があります。
- **表示や画像出力や印刷のレンダリング品質や速度や消費メモリなどの性能について、特定の品質や性能を保証するものではありません。特に Adobe Acrobat/Adobe Reader を含む他社製品の PDF ビューアと同等の品質や性能を保証するものではありません。**
- 表示や画像出力や印刷の品質および性能は、実行環境やプリンタおよびプリンタドライバ、PDF ファイルのデータ、描画領域のサイズ、解像度、カラー、拡大縮小率、その他要因によって影響を受けます。条件が違う場合には、必ずしも同じ結果になりません。
- 表示と画像出力、印刷では異なる結果になる場合があります。
- レンダリング品質や性能について、実運用環境と同等の環境や出力方法、また実際に扱うものと同等の PDF データで導入前によくご確認ください。
- 下記の場合に表示や印刷が遅くなります。
  - 半透明やブレンドを含む PDF
  - 細かなパターンやパスを多数含む PDF
- 用紙サイズの大きな PDF ファイルや、PDF データ内に大きな画像を含む場合など、レンダリング処理中にメモリが不足した場合、ページ全体もしくはページの一部が描画されないことがあります。
- 32bit 版、とくに.NET Framework、Visual Basic のランタイムを併用する場合はメモリ不足に陥りやすいため、そのような場合は、64bit 版やネイティブアプリケーションをご検討ください。
- タスクスケジューラやサービス等、デスクトップが存在しない環境では、場合によって PDF が正しく印刷や画像出力されない可能性があります。

- PDF/A、PDF/X 等の PDF ファイルの表示は可能ですが、表示に関する要件をすべて満たしません。特に PDF のプロファイルの扱いは厳密には従っていないため、色が正確ではない場合があります。
- モノクロ二値のデバイスに表示・印刷した場合や、画像ファイル出力において、モノクロ（IMAGECOLOR_MONOCHROME）、ブラック（IMAGECOLOR_BLACK）を指定した場合は、十分なレンダリング品質が得られません。
- ピクセル単位での描画の正確性は保証しておりません。アンチエイリアスのかかり、線の幅の均一さなどご希望通りとならないことがあります。
- Direcdt2D を使用しない場合、Smoothing には GDI+が使用されますが、Micorosoft 社は GDI+のサービスでの利用をサポートしておりません。サービスでのご利用時は場合によって PDF が正しく印刷や画像出力されない可能性があります。Smoothing を OFF にしてご利用いただくことをお勧めいたします。

3.3. 複数プロセス/スレッドでの同時使用

- 複数のプロセスで API を同時に使用することは問題ありません。
- 同一プロセス内の複数のスレッドで API を同時に使用することは、次の例外を除き問題ありません。
- 複数スレッドでの印刷はできません。PDFDocument クラスの下記の API は、複数スレッドでの同時使用には非対応です。
    - startPrint() / startPrintExt() / startPrintExt2()
    - endPrint()
    - abortPrint()
    - printPage()

3.4. .NET / .NET Framework からの利用

- 画面表示や印刷は Win32 GDI を利用します。ディスプレィやプリンタのデバイスコンテキストなどの準備は Win32 API を利用して上位アプリケーションで行う必要があります。P/Invoke 経由で行うサンプル(ApiDotNetCSharp)を添付しております。
- System.Drawing、System.Printing 名前空間などの .NET / .NET Framework のグラフィックスや印刷フレームワークには対応しておりません。
- System.Pringing 名前空間の印刷機能と drawPage メソッドを使用して印刷する事は可能ですが、印刷速度が遅くなることが判明しておりますので推奨いたしません。

3.5. その他

・　文字列検索では、MediaBox や CropBox の指定に関わらず Box 外の文字列も検索にヒットします。

4. 開発環境

4.1. C++/C

- Microsoft Visual Studio2019　でビルドされています。
- 呼び出し側のプログラムは、互換性のあるコンパイラを使用してください。

検証済みの環境	Microsoft Visual Studio 2019 (Visual C++) Microsoft Visual Studio 2022 (Visual C++)

- バージョンが異なる Visual C++ からは std::stream を引数にもつ API は使用できません。ファイル名指定の API をご利用ください。
- wchar_t はビルトイン型と unsigned short 両方のメソッドを export しています。wchar_t の扱いにより、どちらかのメソッドが呼び出されます。
- Visual C++以外のコンパイラでは C インタフェースを使用してください。
- UWP/Windows ストアアプリ開発に対応しておりません。

4.2. .NET6 または.NET Framework

- .NET6.0 以上または、.NET Framework 4.6.2 以上に対応しています。
- Microsoft Visual Studio 2019、C++/CLI で開発されています。
- 動作にはネイティブ DLL が必要で、アーキテクチャ（x86 または x64）に依存します。
- 上位アプリケーションのアーキテクチャ（x86 または x64）にあわせ、32bit/64bit 版のバイナリを使用してください。
- ANY CPU は推奨しておりません。ご利用になる場合は正しい DLL がロードされるよう構成する必要があります。
- UWP/Windows ストアアプリ開発に対応しておりません。

4.3. インタフェース

4.3.1. C++/C

ネームスペース	AvsViewerSDK
ヘッダファイル	PDFViewerSDK.h
ライブラリファイル	［32bit 版］ lib¥Win32¥AvsPDFViewerSDK.lib ［64bit 版］ lib¥x64¥AvsPDFViewerSDK.lib

※ヘッダファイルに定義されていても、本ドキュメントに記載のない API はサポートされません。

4.3.2. .Net6

ネームスペース	AvsDotNetViewerSDK
DLL	［32bit 版］ bin¥Win32¥AvsPDFViewerSDK50Net60.dll ［64bit 版］ bin¥x64¥AvsPDFViewerSDK50Net60.dll

※定義されていても、本ドキュメントに記載のない API はサポートされません。

4.3.3. .Net Framework

ネームスペース	AvsDotNetViewerSDK
DLL	［32bit 版］ bin¥Win32¥AvsDotNetViewerSDK.dll ［64bit 版］ bin¥x64¥AvsDotNetViewerSDK.dll

※定義されていても、本ドキュメントに記載のない API はサポートされません。

4.4. 実行に必要なファイル

アプリケーションに応じて 32bit 版または 64bit 版をご利用ください。

- アプリケーション本体(exe ファイル)と同じフォルダに置いてください。
- 32bit 版及び 64bit 版のファイル名が同じであるため、ご注意ください。

アーキテクチャ	必要になる DLL やリソース
32bit 版	bin¥Win32 フォルダに含まれる下記ファイル。 ・ AvsPDFViewerSDK50.dll ・ AvsDotNetGuiCtl.dll (.NET Framework の場合) ・ AvsPDFViewerSDK50Net60.dll (.NET 6 の場合) ・ Ijwhost.dll (.NET 6 の場合) ・ icudt55.dll ・ icuin55.dll ・ icuuc55.dll ・ JapanColor2001Coated.icc
64bit 版	bin¥x64 フォルダに含まれる下記ファイル。 ・ AvsPDFViewerSDK50.dll ・ AvsDotNetGuiCtl.dll (.NET Framework の場合) ・ AvsPDFViewerSDK50Net60.dll (.NET 6 の場合) ・ Ijwhost.dll (.NET 6 の場合) ・ icudt55.dll ・ icuin55.dll ・ icuuc55.dll ・ JapanColor2001Coated.icc

・ 動作には別途「Visual Studio 2015、2017、2019 および 2022 用 Microsoft Visual C++ 再頒布可能パッケージ」のインストールが必要です。

以下の URL からダウンロードしてインストールしてください

https://support.microsoft.com/ja-jp/help/2977003/the-latest-supported-visual-c-downloads

[32bit 版]　vc_redist.x86.exe

[64bit 版]　vc_redist.x64.exe

5. 機能

PDF Viewer API には次の機能があります。

カテゴリ	機能
PDF のオープン・クローズ	・ PDF1.3～PDF1.7 ・ PDF2.0 (V5.0)
文書情報の取得	・ 作成日付、更新日付 ・ タイトル、作成者、サブタイトル、キーワード、アプリケーション、PDF 変換 ・ PDF バージョン
アクセス権限情報の取得	・ 内容のコピーが可能か ・ 印刷可能か
ページ	・ ページ数の取得 ・ ページサイズの取得 ・ ページ表示の回転
アプリケーションウインドウへの PDF の描画	・ 文字、線、画像のスムージング：印刷にも適用可 ・ 全てのカラースペースに対応 ・ カラープロファイルの設定（RGB、CMYK、グレースケール、出力）
印刷	・ ページを用紙に合わせる ・ 印刷時の向き、サイズ、印刷位置の調整 ・ ダイレクト印刷（印刷ダイアログを表示しない）
画像化	・ BMP、PNG、JPEG、 TIFF　形式 ・ ページ単位、ページ内矩形領域を画像に変換
検索	・ ページ数単位　全文検索　検索位置の取得 ・ ヒットしたテキスト矩形の取得
テキスト抽出	・ 矩形内のテキスト、ページ単位、全テキスト
注釈	・ Link 注釈の矩形情報、リンク先 URL の取得
表示モード	・ グラフィックエンジン切り替え：Direct2D(V5.0)/GDI*/GDI

	・ GDI+時のスムージング設定 ・ リモートデスクトップ表示モード ・ 注釈表示の切り替え（V5.0） ・ PDF Tool API 閲覧制限 PDF の制限表示（V5.0）
ウォーターマーク	・ ウォーターマークの設定 ・ テキスト、フォントファミリ、ウェイト、スタイル、不透明度
ページの指定範囲の情報取得（V3.5）	・ パス情報 ・ テキスト情報

6. ご利用上の注意

6.1. 文字列

- 扱う文字列は全て UNICODE となります。

6.2. 座標系

- 扱う座標系は、表示上の左上を原点とし、 x 座標は右に向かって正、 y 座標は下に向かって正の整数となります。
- 単位は TWIP です。(1TWIP＝1/20 ポイント＝1/1440 インチ)

6.3. レンダリングモード

6.3.1. Direct2D

- Direct2D を使用して高速な描画を行います。既定では Direct2D を使用します。
- 画面表示・画像出力が対象です。
- Direct2D 利用時でも印刷は従来どおり GDI/GDI+を使用して行います。

6.3.2. GDI+

- Direct2D を使用しない場合、Gdiplus.dll を使用して Smoothing を行うことができます。
- Gdiplus の初期化は行われていなければ SDK が行います。アプリケーションレベルで初期化した場合は SDK 終了後シャットダウンしてください。
- 画像出力、印刷でも使用できます。

注意事項)

- Micorosoft 社は Gdiplus のサービスでの利用はサポートしておりません。
- Smoothing をすべて off にした場合でも、印刷時のシェーディングの描画にのみ Gdiplus が使用されます。

6.4. 32bit 版／64bit 版

- 32bit 版の場合、状況によってメモリ不足で画像出力関数（makeXXXPage）等がエラーで失敗する場合があります。その場合は LAA オプションや 64bit 版を検討ください。
- 64bit 版ではより容量の大きな PDF を処理できます。ただし、既定でプロセスのヒープの上限が 500MB となっておりますので、setLoadMemoryLimit 関数を使用して、上限を設定してください。（大きくし過ぎますと、オープン直後にバックグランドスレッドで PDF ページの解析するページ数も増えるため、動作が遅くなります。）

6.5. エラー処理について

6.5.1. API 呼び出し時の結果判定

- API 呼出の結果判定は、API の戻り値(bool)で行ってください。

  例）openDocument、makeBmpPage 等
- 失敗した場合、エラー番号を取得して原因について確認してください。

6.5.2. エラーについて

- エラーには「致命的エラー」と「致命的でないエラー（警告）」があります。
- 致命的かそうでないかに関わらず、複数のエラーが発生する場合があります。

  例）複数の箇所にフォーマット不正があるなどの場合
- 成功の場合でも「致命的でないエラー（警告）」が発生することがあります。
- 発生したエラーは廃棄しない限り、PDFDocument に蓄積されます。
- getErrorCode() / getErrorMessage()/isFatalError() は蓄積された最初のエラーについての情報を返します。
- 蓄積されたエラーについて確認するには、エラーがなくなるまで、エラー情報の確認と abandonError()でのエラー廃棄を繰り返します。
- ApiPdfPrint サンプル CheckError 関数にエラーチェックの例がございます。

6.6. .NET6 および.NET Framework インタフェース

- PDF Viewer API は C++で開発されたネイティブ DLL を利用しています。このため、32bit・64bit のアーキテクチャに依存します。開発するアプリケーションのプラットフォームは「x64」または「x86」をご利用ください。「ANY CPU」は推奨しません。
- PDF Viewer SDK の DLL を参照するとき、ビルド対象のプラットフォームが”x86”であるか”x64”であるかにあわせて、それぞれ 32bit 版 DLL、64bit 版 DLL を参照する必要があります。Visual Studio の .NET6/.NET Framework プロジェクト上で、プラットフォームに合わせて別の DLL 参照を指定することはできません。このため、例えば次のような方法で切り替える必要があります。

  【方法 1】 プロジェクトを “x86”、”x64”の両方用意し、各プロジェクトでは、それぞれターゲットに応じた DLL を参照します。ソリューションファイルでは、”x86”、”x64”ターゲットにあわせて、追加するプロジェクトを x86 版、x64 版と切り替えるよう構成します。この方法の利点は、プロジェクトファイルを直接編集する必要がなく、Visual Studio 上で編集できることです。

  【方法 2】プロジェクトの DLL 参照を手動で編集する方法です。C#の場合、”.csproj”ファイルにおける参照設定を Include 属性でターゲットにあわせてそれぞれ指定します。

```
    <Reference Include="AvsDotNetViewerSDK">
      <HintPath Condition=" '$(Platform)' == 'x86' ">..¥..¥Windows-
VS2019¥Win32¥Release¥AvsDotNetViewerSDK.dll</HintPath>
      <HintPath Condition=" '$(Platform)' == 'x64' ">..¥..¥Windows-
VS2019¥x64¥Release¥AvsDotNetViewerSDK.dll</HintPath>
    </Reference>
```

  ここでは HintPath で条件指定を行い切り替えていますが、ItemGroup で切り替えることも可能です。

  この方法の利点は x86、x64 それぞれプロジェクトを用意する必要がない点です。一方手動で編集するため、間違いやすい、Visual Sutido で不意に編集を行うと編集が上書きされてしまう等のデメリットがあります。プロジェクトファイルに詳しくなければ 1 つめの方法をお勧めいたします。SDK 添付のサンプルプロジェクトではこちらの方法を使用しています。

  参考) https://docs.microsoft.com/ja-jp/visualstudio/ide/how-to-configure-projects-to-target-platforms?view=vs-2019

6.7. C インタフェース

C のインタフェースは、C++インタフェースの PDFDocument クラスを呼び出すためのものです。

- PDFDocument クラスの作成は PdfvCiCreateViewerDocument()で行い、削除は PdfvCiDeleteViewerDocument()で行います。
- PDFDocument クラスのメソッド呼び出しは、メソッド名に PdfvCi を付け、メソッド名の先頭を大文字にした関数を用意してあります。1 番目のパラメータに PdfvCiCreateViewerDocument()で作成した HPDFVIEWER を指定してください。

  ※一部、未対応の API がございます。
- 機能は PDFDocument クラスの対応する C++インタフェースを参照してください。
- C++インタフェースから以下の変更があります。
  - 変数の参照はポインタになります。
  - std::ostream&のパラメータは wchar_t*のファイル名渡しになります。
  - イメージ作成の戻り値は false がオープン失敗になります。

7. クラス

7.1. クラス名

クラス名は PDFDocument です。

7.2. メソッド

7.2.1. ストリームオープン

```
bool openDocument(std::istream& stream, const wchar_t* passWord=NULL, DWORD loadStart=0,
    const wchar_t* appName=NULL, const wchar_t* keyName=NULL, const wchar_t* value=NULL);
bool OpenDocument(Stream stream, String passWord, long loadStart, String appName, String keyName,
    String value); (.Net)
```

引数：

stream：PDF データストリーム

passWord：パスワード

loadStart：2 ページ目以降の解析開始までの時間(msec)。

appName：アプリケーション名

keyName：アプリケーション辞書キー名

value：アプリケーション辞書キー値

戻り値：

true 成功

false 失敗　失敗した場合、エラーコードを検査します。

エラーコード AvsViewerSDK::PDFDocument::ERROR_PASSWORD は、PDF にパスワードの設定があり、指定したパスワードが一致しないことを示します。パスワードの入力を行い、再度 openDocument()を呼び出します。

解説：

- ストリームを全て読み込みます。stream は、オープン終了後必要ありません。

- 1 ページ目のデータを解析し戻ります。2 ページ目以降は、スレッドを作成し loadStart 時間 Sleep()しページデータを解析します。loadStart が 0 の場合は、最初のページ表示 drawPage()が呼ばれたときにスレッドを作成しページデータの解析を開始します。
- 表示せず印刷を行う場合、loadStart を指定することで解析しながら、印刷できます。また、loadStart が 0 でも印刷時に解析を行います（同じスレッド）。
- 解析済みのページ数は getLoadCount()で取得することができます。
- 200 ページを超える PDF は 100 ページまで読み込み、残りは参照時に読み込みます。読み込んだページ数が 200 を超えた場合、古い参照ページから廃棄し、次回参照時に再度読み込みを行います。このページ数は setLoadPageCount()で変更できます。
- 廃棄されるのはページのデータのみで、ページから参照されるフォントや画像などは一度読み込まれると参照されるページが廃棄されても読み込まれたままとなります。
- アプリケーション名、キー名、キー値を指定すると、一致する Form XObject を非表示にします。

7.2.2. ファイルオープン

```
bool openDocument(const wchar_t* fileName, const wchar_t* passWord=NULL,
    DWORD loadStart=0,   const wchar_t* appName=NULL, const wchar_t* keyName=NULL,
    const wchar_t* value=NULL);
bool OpenDocument(String fileName, String passWord, long loadStart, String appName,
    String keyName, String value); (.Net)
```

引数：

fileName：PDF ファイル名

passWord：パスワード

loadStart：2 ページ目以降の読み込み開始までの時間(msec)。

appName：アプリケーション名

keyName：アプリケーション辞書キー名

value：アプリケーション辞書キー値

戻り値：

true 成功

false の場合はストリームオープンと同様にエラーを検査します。

解説：

- ファイルをオープンし、1 ページ目のデータを読み込み戻ります。
- 1 ページ目のデータを解析し戻ります。2 ページ目以降は、スレッドを作成し loadStart 時間 Sleep()しページデータを解析します。loadStart が 0 の場合は、最初のページ表示 drawPage()が呼ばれたときにスレッドを作成しページデータの解析を開始します。
- 表示せず印刷を行う場合、loadStart を指定することで解析しながら、印刷できます。また、loadStart が 0 でも印刷時に解析を行います（同じスレッド）。
- 解析済みのページ数は getLoadCount()で取得することができます。
- 最後のページの読み込みが終了するとファイルをクローズします。その間ファイルの変更はできません。
- 200 ページを超える PDF は 100 ページまで読み込み、残りは参照時に読み込みます。読み込んだページ数が 200 を超えた場合、古い参照ページから廃棄し、次回参照時に再度読み込みを行います。このページ数は、setLoadPageCount()で変更できます。
- 200 ページを超える文書の場合ファイルはオープンしたままになります。オープンしたままにしたくない場合はストリームオープンを使用してください。
- 廃棄されるのはページのデータのみで、ページから参照されるフォントや画像データは一度読み込まれると参照されるページが廃棄されても読み込まれたままとなります。
- アプリケーション名、キー名、キー値を指定すると、一致する Form XObject を非表示にします

7.2.3. クローズ

void closeDocument();

void CloseDocument(void); (.Net)

戻り値：

なし

7.2.4. 読み込みスレッド開始

void startLoadThread();

void StartLoadThread(void); (.Net)

解説：

後から読み込みスレッドを開始する場合に呼びます。openDocument()で loadStart に 0 を指定し、表示も行わない場合にこのメソッドで開始を指定できます。

既に読み込みスレッドがある場合は何もせず戻ります。

7.2.5. オープンされているか

bool isOpen() const;

property bool IsOpen　(.Net) 読み取りのみ

戻り値：

true オープン済み。

7.2.6. 内容のコピー可能か

bool isEnableCopyContents() const;

property bool IsEnableCopyContents　(.Net) 読み取りのみ

戻り値：

true コピー可能

解説：

PDF ファイルにコピー不可の指定があっても、イメージ作成機能（makeXXXPage）は動作します。PDF ファイルに従ったアクセス制御はアプリケーションの責任で行ってください。

7.2.7. 印刷可能か

bool isEnablePrintout() const;

property bool IsEnablePrintout　(.Net) 読み取りのみ

戻り値：

true 印刷可能

解説：

PDF ファイルが印刷不可の場合、印刷は行えません。

7.2.8. SDK バージョン番号の取得

static int getSDKVersion(wchar_t* buffer, int size);

property String SDKVersion

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.9. PDF バージョンの取得

```
int getVersion(wchar_t* buffer, int size) const;
property String Version　(.Net) 読み取りのみ
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.10.作成日付の取得

```
int getCreationDate(wchar_t* buffer, int size) const;
property String CreationDate　(.Net) 読み取りのみ
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.11.更新日付の取得

```
int getModifyDate(wchar_t* buffer, int size) const;
property String ModifyDate（.Net）読み取りのみ
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.12.タイトルの取得

```
int getTitle(wchar_t* buffer, int size) const;
property String Title（.Net）読み取りのみ
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.13.作成者の取得

```
int getAuthor(wchar_t* buffer, int size) const;
property String Author（.Net）読み取りのみ
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.14.サブタイトルの取得

```
int getSubject(wchar_t* buffer, int size) const;
```

property String Subject（.Net）読み取りのみ

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　（C++）

取得した文字列　（.NET）

7.2.15.キーワードの取得

int getKeywords(wchar_t* buffer, int size) const;

property String Keywords（.Net）読み取りのみ

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　（C++）

取得した文字列　（.NET）

7.2.16.アプリケーションの取得

int getCreator(wchar_t* buffer, int size) const;

property String Creator（.Net）読み取りのみ

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　（C++）

取得した文字列　（.NET）

7.2.17.PDF 変換の取得

int getProducer(wchar_t* buffer, int size) const;

property String Producer（.Net）読み取りのみ

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　（C++）

取得した文字列　（.NET）

7.2.18.ページ数の取得

int getPageCount() const;

property int PageCount（.Net）読み取りのみ

戻り値：

ページ数

7.2.19.解析済みページ数の取得

int getLoadCount() const;

property int LoadCount（.Net）読み取りのみ

戻り値：

解析済みページ数

解説：

内部で解析済みページ数を返します。このページはすぐに表示することができます。

解析済みでないページも表示できますが、表示時に解析を行います。

7.2.20.ページサイズの取得

SIZE getPageSize(int pageNo) const;

AvsSize GetPageSize(int pageNo);（.Net）

戻り値：

ページサイズ(単位：TWIP)

7.2.21.ページの表示

void drawPage(int pageNo, HDC hdc, const POINT* dispOrg, float scale, bool drawBg) const;

void DrawPage(int pageNo, IntPtr hdc, AvsPoint dispOrg, float scale, bool drawBg);　(.Net)

void drawScalingPage(int pageNo, HDC hdc, const POINT* dispOrg, float scaleH, float scaleV, bool drawBg) const;

void DrawScalingPage(int pageNo, IntPtr hdc, AvsPoint dispOrg, float scaleH, float scaleV, bool drawBg); (.Net)

void drawPage(int pageNo, HDC hdc, const POINT* dispOrg, float scale, bool drawBg, bool drawResult) const;

void DrawPage(int pageNo, IntPtr hdc, AvsPoint dispOrg, float scale, bool drawBg, bool drawResult); (.Net)

void drawScalingPage(int pageNo, HDC hdc, const POINT* dispOrg, float scaleH, float scaleV, bool drawBg, bool drawResult) const;

void DrawScalingPage(int pageNo, IntPtr hdc, AvsPoint dispOrg, float scaleH, float scaleV, bool drawBg, bool drawResult); (.Net)

引数：

pageNo：表示ページ番号(1 オリジン)

hdc：Window デバイスコンテキスト

dispOrg：表示オフセット（デバイス単位。例：デバイス解像度 600 dpi の場合、100 ドットは

デバイス上で 100/600 inch = 1/6 inch = 25.4/6 mm の計算になります）

scale：表示倍率(1.0f == 100%)

drawBg：true バックグラウンドも描画

scaleH：水平方向表示倍率(1.0f == 100%)

scaleV：垂直方向表示倍率(1.0f == 100%)

drawResult：true 検索結果を表示する

解説：

表示後スレッドを作成し、ページ読み込み処理を開始します。

印刷だけの目的では、スレッドは作成されません。

7.2.22.リモートデスクトップ表示モードの設定

void setRemoteDesktopDisplayMode(bool remote);

property bool RemoteDesktopDisplayMode (.Net)

引数：

remote：true を指定するとリモートデスクトップ用の表示を行う

解説：

リモートデスクトップでは GdiPlus で直接表示すると、きれいな描画を得られません。リモートデスクトップの場合、ビットマップを作成し、ディスプレイに転送することで、欠けることのない表示を行います。

既定値はリモートデスクトップ環境では true に設定されています。それ以外は false です。

drawPage()にディスプレイのデバイスコンテキストが指定された場合処理します。

その他のデバイスコンテキストには適用されません。

7.2.23.リモートデスクトップ表示モードか問い合わせる

bool isRemoteDesktopDisplayMode() const;

property bool RemoteDesktopDisplayMode (.Net)

戻り値：

リモートデスクトップ表示モードなら true

7.2.24.バッファ表示モードの設定

void setBufferDisplayMode(bool buffer);

property bool BufferDisplayMode (.Net)

引数：

buffer：true を指定するとバッファ表示を行う

解説：

表示イメージをビットマップにコピーし、再表示のときビットマップイメージを使います。

表示する領域が単純な矩形の場合にビットマップを作成します。他のウインドウが上にある場合は作成できません。縦スクロールが起きた場合はビットマップもスクロールしイメージを追加処理します。横スクロールが起きるとビットマップは無効になります。

ビットマップはページ毎に作成します。表示しなくなったページのビットマップは破棄されます。

既定値は true です。

drawPage()にディスプレイのデバイスコンテキストが指定された場合処理します。その他のデバイスコンテキストには適用されません。

リモートデスクトップ表示のときも有効です。

7.2.25.バッファ表示モードか問い合わせる

bool isBufferDisplayMode() const;

property bool BufferDisplayMode (.Net)

戻り値：

バッファ表示モードなら true

7.2.26.表示バッファの削除

void clearDisplayBuffer(HWND hWnd, int pageNo);

void ClearDisplayBuffer(IntPtr hWnd, int pageNo); (.Net)

引数：

hWnd：削除するウインドウ。NULL なら全てのウインドウ

pageNo：削除するページ。 0 なら全てのページ

解説：

上位アプリケーションで背景を描画し、背景を変更した場合などに呼び出します。

保存しているビットマップを無効化します。

7.2.27.スムージングが利用できるか問い合わせる

static bool isSmoothingAvailable();

property bool IsSmoothingAvailable (.Net)

戻り値：

スムージングが可能なら true

解説：

PDFViewerSDK では Gdiplus を利用してスムージング表示を行います。

7.2.28.スムージングの設定

bool setSmoothing(bool text, bool lineart, bool image);

void SetSmoothing(bool text, bool lineart, bool image); (.Net)

引数：

text：文字のスムージング(AntiAlias)を行う

lineart：線画のスムージングを行う

image：イメージのスムージングを行う

戻り値：

再表示が必要なら true

解説：

GDI/GDI+の場合に有効です。設定が無い場合のデフォルトは全て true です。

変更はいつでも行えます。設定変更し、再表示してください。

isSmoothingAvailable()が false の場合は無効です。

V3.0 から表示だけでなく、印刷、イメージ作成にも有効になりました。

7.2.29.スムージング設定の取得

void getSmoothing(bool& text, bool& lineart, bool& image);

void GetSmoothing(out bool text, out bol lineart, out bol image); (.Net)

引数：

text：文字のスムージング(AntiAlias)設定

lineart：線画のスムージング設定

image：イメージのスムージング設定

解説：

現在のスムージング設定を取得します。

7.2.30.文字表示の問い合わせ

bool isCharacterGDI() const;

property bool CharacterGDI (.Net)

戻り値：

文字で表示なら true 埋め込まれたフォントで表示する場合は false

7.2.31.文字表示の設定

void setCharacterGDI(bool character);

property bool CharacterGDI (.Net)

引数：

character：文字で表示する場合 true 埋め込まれたフォントで表示する場合は false

解説：

文字にフォントが埋め込まれている場合、文字で表示するか埋め込まれたフォントで表示するかを選択できます。

フォントによってはライセンスのため正しく表示できない場合があります。フォントの表示が正しくない場合、文字で表示するように設定します。

文字（コード）が無い場合は埋め込まれたフォントで表示します。

既定値は false です。

7.2.32.Direct2D を使用するか

bool isDirect2D() const;

property bool Direct2D (.Net)

戻り値：

Direct2D を使用する場合は true　使用しない場合は false

7.2.33.Direct2D 設定

void setDirect2D(bool direct2D);

property bool Direct2D (.Net)

引数：

Direct2D を使用する場合は true　使用しない場合は false

解説：

表示、画像出力に Direct2D を使用します。

true の場合でも、印刷には Direct2D は使用されません。

7.2.34.注釈表示の問い合わせ

bool getShowAnnotation() const;

property bool ShowAnnotation (.Net)

戻り値：

注釈を表示する場合は true　注釈を表示しない場合は false

7.2.35.注釈表示の設定

void setShowAnnotation(bool show);

property bool ShowAnnotation (.Net)

引数：

show：注釈を表示する場合は true　注釈を表示しない場合は false

解説：

外観ストリームを持つ注釈について、注釈の外観を表示するかを設定します。

外観ストリームを持たない注釈は設定値を true にしても表示されません。

PDF 表示や画像出力の場合には、注釈フラグの Hidden、NoView フラグが考慮されます。

印刷の場合には Hidden、Print フラグが考慮されます。

既定値は false です。

7.2.36.高速プレビューモードか

bool getFastPreviewMode() const;

property bool FastPreviewMode (.Net)

戻り値：

高速プレビューモードの場合は true　そうでない場合は false

7.2.37.高速プレビューモード設定

void setFastPreviewMode(bool preview);

property bool FastPreviewMode (.Net)

引数：

高速プレビューモードを使用する場合は true　使用しない場合は false

解説：

高速プレビューモードは、レンダリングは低品質ですが通常よりも高速にレンダリングするモードです。高速プレビューモードの設定は PDF をオープンする前に行ってください。オープン後の呼び出しは可能ですが、設定は反映されません。

制限事項：

- 時間のかかる処理を省くことで通常のモードよりは高速にレンダリングしますが、レンダリング結果の正確性は保証されません。
- レンダリング結果は改訂版を含むバージョン間で同一になることは保証されません。
- 現時点では通常のモードと大きな速度の差異はありません。

7.2.38.ウォータマークテキスト設定

void setShowAnnotation(bool show);

property String WatermarkText (.Net)

引数：

text：ウォータマーク文字列

解説：

ウォータマークとして出力する文字列を設定します。'¥n'で改行し複数行の文字列を設定することもできます。

7.2.39.ウォータマークテキスト取得

void setWatermarkText(const wchar_t* text);

property String WatermarkText (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数

buffer が NULL の場合は必要なバッファの文字数を返す

7.2.40.ウォータマークフォントファミリ設定

void setWatermarkFontFamily(const wchar_t* family);

property String WatermarkFontFamily (.Net)

引数：

family：ウォータマーク表示フォント名

7.2.41.ウォータマークフォントファミリ取得

int getWatermarkFontFamily(wchar_t* buffer, int size) const;

property String WatermarkFontFamily (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数

buffer が NULL の場合は必要なバッファの文字数を返す

7.2.42.ウォータマークフォントウエイト設定

void setWatermarkFontWeight(const wchar_t* weight);

property String WatermarkFontWeight (.Net)

引数：

weight：ウォータマークフォントウエイト

解説：

”700”、”bold”などを設定します。

7.2.43.ウォータマークフォントウエイト取得

int getWatermarkFontWeight(wchar_t* buffer, int size) const;

property String WatermarkFontWeight (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.44.ウォータマークフォントスタイル設定

void setWatermarkFontStyle(const wchar_t* style);

property String WatermarkFontStyle (.Net)

引数：

style：ウォータマーク表示フォントスタイル

解説：

”italic”を設定します。

7.2.45.ウォータマークフォントスタイル取得

int getWatermarkFontStyle(wchar_t* buffer, int size) const;

property String WatermarkFontStyle (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.46.ウォータマーク不透明度設定

void setWatermarkOpacity(const wchar_t* opacity);

property String WatermarkOpacity (.Net)

引数：

opacity：ウォータマーク不透明度

解説：

”0.3”,”50%”などの不透明度を設定します。

7.2.47.ウォータマーク不透明度取得

int getWatermarkOpacity(wchar_t* buffer, int size) const;

property String WatermarkOpacity (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。Buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列　(.NET)

7.2.48.印刷開始

bool startPrint(HDC hdc, const DEVMODEW* devmode, const DOCINFOW* docInfo,
bool paperSelect = true, bool scaling = false) const;

bool StartPrint(IntPtr hdc, IntPtr devmode, IntPtr docInfo,
bool paperSelect); (.Net)

bool StartPrint(IntPtr hdc, IntPtr devmode, IntPtr docInfo,
bool paperSelect, bool scaling);　(.Net)

引数：

hdc：プリンタデバイスコンテキスト

devmode：DEVMODE 構造体

docinfo：DOCINFO 構造体

paperSelect：true　PDF のページに合わせて用紙を自動で選択する

false　DEVMODE 構造体に設定された用紙サイズを使用する

scaling：true ページを用紙に合わせる

戻り値：

true ::StartDoc()に成功した。

解説：

::StartDoc()を呼び出します。成功した場合は、必ず endPrint()または abortPrint()を呼び出してください

7.2.49.印刷終了

```
void endPrint() const;
```

```
void EndPrint(void); (.Net)
```

解説：

::EndDoc()を呼び出し印刷を終了します

7.2.50.印刷中止

```
void abortPrint() const;
```

```
void AbortPrint(void); (.Net)
```

解説：

::AbortDoc()を呼び出し印刷を中止します

7.2.51.ページの印刷

```
void printPage(int pageNo, bool printMark) const;
```

```
void PrintPage(int pageNo, bool printMark); (.Net)
```

引数：

pageNo：印刷するページ番号(1 オリジン)

printMark：検索結果を印刷に反映する

解説：

::StartPage()から::EndPage()の処理を行います。

印刷を行うには startPrint()を呼び出し成功したならば、印刷したいページを指定し printPage()を繰り返し呼び出します。最後に endPage()で終了します。

サンプルプログラムも参照してください。

7.2.52.印刷開始（拡張）

```
bool startPrintExt(HDC hdc, const DEVMODEW* devmode, const DOCINFOW* docInfo,
        bool paperSelect = true, bool fit = false, float scale = 1.0f,
        PAPERALIGNH halign = PALIGNH_CENTER,
        PAPERALIGNV valign = PALIGNV_CENTER) const;
bool StartPrintExt(IntPtr hdc, IntPtr devmode, IntPtr docInfo,
        bool paperSelect, bool fit, float scale,
        PARENALIGNH halign, PARENALIGNV valign); (.Net)

bool startPrintExt2(HDC hdc, const DEVMODEW* devmode, const DOCINFOW* docInfo,
        bool paperSelect = true,
        PAGESCALING pagescaling = PAGESCALING_NONE, float scale = 1.0f,
        PAPERALIGNH halign = PALIGNH_CENTER,
        PAPERALIGNV valign = PALIGNV_CENTER) const;
bool StartPrintExt2(IntPtr hdc, IntPtr devmode, IntPtr docInfo,
        bool paperSelect, PAGESCALING pagescaling, float scale,
        PARENALIGNH halign, PARENALIGNV valign); (.Net)
```

引数：

hdc：プリンタデバイスコンテキスト

devmode：DEVMODE 構造体

docinfo：DOCINFO 構造体

paperSelect：true　PDF のページに合わせて用紙を自動で選択する

　　false　DEVMODE 構造体に設定された用紙サイズを使用する

fit：true ページを用紙に合わせる

scale：スケーリング倍率（fit の時は無効）

halign：水平配置

```
enum PAPERALIGNH {
        PALIGNH_LEFT,           // 左揃え
        PALIGNH_CENTER,         // 中央揃え
```

```
        PALIGNH_RIGHT           // 右揃え
    };
```

valign：垂直配置

```
    enum PAPERALIGNV {
        PALIGNV_TOP,            // 上揃え
        PALIGNV_CENTER, //  中央揃え
        PALIGNV_BOTTOM          // 下揃え
        };
```

pagescaling：ページの拡大／縮小

```
    enum PAGESCALING {
        PAGESCALING_NONE,       // なし
        PAGESCALING_FIT,        // 用紙に合わせる
        PAGESCALING_SHRINK      // 大きいページを縮小
    };
```

戻り値：

true ::StartDoc()に成功した。

解説：

::StartDoc()を呼び出します。startPrint()を参照してください。

アプリケーションが用紙を選択する場合 paperSelect=false とします。SDK は用紙方向のみを設定します。

PDF を用紙の印字領域に合わせて印刷する場合 fit=true とします。

scale を指定し用紙をはみ出す場合も halign,valign は有効です。左上、中央、右下を合わせることができます。

7.2.53.GdiPlus でイメージを操作するか問い合わせ

```
bool isGdiPlusRasterOperation() const;
```

property bool GdiPlusRasterOperation (.Net)

戻り値：

画像出力においてイメージの変換、縮小化に GdiPlus を使用する場合 true

7.2.54.GdiPlus でイメージを操作する設定

void setGdiPlusRasterOperation(bool bUse) const;

property bool GdiPlusRasterOperation (.Net)

戻り値：

bUse：画像出力においてイメージの変換、縮小化に GdiPlus を使用する場合 true。

既定値は false です。

7.2.55.BMP の作成

bool makeBmpPage(int pageNo, int dpi, const wchar_t* scale, int height,
IMAGE_COLOR color, const RECT& imageRect, std::ostream& stream, bool printMark)
const;

bool MakeBmpPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
AvsRect imageRect, Stream stream, bool printMark);

bool makeBmpPage(int pageNo, int dpi, const wchar_t* scale, int height,
IMAGE_COLOR color, const RECT& imageRect, const wchar_t* fileName, bool printMark)
const;

bool MakeBmpPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
AvsRect imageRect, String fileName, bool printMark); (.Net)

bool makeBmpPage(int pageNo, int dpi, const wchar_t* scale, int height,
IMAGE_COLOR color, const RECT& imageRect, IStreamPtr stream, bool printMark)
const;

※上記以外に、引数を省略した関数が利用できます

引数：

pageNo：BMP を作成するページ番号(1 オリジン)　※必須

dpi：スケーリング時の DPI 値(1 以上、1440 以下)　※必須。

1440dpi を超える場合 1440dpi で処理します。

scale：出力時のスケール文字列。100%以下、”1”==”100%”,”100px”は幅の指定　※必須

height：出力イメージの最大高さ(Pixel)、0 は高さ指定なし。※オプション（省略時 0）

dpi と scale から決定された高さが指定値を超える場合、指定値で出力します。

その場合の幅はページのアスペクト比から計算されます。

color：色数　※省略時　IMAGECOLOR_COLOR

```
enum IMAGE_COLOR {
        IMAGECOLOR_COLOR,           // フルカラー
        IMAGECOLOR_256COLOR,    // 256色カラー
        IMAGECOLOR_GRAYSCALE,  // グレースケール
        IMAGECOLOR_MONOCHROME,      // モノクローム
        IMAGECOLOR_BLACK            // テキスト、線画のハーフトーンなし
};
```

imageRect：イメージを作成するページ内の領域。空の矩形はページ全体

※省略時、空の矩形（ページ全体）

stream：BMP を出力するストリーム

printMark：検索結果をイメージに反映する　※必須

fileName：出力ファイル名

戻り値：

ファイルがオープンできない、出力に失敗した場合 false

注意：

std:ostream インタフェースは SDK と同じバージョンのコンパイラのみ利用できます。

7.2.56.JPEG の作成

```
bool makeJpegPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, int quality, bool progressive, const RECT& imageRect,
        std::ostream& stream, bool printMark) const;
bool MakeJpegPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
        int quality, bool progressive, AvsRect imageRect, Stream stream, bool printMark);
        (.Net)
```

```
bool makeJpegPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, int quality, bool progressive, const RECT& imageRect,
        const wchar_t* fileName, bool printMark) const;
bool MakeJpegPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
        int quality, bool progressive, AvsRect imageRect, String fileName, bool printMark);
        (.Net)

bool makeJpegPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, int quality, bool progressive, const RECT& imageRect,
        IStreamPtr stream, bool printMark) const;
```

※上記以外に、引数を省略した関数が利用できます

引数：

pageNo：JPEG を作成するページ番号(1 オリジン) ※必須

dpi：スケーリング時の DPI 値(1 以上、1440 以下)　※必須。1440dpi を超える場合 1440dpi で処理します。

scale：出力時のスケール文字列。100%以下、”1”==”100%”,”100px”は幅の指定　※必須

height：出力イメージの最大高さ(Pixel)　、0 は高さ指定なし。※オプション（省略時 0）

dpi と scale から決定された高さが指定値を超える場合、指定値で出力します。

その場合の幅はページのアスペクト比から計算されます。

※省略時 0

color：色数　※省略時　IMAGECOLOR_COLOR

```
enum IMAGE_COLOR {
        IMAGECOLOR_COLOR,               // フルカラー
        IMAGECOLOR_256COLOR,    // 256色カラー
        IMAGECOLOR_GRAYSCALE,  // グレースケール
        IMAGECOLOR_MONOCHROME,          // モノクローム
        IMAGECOLOR_BLACK                // テキスト、線画のハーフトーンなし
};
```

quality：JPEG 変換品質　※省略時および 1～100 以外の場合は 80 として処理します。

progressive：Progressive JPEG を作成する ※省略時 false

imageRect：イメージを作成するページ内の領域。空の矩形はページ全体　※省略時、空の矩形（ページ全体）

stream：JPEG を出力するストリーム

printMark：検索結果をイメージに反映する　※必須

fileName：出力ファイル名

戻り値：

ファイルがオープンできない、出力に失敗した場合 false

注意：

std:ostream インタフェースは SDK と同じバージョンのコンパイラのみ利用できます。

IMAGECOLOR_MONOCHROME、IMAGECOLOR_BLACK はサポートされません。

指定しても IMAGECOLOR_GRAYSCALE となります。

7.2.57.PNG の作成

```
bool makePngPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, bool interlace, const RECT& imageRect, std::ostream& stream,
        bool printMark) const;
bool MakePngPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
        bool interlace, AvsRect imageRect, Stream stream, bool printMark); (.Net)

bool makePngPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, bool interlace, const RECT& imageRect,
        const wchar_t* fileName, bool printMark) const;
bool MakePngPage(int pageNo, int dpi, String scale, int height, IMAGE_COLOR color,
        bool interlace, AvsRect imageRect, String fileName, bool printMark);   (.Net)

bool makePngPage(int pageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, bool interlace, const RECT& imageRect, IStreamPtr stream,
        bool printMark) const;
```

※上記以外に、引数を省略した関数が利用できます

引数：

pageNo：PNG を作成するページ番号(1 オリジン)　※必須

dpi：スケーリング時の DPI 値(1 以上、1440 以下)　※必須。1440dpi を超える場合 1440dpi で処理します。

scale：出力時のスケール文字列。100%以下、”1”==”100%”,”100px”は幅の指定　※必須

height：出力イメージの最大高さ(Pixel)、0 は高さ指定なし。※オプション（省略時 0）

dpi と scale から決定された高さが指定値を超える場合、指定値で出力します。

その場合の幅はページのアスペクト比から計算されます。

color：色数　※省略時　IMAGECOLOR_COLOR

```
enum IMAGE_COLOR {
        IMAGECOLOR_COLOR,           // フルカラー
        IMAGECOLOR_256COLOR,    // 256色カラー
        IMAGECOLOR_GRAYSCALE,  // グレースケール
        IMAGECOLOR_MONOCHROME,      // モノクローム
        IMAGECOLOR_BLACK            // テキスト、線画のハーフトーンなし
};
```

interlace：Interlace PNG を作成する ※省略時 false

imageRect：イメージを作成するページ内の領域。空の矩形はページ全体 ※省略時、空の矩形（ページ全体）

stream：PNG を出力するストリーム

printMark：検索結果をイメージに反映する　※必須

fileName：出力ファイル名

戻り値：

ファイルがオープンできない、出力に失敗した場合 false

注意：

std:ostream インタフェースは SDK と同じバージョンのコンパイラのみ利用できます。

7.2.58.TIFF の作成

```
bool makeTiffPage(int startPageNo, int endPageNo, int dpi, const wchar_t* scale,
        int height, IMAGE_COLOR color, const RECT& imageRect, std::ostream& stream,
```

```
        TIFFCOMPRESSION comp, int quality, bool printMark) const;
bool MakeTiffPage(int startPageNo, int endPageNo, int dpi, String scale, int height,
        AvsRect imageRect, Stream stream, bool printMark);

bool makeTiffPage(int startPageNo, int endPageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, const RECT& imageRect, const wchar_t* fileName,
        TIFFCOMPRESSION comp, int quality, bool printMark) const;
bool MakeTiffPage(int startPageNo, int endPageNo, int dpi, String scale, int height,
        IMAGE_COLOR color, AvsRect imageRect, String fileName, TIFFCOMPRESSION comp,
        int quality, bool printMark); (.Net)

bool makeTiffPage(int startPageNo, int endPageNo, int dpi, const wchar_t* scale, int height,
        IMAGE_COLOR color, const RECT& imageRect, IStreamPtr stream,
        TIFFCOMPRESSION comp, int quality, bool printMark) const;
```

※上記以外に、引数を省略した関数が利用できます

引数：

startPageNo：TIFF を作成する開始ページ番号(1 オリジン)　※必須

endPageNo：終了ページ番号　※必須

dpi：スケーリング時の DPI 値(1 以上、1440 以下)　※必須。1440dpi を超える場合 1440dpi で処理します。

scale：出力時のスケール文字列。100%以下、”1”==”100%”,”100px”は幅の指定　※必須

height：出力イメージの最大高さ(Pixel)、0 は高さ指定なし。※オプション（省略時 0）

dpi と scale から決定された高さが指定値を超える場合、指定値で出力します。

その場合の幅はページのアスペクト比から計算されます。

color：色数　※省略時　IMAGECOLOR_COLOR

```
enum IMAGE_COLOR {
        IMAGECOLOR_COLOR,               // フルカラー
        IMAGECOLOR_256COLOR,    // 256色カラー
        IMAGECOLOR_GRAYSCALE,  // グレースケール
        IMAGECOLOR_MONOCHROME,          // モノクローム
```

```
        IMAGECOLOR_BLACK                    // テキスト、線画のハーフトーンなし
    };
```

imageRect：イメージを作成するページ内の領域。空の矩形はページ全体 ※必須

stream：TIFF を出力するストリーム

printMark：検索結果をイメージに反映する ※必須

fileName：出力ファイル名

comp：圧縮方法の指定　※省略時、圧縮なし

```
    enum TIFFCOMPRESSION {
        TIFFCOMP_None = 0,          // 圧縮なし
        TIFFCOMP_LZW,                       // LZW圧縮
        TIFFCOMP_DCT,                       // JPEG圧縮
        TIFFCOMP_Flate,             // Flate圧縮
        TIFFCOMP_RunLength,         // ランレングス
        TIFFCOMP_CCITT3,            // CCITT G3圧縮
        TIFFCOMP_CCITT4             // CCITT G4圧縮
    };
```

quality：JPEG 変換品質(圧縮方法が DCT(JPEG)の場合有効)

※省略時および 1～100 以外の場合は 80 として処理します。

戻り値：

ファイルがオープンできない、出力に失敗した場合 false

注意：

std:ostream インタフェースは SDK と同じバージョンのコンパイラのみ利用できます。

7.2.59.検索ページ指定

```
void setCurrentPage(int pageNo) const;
void SetCurrentPage(int pageNo); (.Net)
```

引数：

pageNo：検索ページ番号（1 オリジン）

解説：

searchText での検索位置を pageNo の先頭に設定します。

ページ数より大きな数を指定すると最後のページの最終位置になります。

7.2.60.検索ページ数の設定

```
void setSearchPageCount(int count) const;
```

```
void SetSearchPageCount(int count); (.Net)
```

引数：

count：検索ページ数

解説：

searchText,searchAll で検索するページ数を設定します。

0 は未設定、ページ終了まで検索します。

7.2.61.文字列検索

```
bool searchText(const wchar_t* str, bool ignoreCase, bool ignoreWidth, bool reverse) const;
```

```
bool SearchText(String str, bool ignoreCase, bool ignoreWidth, bool reverse); (.Net)
```

引数：

str：検索文字列

ignoreCase：true 大文字小文字を区別しない

ignoreWidth：true 全角半角を区別しない

reverse：true 上検索

戻り値：

true 検索文字列が見つかった

解説：

検索文字列が無い場合、検索状態は変更されません

7.2.62.全検索

```
bool searchAll(const wchar_t* str, bool ignoreCase, bool ignoreWidth, bool reverse) const;
```

```
bool SearchAll(String str, bool ignoreCase, bool ignoreWidth, bool reverse); (.Net)
```

引数：

str：検索文字列

ignoreCase：true 大文字小文字を区別しない

ignoreWidth：true 全角半角を区別しない

reverse：true 上検索（V5.0MR1 以降）

戻り値：

true 検索文字列が見つかった

解説：

文書全体から文字列を検索します。複数検索される場合もあります。

検索文字列が無い場合、検索状態は変更されません

7.2.63.次検索

```
bool searchNext(bool reverse) const;
bool SearchNext(bool reverse); (.Net)
```

引数：

reverse：true 上検索

戻り値：

true 検索文字列が見つかった

解説：

文字列検索の場合は searchText()で検索した位置から次の検索を行います

全検索の場合は検索済み文字列の次の要素へ移動します

7.2.64.検索状態解除

```
bool clearSearch() const;
bool ClearSearch(void); (.Net)
```

戻り値：

true 検索状態を解除した。再表示が必要

7.2.65.マーク解除

```
void clearCurrentMark() const;
void ClearCurrentMark(void); (.Net)
```

解説：

全検索した現在位置のマークを解除します。

7.2.66.検索位置取得

```
int getSearchPosition(RECT& rect, bool bBox=false) const;
int GetSearchPosition(out AvsRect rect, bool bBox); (.Net)
int getSearchPositionRegion(HRGN& hRgn) const;
int GetSearchPositionRegion(out AvsRegion hRgn); (.Net)
```

引数：

rect：検索文字列矩形を受け取る RECT 構造体

bBox：true 検索文字列全体の矩形を得る

hRgn：検索文字列リージョン

戻り値：

検索文字列があるページ番号（1 オリジン）。0 は検索状態ではない

解説：

検索状態にある場合、これらの関数で現在の位置（検索でハイライトされた領域）を取得できます。複数行にまたがる文字列にヒットした場合はこの領域は単純な矩形にならず複数の矩形を合成した領域になります。このような場合、関数によって取得する情報が異なります。

getSearchPosition では取得する結果は矩形になります。

- bBox が true の場合はハイライトされた領域全体を含む矩形を取得します。
- bBox が false の場合は先頭の矩形検索位置の文字列の矩形を取得します。

また getSearchPositionRegion では、現在の位置は HRGN 型として得られます。検索結果の範囲が複数行にまたがる場合、リージョンは複数の矩形を合成した領域です。取得した hRgn は呼び出し側で DeleteObject()してください。

HRGN は Win32API に渡すことで描画など使用できます。HRGN や関連する API について [1]を参照してください。HRGN を構成する個々の矩形は RGNDATA 構造体から取得することができます。[2] ただし、.NET/.NET Framework の場合は対応する API がありませんので P/Invoke などを利用して Win32 API で RGNDATA を扱う必要があります。この場合、AvsRect の GetPointer()メソッドで HRGN を取得することができます。

参考 URL

[1] https://learn.microsoft.com/en-us/windows/win32/gdi/regions

[2] https://learn.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-rgndata

7.2.67.ページの検索結果取得

HRGN getPageSearchRegion(int pageNo) const;

AvsRegion GetPageSearchRegion(int pageNo); (.Net)

引数：

pageNo：取得するページ

戻り値：

検索結果のリージョン

解説：

全検索をしたときの指定ページの検索された文字列のリージョンを取得します。

リージョンは呼び出し側で DeleteObject()してください。

7.2.68.検索状態かの問い合わせ

bool isSearched() const;

property bool IsSearched (.Net) 読み取りのみ

戻り値：

true 検索状態（文字列検索、全検索）

7.2.69.全検索状態かの問い合わせ

bool isSearchedAll() const;

property bool IsSearchedAll (.Net) 読み取りのみ

戻り値：

true 全検索状態

7.2.70.全検索でマークした個数を得る

int getMarkupCount() const;

property int MarkupCount (.Net) 読み取りのみ

戻り値：

全検索で見つかった個数

7.2.71.反転色の設定

void setSearchedColor(COLORREF selectText, COLORREF selectBack, COLORREF currentText, COLORREF currentBack) const;

void SetSearchedColor(AvsColor selectText, AvsColor selectBack, AvsColor currentText, AvsColor currentBack); (.Net)

引数：

selectText：検索した文字列の色

selectBack：検索した文字列の背景色

currentText：現在位置の文字列の色

currentBack：現在位置の文字列の背景色

7.2.72.矩形内テキストの取得

int getTextInRect(int pageNo, const RECT* rect, wchar_t* buffer, int size) const;

String GetTextInRect(int pageNo, AvsRect rect); (.Net)

引数：

pageNo：指定矩形ページ番号（１オリジン）

rect：矩形、NULL の場合はページ全体

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。buffer が NULL の場合は必要なバッファの文字数を返す（C++）

取得した文字列 (.Net)

解説：

指定した矩形に完全に含まれるテキストを取得します。

getSearchPosition で取得した矩形を使う場合は、その矩形を 1TWIP 単位上下左右に拡げた矩形を渡してください。

7.2.73.全選択設定

```
bool setSelectAll(bool select) const;
```

```
bool SetSelectAll(bool select);　(.Net)
```

引数：

select：全選択状態

戻り値：

以前の状態

7.2.74.全テキスト取得

```
int getAllText(wchar_t* buffer, int size) const;
```

```
String GetAllText(); (.Net)
```

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字。buffer が NULL の場合は必要なバッファの文字数を返す　(C++)

取得した文字列（.Net)

解説：

全選択状態のテキストを取得します。

setSelectAll(true)に設定して取得します。

読み込み済みページまでのテキストのみ取得できます。

7.2.75.矩形選択の設定

```
void setReverseTextRect(int pageNo, const RECT* rect,
        COLORREF text=RGB(255,255,255), COLORREF back=RGB(0,0,0)) const;
```

```
void SetReverseTextRect(int pageNo, AvsRect rect, AvsColor text, AvsColor back); (.Net)
```

引数：

pageNo：矩形ページ番号

rect：矩形、NULL は解除

text：文字色

back：背景色

解説：

指定した矩形内の文字列を text,back の色で表示するよう設定します。

矩形内のテキストは getTextInRect()で取得することができます。

rect に NULL を指定すると設定を解除します。

表示の優先順位は、全選択状態＞矩形選択＞検索状態になります。

7.2.76.ページ回転の取得

```
int getPageRotation(int pageNo) const;
int GetPageRotation(int pageNo); (.Net)
```

引数：

pageNo：回転を取得するページ番号

戻り値：

ページの回転角度(0, 90, 180, 270)

7.2.77.ページ回転の設定

```
int setPageRotation(int pageNo, int rotation) const;
int SetPageRotation(int pageNo, int rotation); (.Net)
```

引数：

pageNo：回転を設定するページ番号

rotation：新しい回転角度(0, 90, 180, 270)

戻り値：

設定されたページの回転角度(0, 90, 180, 270)

解説：

回転を変更し再表示してください。ページの幅、高さは変更されます。

7.2.78.リンク注釈ヒットテスト

```
LINK_ACTION hitTestAnchor(int pageNo, const POINT& mousePos, RECT* anchorRect) const;
```

LINK_ACTION HitTestAnchor(int pageNo, AvsPoint mousePos, out AvsRect anchorRect); (.Net)

引数：

pageNo：ヒットテストを行うページ番号

mousePos：マウス位置

anchorRect：アンカー矩形を取得する RECT

戻り値：

マウス位置のリンク注釈アクション

```
enum LINK_ACTION {
        ACTIONTYPE_None = 0,      // アクションなし
        ACTIONTYPE_GoTo,          // Gotoアクション
        ACTIONTYPE_GoToR,         // GotoRアクション
        ACTIONTYPE_Launch,        // Launchアクション
        ACTIONTYPE_URI            // URIアクション
};
```

7.2.79.リンク注釈内部矩形取得

int getInternalTarget(RECT* targetRect) const;

int GetInternalTarget(out AvsRect% targetRect);

引数：

targetRect：飛び先の矩形を取得する RECT

戻り値：

飛び先のページ番号（0 は無効）

解説：

前回の hitTestAnchor()で得られたリンク注釈が PDF 内(GoTo)の場合、その飛び先を取得します。

7.2.80.リンク注釈外部ファイル名取得

int getExternalUri(wchar_t* buffer, int size) const;

property String ExternalUri (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数

buffer が NULL の場合は必要なバッファの文字数を返す

解説：

前回の hitTestAnchor()で得られたリンク注釈が外部ファイル(GoToR,Launch,URI)の場合、その飛び先の URI を取得します。

GoToR,Launch でファイルが相対指定で、PDF の open()がファイル名指定の場合、開いている PDF の相対ファイルとして実際のファイル名を返します。

URI の相対指定は PDF の URI 辞書の/Base の指定があればそこからの相対として実際の URI を返します。

簡単なリンク注釈の使い方がサンプルプログラムに実装してあります。参照してください。

7.2.81.デフォルトフォントの設定

```
void setGenericFont(const wchar_t* script, const wchar_t* genericFont,
        const wchar_t* fontname);
void SetGenericFont(String script, String genericFont, String fontname); (.Net)
```

引数：

script：文字のスクリプト。日本は”ja”

genericFont：generic フォント名。”serif”, “sans-serif”

fontname：フォント名

解説：

PDF にフォントファイルがなく指定されているフォント名が探せない場合、スクリプトごと generic フォント名に登録されているフォントを使用します。

generic フォント名は PDF フォントデスクリプタの Flags に Serif が指定されている場合”sefif”、指定されていなければ”sans-serif”になります。

デフォルトでは日本語の”serif”は”ＭＳ　明朝”です。これを”ＭＳ　Ｐ明朝”に指定する場合は

setGenericFont(“ja”, “serif”, “ＭＳ　Ｐ明朝”);

とします。

呼び出しはオープンの前に行ってください。

7.2.82.読み込みページ数の設定

```
void setLoadPageCount(int count);
property int LoadPageCount (.Net)
```

引数：

count：メモリに読み込むページ数。初期値２００。値範囲：１０以上、上限なし。

解説：

メモリに読み込むページ数を設定します。

呼び出しはオープンの前に行ってください。

設定したページ数より PDF のページ数が多いと、設定したページ数までメモリに持ちます。

ファイル名でオープンした場合はファイルがオープンしたままになります。

オープンしたままにしたくない場合はストリームでオープンしてください。

7.2.83.読み込みメモリ上限の設定

```
void setLoadMemoryLimit(DWORD limit);
property UInt32 LoadMemoryLimit
```

引数：

limit：プロセスヒープメモリの上限。（単位 Byte、KB 未満切捨て）

初期値５００ＭＢ。値範囲：0 以上、上限なし。

解説：

プロセスのヒープメモリの上限を設定します。

呼び出しはオープンの前に行ってください。

プロセスのヒープメモリが設定した値を超えると、読み込みスレッドを打ち切り、そのときのページ数までメモリに保持します。

ファイル名でオープンした場合はファイルがオープンしたままになります。

オープンしたままにしたくない場合はストリームでオープンしてください。

ストリームでオープンする場合、内部のメモリストリームにコピーしますので、ファイルサイズ分のメモリが必要となります。ファイル名でオープンする場合は、ストリームへのコピーは行いませんのでメモリ消費が少なくなります。

7.2.84.テキスト位置の取得

int getPageTextPosition(int pageNo, const POINT& mousePos) const;

int GetPageTextPosition(int pageNo, AvsPoint mousePos); (.Net)

引数：

pageNo：位置を取得するページ番号

mousePos：マウス位置

戻り値：

テキスト位置

解説：

マウス位置のテキスト位置を取得します。

サンプルでは OnLButtonDown()で選択開始、終了位置を取得しています。

OnMouseMove()で終了位置を取得、更新し、再描画範囲を設定します。

7.2.85.テキスト範囲の取得

HRGN getPageTextRegion(int pageNo, int start, int end) const;

AvsRegion GetPageTextRegion(int pageNo, int start, int end); (.Net)

HRGN getPageTextRegion(int pageNo, int start, int end, bool encodeV) const;

AvsRegion GetPageTextRegion(int pageNo, int start, int end, bool encodeV); (.Net)

引数：

pageNo：範囲を取得するページ番号

start：選択開始位置

end：選択終了位置

encodeV：true の場合は縦書きテキスト。false では縦書き以外

戻り値：

開始から終了までのテキストのリージョン

解説：

取得した HRGN は呼び出し側で DeleteObject()してください。

ページをまたがる範囲の場合（1 ページ目 start から 2 ページ目 end など）、テキスト位置がページの先頭の場合は TEXTPOSITION_FIRST を、ページ最後の場合は TEXTPOSITION_LAST を指定してください。

```
getPageTextRegion(1, start, TEXTPOSITION_LAST);    //  1 ページ目
getPageTextRegion(2, TEXTPOSITION_FIRST, end);    //  2 ページ目
```

7.2.86.位置指定テキスト取得

```
int getPageTextString(int pageNo, o, int start, int end, wchar_t* buffer, int size) const;
String GetPageTextString(int pageNo, int start, int end); (.Net)
```

引数：

pageNo：テキストのあるページ番号

start：選択開始位置

end：選択終了位置

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。buffer が NULL の場合は必要なバッファの文字数を返す

取得した文字列　(.Net)

7.2.87.矩形内テキスト範囲の取得

```
HRGN getTextRegionInRect(int pageNo, const RECT* rect) const;
AvsRegion GetTextRegionInRect(int pageNo, AvsRect rect); (.Net)
HRGN getTextRegionInRect(int pageNo, const RECT* rect, bool encodeV) const;
AvsRegion GetTextRegionInRect(int pageNo, AvsRect rect, bool encodeV); (.Net)
```

引数：

pageNo：指定矩形ページ番号（1 オリジン）

rect：矩形、NULL の場合はページ全体

encodeV：true の場合は縦書きテキスト。false では縦書き以外

戻り値：

矩形内テキストのリージョン

解説：

取得した HRGN は呼び出し側で DeleteObject()してください。

7.2.88.矩形内テキスト情報の取得

```
int getTextInfoInRect(int pageNo, const RECT* rect, TextInfo* buffer, int size) const;
ArrayList GetTextInfoInRect(int pageNo, AvsRect rect); (.Net)
```

C++

```
enum {
        TI_MAX_TEXT_LEN = 4,
        TI_MAX_FONTNAME_LEN = 128
};
enum {
        WM_f_isRtlOrBtt = 1 << 0,
        WM_f_isVerticalText = 1 << 1,
        WM_f_isBPDirLtrOrBtt = 1 << 2
};
enum WritingModeEnum {
        WM_lr_tb = 0,
        WM_rl_tb = WM_f_isRtlOrBtt,
        WM_tb_rl = WM_f_isVerticalText,
        WM_bt_rl = WM_f_isRtlOrBtt | WM_f_isVerticalText,
        WM_lr_bt = WM_f_isBPDirLtrOrBtt,
        WM_rl_bt = WM_f_isRtlOrBtt | WM_f_isBPDirLtrOrBtt,
        WM_tb_lr = WM_f_isVerticalText | WM_f_isBPDirLtrOrBtt,
        WM_bt_lr = WM_f_isRtlOrBtt | WM_f_isVerticalText | WM_f_isBPDirLtrOrBtt
};
class TextInfo {
public:
        wchar_t   text[TI_MAX_TEXT_LEN];
        RECT      rect;
```

```
    int         baseline;
    int         fontSize;
    wchar_t     fontName[TI_MAX_FONTNAME_LEN];
    bool        isNewLine;
    int         writingMode;
    bool        isVertical;
};
```

.NET Framework /.NET6 (C#)

```
public enum AvsWritingMode {
    WM_lr_tb = 0,
    WM_rl_tb = 1, //WM_f_isRtlOrBtt,
    WM_tb_rl = 2, //WM_f_isVerticalText,
    WM_bt_rl = 3, //WM_f_isRtlOrBtt | WM_f_isVerticalText,
    WM_lr_bt = 4, //WM_f_isBPDirLtrOrBtt,
    WM_rl_bt = 5, //WM_f_isRtlOrBtt | WM_f_isBPDirLtrOrBtt,
    WM_tb_lr = 6, //WM_f_isVerticalText | WM_f_isBPDirLtrOrBtt,
    WM_bt_lr = 7, //WM_f_isRtlOrBtt | WM_f_isVerticalText | WM_f_isBPDirLtrOrBtt
}
public class AvsTextInfo {
    public String       Text;
    public AvsRect      Rect;
    public int          Baseline;
    public int          FontSize;
    public String       FontName;
    public bool         IsNewLine;
    public int          PageNumber;
    public AvsWritingMode       WritingMode;
    public bool         IsVertical;
```

}

引数：

pageNo：指定矩形ページ番号（１オリジン）

rect：矩形、NULL の場合はページ全体

buffer: 取得したテキスト情報(TextInfo クラス)を格納する配列

size:バッファのサイズ

戻り値：

取得されたサイズ

buffer が NULL の場合は必要なバッファのサイズを返す

解説：

指定ページの指定された矩形内に完全に含まれる文字の情報を取得します。

・フォント名はベースフォント名を取得します。

・文字の矩形、ベースラインは論理座標におけるページの原点に対する位置になります。

・改行位置には、isNewLine=true である文字情報が含まれます。isNewLine=true の場合、TextInfo のその他のメンバー変数は無効です。

・writingMode は文字列の方向です。以下の数値で表されます。方向は PDF Viewer API の座標系における見かけ上の方向を表します。ページ回転も考慮されます。

WM_lr_tb (0)　左から右

WM_rl_tb (1)　右から左

WM_tb_rl (2)　上から下

WM_bt_rl (3)　下から上

WM_lr_bt (4)　左から右（WM_lr_tb の上下反転）

WM_rl_bt (5)　右から左（WM_rl_tb の上下反転）

WM_tb_lr (6)　上から下（WM_tb_rl の左右反転）

WM_bt_lr (7)　下から上（WM_bt_rl の左右反転）

・isVertical は内部の PDF のフォントが縦書き（tue）用か横書き（false）用かを表します。必ずしも見かけ上の縦書きか横書きかとは一致しません。

制限事項：

・座標変換により文字に回転や歪み(skew)がある場合は正しい情報が取得できません。

・現在の実装では、テキストのベースラインに一定量変化があるときに改行と判定しています。縦書きや斜めに傾いている場合には正しく機能しません。

・改行位置の判定は PDF 仕様では明確に既定されないため、弊社リーダーの実装依存の動作になります。このため、他社のリーダーと同一にはなりません。

・座標変換に 0/90/180/270 度以外の回転がある場合は文字列の方向が正しく取得できません。

7.2.89.位置指定テキスト情報の取得

```
int getPageTextInfo(int pageNo, int start, int end, TextInfo* buffer, int size) const;
```

※本 API は C++ I/F のみです。

引数：

pageNo：指定矩形ページ番号（１オリジン）

start：選択開始位置

end：選択終了位置

buffer: 取得したテキスト情報(TextInfo クラス)を格納する配列

size:バッファのサイズ

戻り値：

取得されたサイズ

buffer が NULL の場合は必要なバッファのサイズを返す

解説：

指定ページの指定された範囲の文字の情報を取得します。

取得できる情報（AvsTextInfo）について、getTextInfoInRect メソッドの解説を参照ください。

制限事項：

getTextInfoInRect メソッドの制限事項を参照ください。

7.2.90.矩形内パス情報の取得

```
int getPagePathInRect(int pageNo, const RECT* rect, PDFPath** list) const;
```

```
enum PDFPathOp {                // パス生成オペレータ
        PathOp_Unknown,
        PathOp_m,
```

```
        PathOp_l,
        PathOp_c,
        PathOp_v,
        PathOp_y,
        PathOp_h,
        PathOp_re
};
enum PDFPaintOp {                   // パスペイントオペレータ
        PaintOp_Unknown,
        PaintOp_S,
        PaintOp_s,
        PaintOp_f,
        PaintOp_F,
        PaintOp_f_,
        PaintOp_B,
        PaintOp_B_,
        PaintOp_b,
        PaintOp_b_,
        PaintOp_n,
        PaintOp_W,
        PaintOp_W_
};
class PDFSubPath {          // sub path クラス
public:
        int size;                   // subpath を構成する点の数
        POINT* points;              // subpath を構成する点の配列
        PDFPathOp pathOp;           // subpath のオペレータ
};
class PDFPath {                     // path クラス
```

```
public:
        int size;                   // subpath の数
        PDFSubPath* subpaths;       // subpath の配列
        PDFPaintOp paintOp;         // パスペイントオペレータ
        int lineWidth;              //
};
```

※本 API は C++ I/F のみです。

引数：

pageNo：指定矩形ページ番号（１オリジン）

rect：矩形、NULL の場合はページ全体

path: 取得したパスを格納する配列へのポインタ

戻り値：

取得されたパスの個数

解説：

指定ページの指定された矩形内に完全に含まれるパスをすべて取得します。取得される
パスには実際に描画されるパスだけでなく、クリッピング用のパスなどすべてのパスが
含まれます。

パスがどのように描画されるかは、PDFPath クラスの paintOp メンバ変数のパスペイントオペレータ（enum PDFPaintOp 型）で判定してください。パスペイントオペレータの詳細は ISO32000-2 の「8.5.3Path-Painting Operators」をご覧ください。

パスは複数のサブパス(PDFSubPath クラス)で構成されます。パスの構成については、ISO32000-2 の「8.5.2Path Construction Operators」をご覧ください。

取得されるパス情報は API 側でメモリが割り当てられます。メモリの廃棄はアプリケーション側で行ってください。

7.2.91.カラープロファイルの設定

```
void setOutputColorProfile(std::istream& outputRGBProfile);
void SetOutputColorProfile(Stream outputRGBProfile); (.Net)

void setRGBColorProfile(std::istream& readRGBProfile);
```

```
void SetRGBColorProfile(Stream readRGBProfile); (.Net)

void setGrayscaleColorProfile(std::istream& readGrayscaleProfile);
void SetGrayscaleColorProfile(Stream readGrayscaleProfile); (.Net)

void setCMYKColorProfile(std::istream& readCMYKProfile);
void SetCMYKColorProfile(Stream readCMYKProfile); (.Net)

void setOutputColorProfile(const wchar_t* outputRGBProfile);
void SetOutputColorProfile(String outputRGBProfile); (.Net)

void setRGBColorProfile(const wchar_t*   readRGBProfile);
void SetRGBColorProfile(String readRGBProfile); (.Net)

void setGrayscaleColorProfile(const wchar_t*   readGrayscaleProfile);
void SetGrayscaleColorProfile(String readGrayscaleProfile); (.Net)

void setCMYKColorProfile(const wchar_t*   readCMYKProfile);
void SetCMYKColorProfile(String^ readCMYKProfile);   (.Net)
```

引数：

outputRGBProfile：表示用 RGB カラープロファイル

readRGBProfile：PDF RGB カラープロファイル

readGrayscaleProfile：PDF GrayScale カラープロファイル

readCMYKProfile：PDF CMYK カラープロファイル

解説：

カラープロファイルの指定は openDocument()前に行ってください。

PDF ファイルをオープン後にプロファイルを指定した場合、次回のオープンまで反映されません。

7.2.92.エラーがあるか

bool hasError() const;

property bool HasError (.Net) 読み取りのみ

引数：

戻り値：

true エラーがある

解説：

エラーは複数ある場合があります。エラーがある場合は getErrorMessage()で得られるメッセージを表示し、abandonError()または clearError()でエラーを廃棄してください。廃棄しない場合、エラーが発生する度にエラーは蓄積されていきます。API 呼出が成功した場合でも致命的でないエラー（警告）が発生する場合があります。

7.2.93.致命的エラーか

bool isFatalError() const;

property bool IsFatalError (.Net)

戻り値：

true エラーが致命的エラーである

解説：

蓄積された最初のエラーについて確認します。

7.2.94.エラー番号の取得

unsigned short getErrorCode() const;

property unsigned short ErrorCode (.Net)

戻り値：

エラー番号

解説：

蓄積された最初のエラーのエラー番号を返します。

7.2.95.エラーメッセージの取得

int getErrorMessage(wchar_t* buffer, int size) const;

property String ErrorMessage (.Net)

引数：

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。buffer が NULL の場合は必要なバッファの文字数を返す（C++）

取得した文字列　(.Net)

解説：

蓄積された最初のエラーのエラーメッセージを取得します。

7.2.96.エラーの廃棄

void abandonError() const;

void AbandonError(void); (.Net)

解説：

蓄積されたエラーのうち、最初のエラーを一つ廃棄します

7.2.97.エラーのクリア

void clearError() const;

void ClearError(void); (.Net)

解説：

エラーを全て廃棄します

7.2.98.ライセンス情報の取得

static int getLicenseInfo(LicenseInfoType type, wchar_t* buffer, int size) const;

String GetLicenseInfo(AvsLicenseInfoType type)（.Net）

引数：

type：取得するライセンス情報の種別

buffer：文字列バッファ

size：バッファサイズ（文字数）

戻り値：

コピーした文字数。buffer が NULL の場合は必要なバッファの文字数を返す（C++）

取得した文字列 （.Net）

C++

```
enum LicenseInfoType {
        LI_PATH,
        LI_SERIAL,
        LI_COMPANY,
        LI_SECTION,
        LI_USERNAME,
        LI_MAINTENANCE_LIMIT,
        };
```

.NET Framework /.NET6 (C#)

```
public enum class AvsLicenseInfoType {
        LI_PATH,
        LI_SERIAL,
        LI_COMPANY,
        LI_SECTION,
        LI_USERNAME,
        LI_MAINTENANCE_LIMIT,
};
```

enum 値	値	説明
LI_PATH	0	ライセンスファイル（pdfviewersdk.lic）の絶対パス

LI_SERIAL	1	シリアルナンバー
LI_COMPANY	2	会社名
LI_SECTION	3	部署名
LI_USERNAME	4	ユーザー名
LI_MAINTENANCE_LIMIT	5	保守期限

解説：

指定されたライセンス情報を取得します。

ライセンスファイルに登録がない場合、空文字列が返る場合があります。

8. エラー一覧表

- エラーメッセージ中の XXXXX はメッセージの詳細です。「メッセージ詳細」を参照ください。
- 「メッセージ詳細」に「※PDF エラー一覧」と記載があるメッセージは、詳細なエラーコード（xxxx）および内容（XXXXX）について「PDF エラー一覧」を参照ください。
- 「内容」の「※」は PDF 仕様をサポートしていない場合に出力されるメッセージであることを示しています。
- エラーメッセージには、この他にも詳細情報が付加される場合があります。

8.1. エラー一覧表

エラーコード 10 進(16 進)	エラーメッセージ	内容	メッセージ詳細
1 (0001)	SYSTEM ERROR: Out of memory.	メモリ不足	−
2 (0002)	SYSTEM ERROR: C++ exception: <message>	C++ライブラリ例外	−
16385 (4001)	Printing is canceled. XXXXX	印刷中止	(StartDoc() failed.) <printer_name>
16387 (4003)	StartPage() failed or canceled.	StartPage() 失敗	−
16388 (4004)	EndPage() failed.	EndPage() 失敗	−

16401 (4011)	Rasterization error: XXXXX	画像出力失敗	Cannot create rendering bitmap. Cannot open file Cannot save bitmap. Cannot write stream Cannot convert image. Cannot write multi frame TIFF Cannot flush multi frame TIFF
16413 (401D)	AHPDF error: System Error : page <pageno>	PDF データ解析時のエラー	－
16420 (4024)	Cannot create font for water-mark.	ウォーターマーク用フォントが選択できない	－
16897 (4201)	File name is not specified: axf:annotation-file-attachment.	注釈ファイル名がない	－
16898 (4202)	Attachment is not found: axf:annotation-file-attachment. file: <filename>	添付ファイルがない	－
16902 (4206)	AHPDF error (xxxx) : XXXXX	ページコンテンツ解析時のエラー	※PDF エラー一覧
16903 (4207)	AHPDF warning (xxxx) : XXXXX	ページコンテンツ解析時の警告	※PDF エラー一覧
16905 (4209)	Unsupported PDF Shading Type <shading_type>	サポートされないシェーディングタイプ　※	－
17284 (4384)	AHPDF information (xxxx) : XXXXX	ページコンテンツ解析時の情報	※PDF エラー一覧
24585 (6009)	Cannot load license file: <licpath>	ライセンスが読めない	

24591 (600F)	Evaluation license is expired: <licpath>	期限切れの評価版/試用版	
32791 (8017)	Cannot open input file: <uri> Cannot open input file: <uri> XXXX	入力ファイルがオープンできない。	Cannot connect the Internet Cannot open URL HTTP request failure HTTP status is yyy Cannot get HTTP status Cannot access the Internet
57473 (E081)	Cannot open PDF file.	入力ファイルがオープンできない。	－
	Invalid PDF file. (xxxx)	入力 PDF ファイルが不正	※PDF エラー一覧
	Invalid PDF file. XXXXX (xxxx)	入力 PDF ファイルが不正	※PDF エラー一覧
	Invalid PDF stream. (xxxx)	入力 PDF ストリームが不正	※PDF エラー一覧
	Invalid PDF stream. XXXXX (xxxx)	入力 PDF ストリームが不正	※PDF エラー一覧
	PDF file is empty page.	PDF ファイルのページがない	－
57474 (E082)	PDF file is protected by a Password.	PDF ファイルはパスワードで保護されている	－
	The password is incorrect.	パスワードが不正	－
57985 (E281)	AHPDF warning (xxxx): XXXXX	PDF ファイル解析時の警告（ページコンテンツ以外）	※PDF エラー一覧
58241 (E381)	AHPDF information (xxxx): XXXXX	PDF ファイル解析時の情報（ページコンテンツ以外）	※PDF エラー一覧

58241 (E381)	RenderDirect2D warning (xxxx): XXXXX	Direct2D レンダリング時の警告	drawGraphic failed with E_INVALIDARG drawGraphic failed with E_OUTOFMEMORY

8.2. PDF エラー一覧

エラーコード 10 進	エラーメッセージ	内容
2	No authentication	認証されていない
3	Password is required	パスワードが必要
4	Password is incorrect	パスワード不一致
101	URI is empty	URI が空
102	Cannot create input stream	入力ストリームが作れない
201	This is not a PDF	PDF でない
202	Invalid PDF header	PDF ヘッダ不正
203	Unsupported PDF version	非サポート PDF バージョン　※
205	Missing startxref	startxref がない
206	Invalid <offset> offset	startxref オフセットが正しくない
207	Out of range xref offset <offset>	xref オフセットが範囲外
208	Expected xref	xref が見つからない
211	Expected trailer	trailer がない
212	Missing catalog dictionary for the PDF document	文書カタログがない
215	Expected integer object	整数でなければならない
217	Invalid value	値不正（引数に値）
218	Expected dictionary object	辞書でなければならない
219	Missing dictionary entry	辞書のエントリが存在しない
220	Missing or invalid dictionary entry	辞書のエントリが存在しないか値が正しくない
221	Invalid dictionary entry	辞書のエントリ値が正しくない（引数にキーと値）
222	Invalid array element at <offset>	配列の要素が正しくない（引数に要素番号）
223	Missing stream dictionary	stream 辞書がない
224	Insufficient cross-reference stream	cross-reference ストリームのサイズ不足

225	Invalid cross-reference stream	cross-reference ストリームが正しくない
227	Unsupported security handler	非サポートセキュリティハンドラ　※
228	Unsupported crypt algorithm	非サポート暗号アルゴリズム　※
231	Unsupported revision of standard security handler	非サポートリビジョン（標準セキュリティハンドラ）※
232	Failure to decrypt object	オブジェクトの復号に失敗
235	Undefined font of <token_offset> at <offset>	フォントがない
236	Unsupported PDF that exceeds 2GB	2GB 超の PDF　※
1030	Missing %%EOF	%%EOF がない
1031	Missing startxref	startxref がない
1032	Invalid startxref offset	startxref オフセットが正しくない
1034	Out of range xref offset <offset>	xref オフセットが範囲外
1035	Expected xref	xref が見つからない
1036	Expected xref	xref が見つからない
1037	Invalid EOL xref entry	xref のエントリ末 EOL 不正
1038	Invalid xref entry	xref のエントリ不正
1041	Expected number object	数値でなければならない
1042	Expected name object	名前でなければならない
1043	Expected string object	文字列でなければならない
1044	Expected dictionary object	辞書でなければならない
1046	Expected name or array object	名前または配列でなければならない
1047	Expected stream object	ストリームでなければならない
1048	Invalid token	不正なトークン（無視される）
1050	Invalid name	/の後の名前に不正な#が含まれている(#を適当に解釈)
1051	Too long name	/の後の名前が長すぎる（そのまま）
1052	Expected object number	object number が含まれていない
1053	Expected generation number	generation number が含まれていない

1054	Invalid object number	object number 不正
1055	Invalid generation number	generation number 不正
1057	Too many obj contents	obj 内容が複数
1058	No obj contents	obj 内容がない（NullObject で代用）
1059	Missing endobj	endobj がない
1060	No <offset> entry <objnum> <generation>	xref に登録されていない object number
1061	Missing <endtoken>	終端トークンがない
1062	Invalid value	値不正（引数に値）
1066	Missing dictionary entry	辞書のエントリが存在しない（引数にキー）
1067	Missing or invalid dictionary entry	辞書のエントリが存在しないか値が正しくない（引数にキー）
1068	Invalid dictionary entry	辞書のエントリ値が正しくない（引数にキーと値）
1069	Dictionary key is not a name	辞書のキーが名前でない
1070	Duplicated dictionary key	辞書のキーが登録済み
1071	Missing dictionary value	辞書の値がない
1073	Dictionary value is not an integer	辞書の値が整数でない
1074	Dictionary value is not a number	辞書の値が数値でない
1075	Dictionary value is not an array	辞書の値が配列でない
1076	Dictionary value is not a dictionary	辞書の値が辞書でない
1078	Invalid rectangle array	辞書中の配列が矩形として正しくない（引数に辞書キー）
1079	Invalid number value	辞書の数値が正しくない（引数に辞書キーと値）
1084	Missing endstream	endstream がない
1085	Out of object-stream content	object-stream の内容量以上の指定
1086	Invalid object-stream content	object-stream の内容が正しくない
1088	Invalid array element at <arrayoffset>	配列の要素が正しくない（引数に要素番号）
1089	Array size is odd	配列の要素数が奇数
1090	Insufficent array size	配列の要素数が足りない（引数に必要数）

1091	Excessive array size	配列の要素数が多すぎる（引数に必要数）
1092	Invalid array size	配列の要素数が正しくない（引数に必要数）
1093	Invalud number value	配列の数値が正しくない（引数に要素番号と値）
1094	Unknown or unsupported filter	不明または未サポートのフィルタ　※
1096	Invalid stream data	ストリームデータ不正（デコード失敗）
1098	Invalid operand <operand> of <offset>	コンテントのオペランド不足または、型が違う
1099	Reach end of contents, but <size> objects are left	コンテントが空でないのに終わりに達した
1100	Graphic state is not saved	GraphicState が保存されていない（q が多い）
1101	Nested <operator> at <offset>	BT/BI が入れ子になっている
1102	<operator> is not called	BT/BX/BI が呼ばれていない
1103	Invalid key value in BI-ID content	BI-ID 内の対の要素の最初がキーでない
1104	Invalid entries count in BI-ID content	BI-ID 内の要素が奇数
1105	Missing operator <operator>	EI/EX/EMC がない
1109	Invalid color value	不正な色（値が範囲外）
1111	Unsupported color space	未サポートの色空間　※
1112	Unsupported font type	未サポートのフォント種別　※
1113	Invalid /Differences code	/Difference のコード不正
1115	Unknown or unsupported annotation type	不明な注釈種別　※
1116	Unknown or unsupported action type	不明なアクション種別　※
1119	Not found destination name	宛先名が見つからない
1120	No file spec exists	ファイル指定にファイル名が含まれていない
1121	Invalid RichText XML	不正な RichText
1122	Invalid CSS style text	不正な CSS スタイル
1123	Not found first page object <objnum> in Linearized PDF	線形化 PDF で最初のページのオブジェクトがない
1124	First page of Linearized PDF is not 0	線形化 PDF で最初のページが 0 でない
1126	Unknown or unsupported field type	不明なフィールド種別　※

1129	Invalid white space after ID	ID に続く空白文字が不正
1131	Cannot resolve indirect object	間接参照が解決できない
1132	Not found page object	ページオブジェクトがない
1151	Invalid image size	画像サイズ不正
1191	PostScript stack underflow	関数評価でスタックアンダフロー
1192	PostScript divide by zero	関数評価でゼロ除算
1193	PostScript math error	関数評価で算術演算例外
1194	PostScript syntax error	PS 構文エラー
1195	PostScript invalid operand	PS オペランド不正
1196	PostScript empty dictionary stack	PS 辞書スタックが空
2049	Password is required	パスワードが必要
2051	Password is incorrect	パスワード不一致
2104	Unsupported value	未サポートの辞書の値（引数に辞書キーと値） ※
2105	Invalid date value	不正な日付書式（引数にキーと値）
2106	Invalid text string	不正なテキスト文字列（引数にキー）
2083	Missing endobj	endobj がない
2091	Missing or invalid dictionary entry	辞書のエントリが存在しないか値が正しくない（引数にキー）
2092	Invalid dictionary entry	辞書のエントリ値が正しくない（引数にキーと値）
2096	Dictionary value is not an boolean	辞書の値が論理値でない
2119	Invalid use of filter alias	不正な別名フィルタの使用
2121	Invalid stream length	ストリームのサイズ不正（endstream が後ろに出現）
2130	Unknown operator	不明なオペレータ（他で代替）
2141	Unsupported action type	未サポートのアクション（引数に注釈種別） ※
2142	Unsupported annotation type	未サポートの注釈（引数に注釈種別） ※
2149	Hint stream object should be <num> instead of <num>	ヒントストリームが最後のオブジェクトでない
2176	Conflict colorspace image	色空間の矛盾する画像
2177	Conflict bits per component image	BitsPerComponent の矛盾する画像

2178	Conflict image mask	MASK の矛盾する画像
2500	PDF header is not top of file	%PDF がファイルの先頭でない
2501	Reconstruct xref table	xref 再構築

改訂履歴

年月	改定内容
2021 年 6 月	初版
2021 年 12 月	V5.0 MR1
2022 年 7 月	V5.0 MR2
2022 年 12 月	V5.0 改訂 3 版
2023 年 4 月	V5.0 改訂 4 版