Foxit®

User Manual

Foxit® PDF SDK (ActiveX) Programming Guide

For Windows

Microsoft® Partner

Gold Independent Software Vendor (ISV)

Contents

Overview

Foxit PDF SDK ActiveX is a visual programming component that offers PDF displaying capability with minimal resource demand and redistribution size. It can be easily integrated into a wide range of applications.

Foxit PDF SDK ActiveX uses the same parsing and rendering engine as Foxit Reader. Therefore it can display PDF files with the same high quality and fast speed as Foxit Reader.

Compared to the DLL version of Foxit SDK, the ActiveX version is much easier to use and has much more rich features built inside. A programmer can simply drag and drop the component into their application and instantly add PDF displaying functionality. In addition, the ActiveX allows users to navigate, zoom, rotate, scroll and print out PDF documents.

Version 5.0 incorporates Signature Module and many advance PDF features. It allows developers to enable their applications to digitally sign PDF documents using third-party digital certificates located in the Windows system certificate store. It also supports digital signature verification (which authenticates the source and verifies the integrity of content of digitally signed PDF documents), signature timestamp, signature print control, and more.

Version 5.0 also introduces more functions and events, giving programmers more control over the component and more access to the PDF documents.

Foxit offers two versions of ActiveX 5.0 (standard and professional), which are provided with different GUID numbers that allows you to register both versions onto the same computer and access them with their own GUID. Compared with the professional version, the standard version doesn't include the following features: creating/editing annotation, importing/exporting form data, running JavaScript, converting PDF to text, Digital Signature, etc. Based on your requirement needs you are able to pick and choose which ActiveX 5.0 version would be best for your application. You can purchase either the standard or professional version online. For the Form, Annotation, and Signature Module, please contact at sales@foxitsoftware.com for licensing details. In this developer's guide, all the properties and functions marked with an asterisk (*) are available only in the professional version, all the properties and functions marked with a hash sign (#) are available only in the Form Module, all the properties and functions marked with a caret (^) are available only in the Annotation Module, and the properties and functions marked with an ampersand (&) are only available in Signature Module.

Foxit PDF SDK ActiveX runs on Windows 95/NT or later. This is a standalone component and does not require any extra PDF software installed. Please note a user might need to have administrator rights to register the ActiveX under Windows successfully.

There are several complete demo programs written in different languages, including Visual Basic, Visual C++, Delphi, showing how to use the properties and methods of the ActiveX. To download these methods please visit www.foxitsoftware.com.

Foxit PDF SDK ActiveX 5.0 Professional adds many new interfaces to support PDF Signature, which allow users to add, delete, print Signature in their PDF documents. The Signature Module is licensed separately from ActiveX 5.0 Professional. If you would like to use this module, please contact us at sales@foxitsoftware.com.

UNLOCK Code: If you have purchased Foxit PDF SDK ActiveX and received the full version of the ActiveX and the unlock code, you should call UnLockActiveX or UnLockActiveXEx functions once inside your program before you call ANY other functions of the ActiveX. This function is described in the Reference section. You don't need to call this function if you just want to evaluate the ActiveX.

GUID for standard version: 0F6C092B-6E4C-4976-B386-27A9FD9E96A1

GUID for professional version: F53B7748-643C-4A78-8DBC-01A4855D1A10

Tutorials

The Foxit SDK ActiveX control comes with a single OCX file. To install it, please use command "regsvr32 OCX name". You may need to specify the proper path if OCX is not stored in current directory.

The ActiveX control handles user interface for you. It also supplies multiple properties and methods so that your application is able to control the ActiveX.

The following is an example, suppose we had already created an ActiveX control called FoxitReaderSDK.

Std & Pro Version

1. Open a PDF File

We will open a PDF document named “testdoc.pdf”

```
// NOTE: If you are evaluating ActiveX, you don’t need to unlock it,
// then evaluation marks will be shown on all PDF pages.
// For paid customers, please unlock the ActiveX first.
FoxitReaderSDK.UnLockActiveX(“license_id”,”unlock_code”);
FoxitReaderSDK.OpenFile(“testdoc.pdf”,””);
```

2. Go to a specific page

We will go to the third page of the document

```
FoxitReaderSDK.GoToPage(2). //The index of first page is 0.
```

3. Zoom a page

If you want to show a PDF page with its original size, you can use the following code in VC:

```
FoxitReaderSDK.SetZoomLevel(0);
```

Or

```
FoxitReaderSDK. SetZoomLevel(100);
```

If you want to set the zoom factor to 200%, use the following code:

FoxitReaderSDK. SetZoomLevel(200);

For more information, see Function Reference section of this guide.

4. Rotate a page

A PDF page can be displayed in four directions: upright, rotated 90 degree, rotated 180 degree, or rotated 270 degree. To display it in different direction, you only need to call “SetRotate” () to change the Rotate property.

If you want to rotate the page (90 degrees) clockwise, you can use the following code in VC:

FoxitReaderSDK. SetRotate(1);

5. Print a PDF document

What you need to do is call “PrintWithDialog” method, a print dialog will pop up and you will be able to specify parameters and then print out the PDF document. If you want to print without popping up a dialog, you need to use the IPDFPrinter interface.

6. Hide or show UI elements

You can call “ShowToolBar” to show or hide the toolbar. Likewise, you may call “ShowStatusBar” to show or hide the status bar. If you prefer to build your own toolbar, you may hide the built-in ActiveX toolbar and then create your own outside the ActiveX.

7. Iterate the whole outline tree

You can call “GetOutlineFirstChild” or “GetOutlineNextSibling” to iterate the whole outline tree, allowing you to view the outline information from IPDFOutline Interface.

8. Search a PDF document

You can call “FindFirst” to find the first instance of the given text in the whole document. If no occurrence is found, then the function returns 0. If an occurrence is found, the function returns a nonzero value and the occurrence will be

highlighted. Select “FindNext” again to search for the next occurrence.

If you want to search for text inside PDF files without opening and displaying them, you may use the “FindFileFirst” or “FindFileNext”.

Signature Module

Developers can digitally sign PDF documents using third-party digital certificates located in the Windows system certificate store. It also supports digital signature verification (which authenticates the source and verifies the integrity of content of digitally signed PDF documents), signature timestamp, signature print control, and more.

Annotations Tools

End users can draw lines, circles and other shapes on a PDF document by using different markup tools. Your application may also change “CurrentTool” property programmatically, for example, to the “Line Tool”.

In version 4.0 or later, Foxit ActiveX provides some interfaces to operate Annotation objects. You can build your own annotations with these interfaces.

Form Application

In version 3.0, Foxit ActiveX provides an interface for PDF forms. By calling “GetCurrentForm”, you can obtain the IForm interface pointer for the current PDF document. Then you can use properties and methods in IPDFForm and IPDFFormField interface classes to perform actions to PDF forms. The following code shows how to add a form button in a PDF page:

```
CPDFformField      button1 =
form1.AddField("button1","button",m_nCurPage,0,0,55,30);
button1.SetButtonCaption("N","Normal");
button1.SetButtonCaption("R","Rollover");
button1.SetButtonCaption("D","Down");
button1.SetBehavior("push"); // push;Invert; NULL; Outline;
button1.SetTooltip("reset all form");
button1.SetTextFont("Courier");
button1.SetTextSize(15);
button1.SetJavaScriptAction("down","app.alert(\"Mouse Down!\")");
```

IFoxitPDFSDK

This section describes all properties and methods exposed by ActiveX. Please note that the reference shows everything in C syntax. If you use a programming language other than C/C++, you have to follow the syntax of that language.

Note: The functions marked with an asterisk (*) only apply to the professional version. The functions marked with a hash sign (#) are available only in the Form Module and the functions marked with a caret (^) are available only in the Annotation Module.

Properties

1) FilePath

Type:

BSTR, read-only

Description:

Full path of the currently opened PDF document. If no PDF file is opened or the file is opened from the buffer or stream, the value is an empty string.

2) Password

Type:

BSTR, read-only

Description:

Password of a PDF document.

3) PageCount

Type:

long, read-only

Description:

Total number of pages in the current open PDF file

4) CurPage

Type:

long, read-only

Description:

The index of the current PDF page. Page index starts from zero for the first

page.

5) Rotate

Type:

short, read and write

Description:

Current rotate orientation, the value can be one of the following:

0: (Normal);

1: (Rotated 90 degrees clockwise);

2: (Rotated 180 degrees);

3: (Rotated 90 degrees counter-clockwise).

6) Zoomlevel

Type:

long, read and write

Description:

Normally, the value of this zoom factor is between 10 and 1600. You may also use the following special values:

0: Displaying the page in actual page size, this is the same as setting zoom level to 100%.

1: Displaying the page with proper zoom level so that the whole page can be fit into the client window.

2: Displaying the document with proper zoom level so that the width of the page fit to the client window.

7) CurrentTool

Type:

BSTR, read and write

Description:

Read and set the current tool. The value can be set to one of following strings:

“Hand Tool”

“ZoomOut Tool”

“ZoomIn Tool”

“Select Text Tool”

“Find Text Tool”

“Snapshot Tool”

*“Loupe Tool”
*“Magnifier”
*“Annot Tool”
*“Rectangle Link Tool”
*“Quadrilateral Link Tool”
*“Arrow Tool”
*“Line Tool”
*“Dimension Tool”
*“Square Tool”
*“Rectangle Tool”
*“Circle Tool”
*“Ellipse Tool”
*“Polygon Tool”
*“Cloudy Tool”
*“Polyline Tool”
*“Pencil Tool”
*“Rubber Tool”
*“Highlight Tool”
*“Underline Tool”
*“Strikeout Tool”
*“Squiggly Tool”
*“Replace Tool”
*“Note Tool”
*“Push Button Tool”
*“Check Box Tool”
*“Radio Button Tool”
*“Combo Box Tool”
*“List Box Tool”
*“Text Field Tool”
*“Distance Tool”
*“Perimeter Tool”
*“Area Tool”
*“Typewriter”
*“CallOut”
*”Textbox”
*“Image Tool”
*“Sound Tool”
*“Movie Tool”

*“FileAttachment Tool”

*“Attach a file”

And so on.

Note:

You can call CountTools to learn how many tools are available in the current version of ActiveX, and then call GetToolByIndex to get the tool names.

8) Printer

Type:

IPDFPrinter, read-only

Description:

Printer property returns an IPDFPrinter interface that you can use for managing the printer and sending the printout.

9) DocumentInfo

Type:

IPDFDocumentInfo*, read-only

Description:

DocumentInfo property returns an IPDFDocumentInfo interface with which you can use to retrieve PDF document information, such as Author, Creator, Creation Date, Keywords, ModDate, Producer Subject and Title.

10) ActiveXVersion

Type:

BSTR, read-only

Description:

Get the Version info of ActiveX control.

11) *bHasFormFields

Type:

BOOL, read-only

Description:

If the current document contains form fields, the value of bHasFormFields is TRUE. Otherwise, it is FALSE.

12) *bHighlightFormFields

Type:

BOOL, read and write

Description:

Setting the value to True will highlight all interactive form fields to get better visual effects.

13) *FormFieldsHighlightAlpha

Type:

short, read and write

Description:

Represent 256 levels of transparency of form field highlight color.

0: transparent;

255: opaque.

14) *FormFieldsHighlightColor

Type:

OLE_COLOR, read and write

Description:

The highlight color of form fields.

Method

9. Unlock

1) UnLockActiveX

Unlock the ActiveX using license key received from Foxit Corporation.

Prototype:

Void UnLockActiveX(BSTR license_id, BSTR unlock_code)

Parameters:

license_id: A string received from Foxit to identify the SDK licensee.

unlock_code: A string received from Foxit to unlock the ActiveX.

Return Value:

[None]

Note:

When evaluating ActiveX, you don’t need to call this function and the

evaluation marks will be shown on all rendered pages. After getting the license key, you could call this function to unlock ActiveX so that you will not get the evaluation mark.

2) UnLockActiveXEx

Unlock the ActiveX using license key received from Foxit Corporation.

Prototype:

Void UnLockActiveXEx(BSTR strLicense)

Parameters:

strLicense: A string received from Foxit to identify the SDK license key

Return Value:

[None]

Note:

This function performs the same function as the UnlockActiveX.

3) *RemoveEvaluationMark

Remove the Foxit evaluation mark once if you have unlocked ActiveX with a license key.

Prototype:

BOOL RemoveEvaluationMark()

Parameters:

[None]

Return Value:

Returns TRUE if successful, FALSE if failed.

10. Global Settings

1) SetCurrentLanguage

Switch to the language you want by ID once if there is a corresponding XML file.

Prototype:

void SetCurrentLanguage(short LanguageID);

Parameters:

LanguageID: Language identifier. A value from 0 to 30 represents a different language.

1 - Arabic

2 - Bulgarian
3 - Hungarian
4 - Catalan
5 - Czech
6 - Chinese-Simplified
7 - Chinese-Traditional
8 - Danish
9 - Dutch
10 - English
11 - Estonian
12 - Finnish
13 - French
14 - Galician
15 - German
16 - Greek
17 - Italian
18 - Korean
19 - Latvian
20 - Lithuanian
21 - Norwegian
22 - Polish
23 - Portuguese
24 - Portuguese_Brazilian
25 - Romanian
26 - Russian
27 - Slovenian
28 - Spanish
29 - Swedish
30 - Turkish
31 - Hebrew
32 - Japanese
33 - Thai
34 - Valencian

Return Value:

[None]

Note:

The user interface of ActiveX can be switched to one of the 30+ languages dynamically. This requires an extra language file (in xml format) to be

accompanied with the ActiveX. If you want a specific language file, please contact us at sales@foxitsoftware.com.

2) SetCurrentLanguageByString

Switch to the language by specifying the path of the language file (XML).

Prototype:

void SetCurrentLanguageByString(BSTR FileName);

Parameters:

FileName - The file path of the language file.

Return Value:

[None]

Note:

The user interface of ActiveX can be switched to one of the 30+ languages dynamically. This requires an extra language file (in xml format) to be accompanied with the ActiveX. If you want a specific language file, please contact us at sales@foxitsoftware.com.

3) SetModulePath

Specify the path of fpdfcjk.bin file so that it could be used when rendering the Chinese, Japanese and Korean text of a PDF.

Prototype:

void SetModulePath(LPCTSTR lpFolderName)

Parameters:

lpFolderName - The path of fpdfcjk.bin file

Return Value:

[None]

Note:

The add-on fpdfcjk.bin is use to help display the Chinese, Japanese and Korean text of a PDF document.

4) SetCrashLog

Set a log file so that all the crash issues you encounter will be recorded in the file.

Prototype:

BOOL SetCrashLog (BSTR filepath);

Parameters:

Filepath: the crash log file's path.

Return Value:

A bool value indicates whether the log file has been set or not.

5) SetLogFile

Set a log file for your application so that each function you called will be recorded to the file.

Prototype:

BOOL SetLogFile(BSTR filepath);

Parameters:

Filepath: the log file's path.

Return Value:

A bool value specifies whether the log file has been set.

11. Open and close PDF File

1) OpenFile

Open a PDF file from a local disk or from an http server.

Prototype:

BOOL OpenFile (BSTR FilePath, BSTR Password)

Parameters:

FilePath: A path of the local PDF file or a URL of the PDF in HTTP server.

Password: Password for opening the PDF.

Return Value:

A nonzero value if the PDF file is successfully opened. Otherwise zero.

Note:

The file will not be locked if it is opened by this method. It can be opened by another program.

2) OpenMemFile

Open a PDF file that is stored in memory.

Prototype:

BOOL OpenMemFile(long pBuffer, long Size, BSTR Password)

Parameters:

pBuffer: Caller-supplied pointer to a buffer containing PDF data .

Size: Size of the buffer pointed to by pBuffer.

Password: Password for opening the PDF.

Return Value:

A nonzero value if the PDF file is successfully opened. Otherwise it is zero.

3) OpenBuffer

Open a PDF file from the buffer.

Prototype:

BOOL OpenBuffer(VARIANT Buffer, long size, BSTR password);

Parameters:

Buffer: Byte array containing the PDF.

Size: Size of the byte array.

Password: Password for opening the PDF.

Return Value:

A nonzero value if the PDF file is successfully opened. Otherwise zero.

4) OpenStream

Open a PDF file from the IStream interface..

Prototype:

BOOL OpenStream (IStream* Stream, BSTR Password)

Parameters:

Stream: A IStream interface.

Password: Password for opening the PDF.

Return Value:

A nonzero value if the PDF file is successfully opened. Otherwise zero.

5) OpenCustomFile

Open a PDF document from a custom access descriptor.

Prototype:

BOOL OpenCustomFile(BSTR Password)

Parameter:

Password: Password for opening the PDF.

Return Value:

A non-zero value if the PDF file has been successfully opened. Otherwise zero.

Note:

When your program calls this method, ActiveX will trigger the CustomFileGetSize and CustomFileGetBlock events. Inside the event handler, your program will open the PDF document from a custom format;

return the file size and block of data. See the description of CustomFileGetSize and CustomFileGetBlock for more details.

6) OpenFtpFile

Open a PDF file from a FTP server.

Prototype:

BOOL OpenFtpFile(BSTR ftpName, BSTR username, BSTR userPassword, long port, BSTR filePath, BSTR filePassword, boolean Passive);

Parameter:

ftpName:	FTP server name
username:	Username for connecting to FTP.
userPassword:	Password for connecting to FTP.
port:	Port on FTP server.
filePath:	File path of the PDF in FTP.
filePassword:	Password for opening the PDF.
Passive:	Passive or active connection.

Return Value:

Value “1” if the PDF file has been successfully opened. Otherwise zero (0).

7) CloseFile

Close the currently loaded PDF file.

Prototype:

Void CloseFile()

Parameter:

[None]

Return Value:

[None]

8) SetFileStreamOption

Set the file stream option when opening the file.

Prototype:

Void SetFileStreamOption(BOOL bFileStream);

Parameter:

bFileStream:	A BOOL value indicating the file stream option.

Return Value:

[None]

Note:

Loading the stream contents into memory will improve performance if the file is frequently accessed. However it will consume more memory.

9) OpenFileAsync

Mainly be used for opening linearized PDF documents. When in B/S architecture, it could help quickly download the data of PDF and render it.

Prototype:

BOOL OpenFileAsync(LPCTSTR strURL, LPCTSTR strPDFPassword, LPCTSTR strUserName, LPCTSTR strUserPassword)

Parameters:

strURL:	URL of a PDF from FTP or HTTP.
strPDFPassword:	Password for opening the PDF if needed.
strUserName:	Username for connecting to FTP if needed.
strUserPassword:	Password for connecting to FTP if needed.

Return Value:

Returns TRUE if successfully perform this function, otherwise FALSE.

12. Navigation

1) ExistForwardStack

Detect the existence of next view.

Prototype:

BOOL ExistForwardStack ();

Parameters:

[None]

Return Value:

If next view exists, then it will return TRUE. Otherwise, it will return FALSE.

Note:

Quite often, when a user navigates through a PDF file, he would like to go back to a previous reading point. View is a concept that defines certain reading point or displaying status. Certain user actions will create new views. For example, if a user turns to a new page, and then zooms in the page, these two actions will create two new views. A program may call this group of methods to allow user to jump among different views conveniently.

2) GoForwardStack

Go to the next view.

Prototype:

Void GoForwardStack ();

Parameters:

[None]

Return Value:

[None]

3) ExistBackwardStack

Detect the existence of previous view.

Prototype:

BOOL ExistBackwardStack ();

Parameters:

[None]

Return Value:

If previous view exists, then it will return TRUE. Otherwise, it will return FALSE.

4) GoBackwardStack

Jump to previous view.

Prototype:

Void GoBackwardStack ();

Parameters:

[None]

Return Value:

[None]

5) SetViewRect

Display a rectangle of current PDF page.

Prototype:

Void SetViewRect (float Left, float Top, float Width, float Height);

Parameters:

Left: The horizontal coordinate of the top left corner.

Top: The vertical coordinate of the top left corner.

Width: The width of the rectangle.

Height: The height of the rectangle.

Return Value:

[None]

Note:

This function will show a rectangle of current PDF page. The coordinate here is PDF coordinate, not device coordinate. And the unit is a PDF point. The function will keep the position and size of ActiveX window unchanged and will adjust the position and the zoom factor of current PDF page so that the designated rectangle of current PDF page will be shown fully inside the ActiveX window. A typical application is: the end user uses the mouse to click and drag a rectangle and then releases the mouse, the program will call ConvertClientCoodToPageCood to convert the mouse coordinates into PDF coordinates and then call SetViewRect to display the specified area in full view.

6) ConvertClientCoordToPageCoord

Converts a point in ActiveX control window's client co-ordinates into PDF page coordinate.

Prototype:

BOOL ConvertClientCoordToPageCoord (long nClientX, long nClientY, long* pnPageIndex, float* pPageX, float* pPageY);

Parameters:

nClientX: X coordinate in the ActiveX control window's client co-ordinates, in pixels

nClientY: Y coordinate in the ActiveX control window's client co-ordinates, in pixels.

pnPageIndex: For returning page number in which the given point falls on.

pPageX: For returning x coordinate of the point inside the PDF page (in PDF co-ordinate system)

pPageY: For returning y coordinate of the point inside the PDF page (in PDF co-ordinate system)

Return Value:

Return value indicates whether the conversion is successful. The client area contains the PDF page being shown as well as some grey background.

7) ConvertClientCoordToPageCoordEx

The extension of ConvertClientCoordToPageCoord function.

Prototype:

BOOL ConvertClientCoordToPageCoordEx (long nClientX, long nClientY, VARIANT* pnPageIndex, VARIANT* pPageX, VARIANT* pPageY);

Parameters:

nClientX: X coordinate in the ActiveX control window's client co-ordinates, in pixels

nClientY: Y coordinate in the ActiveX control window's client co-ordinates, in pixels

pnPageIndex: For returning page number on which the given point falls

pPageX: For returning x coordinate of the point inside the PDF page (in PDF co-ordinate system)

pPageY: For returning y coordinate of the point inside the PDF page (in PDF co-ordinate system)

Return Value:

Return value indicates whether the conversion is successful or not. The client area contains the PDF page being shown as well as some grey background.

8) ConvertPageCoordToClientCoord

Converts PDF page coordinates to the coordinates inside ActiveX control window's client area.

Prototype:

BOOL ConvertPageCoordToClientCoord (long nPageIndex, float dPageX, float dPageY, long* pnClientX, long* pnClientY);

Parameters:

nPageIndex: page number

dPageX: X coordinate inside the PDF page (in PDF co-ordinate system)

dPageY: Y coordinate inside the PDF page (in PDF co-ordinate system)

pnClientX: For returning X coordinate in the ActiveX control window's client area. A negative result indicates that the point is outside the ActiveX control window's client area.

pnClientY: For returning Y coordinate in the ActiveX control window's client area. A negative result indicates that the point is outside the ActiveX control window's client area.

Return Value:

Return value indicates whether the conversion is successful. If the document is not opened properly or if the page number is incorrect, then the return value will be FALSE. Otherwise, the return value will be TRUE.

9) GotoPageDest

Go to a specified position in a PDF document.

Prototype:

Void GotoPageDest (ILink_Dest * link_dest);

Parameters:

link_dest: An ILink_Dest interface you get from event OnHyperLink .

Return Value:

[None]

10) GoToPagePos

Go to a specified position in a PDF document.

Prototype:

Void GoToPagePos (long nPageIndex, float PageX, float PageY);

Parameters:

NPageIndex: Index of the page you want to view.

PageX: Specify the x coordinate of the position inside the PDF page which index is specified by nPageIndex. (in PDF co-ordinate system)

PageY: Specify the y coordinate of the position inside the PDF page which index is specified by nPageIndex.

Return Value:

[None]

11) GetVisibleLeftTopPage

Get the page number of the view at the top left side

Prototype:

Long GetVisibleLeftTopPage ();

Parameter:

[None]

Return Value:

The index of the page which is showed on the left top.

12) ScrollView

Scroll the current view by dx, dy, the unit is device pixel.

Prototype:

Void ScrollView (long dx, long dy);

Parameters:

dx: The horizontal distance of the scrolling action.

dy : The vertical distance of the scrolling action.

Return Value:

[None]

13) GetScrollLocation

Get the current scroll location in current page.

Prototype:

Void GetScrollLocation (long *dx, long *dy);

Parameters:

dx: For returning x coordinate of the current scroll location.

dy: For returning y coordinate of the current scroll location.

Return Value:

[None]

14) GetScrollLocationEx

The extension of GetScrollLocation function.

Prototype:

void GetScrollLocationEx (VARIANT * HPos, VARIANT * VPos);

Parameters:

HPos: For returning x coordinate of the current location you scroll to.

VPos: For returning y coordinate of the current location you scroll to.

Return Value:

[None]

15) GoToPage

Goes to the specific page of the currently opened PDF document

Prototype:

Void GoToPage(long page_index)

Parameters:

page_index: The specific page number

Return Value:

[None]

16) GoToNextPage

Traverse to the next page of the currently opened PDF document.

Prototype:

Void GoToNextPage ();

Parameters:

[None]

Return Value:

[None]

Note:

This function will be removed in further version and you can use function GoToPage instead.

17) GoToPrevPage

Traverse to the previous page of the currently opened PDF document.

Prototype:

Void GoToPrevPage ();

Parameters:

[None]

Return Value:

[None]

Note:

This function will be removed in further version and you can use function GoToPage Instead

13. View

1) AboutBox

Pop up the about box.

Prototype:

Void AboutBox()

Parameters:

[None]

Return Value:

[None]

2) ShowDocumentInfoDialog

Pop up document properties dialog.

Prototype:

Void ShowDocumentInfoDialog()

Parameter:

[None]

Return Value:

[None]

3) SetPDFMeasureUnit

Set the measurement unit of PDF documents.

Prototype:

BOOL SetPDFMeasureUnit(short nType);

Parameters:

nType: Measurement unit and the value could be as follows.

0 = Point;

1 = Inch;

2 = Centimeter;

3 = Pixel

Return Value:

Returns True if successful, False otherwise.

4) GetPageWidth

Get the width of a specific PDF page.

Prototype:

Float GetPageWidth(short nPageIndex);

Parameters:

nPageIndex: The index of a PDF page.

Return Value:

Page width (excluding non-displayable area) which is measured in points. One point is 1/72 inch (around 0.3528 mm).

5) GetPageHeight

Get the height of a specific PDF page.

Prototype:

Float GetPageHeight(short nPageIndex);

Parameters:

nPageIndex: The index of a PDF page.

Return Value:

Page height (excluding non-displayable area) measured in points. One point is 1/72 inch (around 0.3528 mm).

6) CountTools

Count the tools that can be used in the current version of ActiveX.

Prototype:

Short CountTools()

Parameters:

[None]

Return Value:

The total number of the tools in ActiveX.

7) GetToolByIndex

Get the name of a tool.

Prototype:

BSTR GetToolByIndex (short nIndex)

Parameters:

nIndex: The index of a tool. The range of nIndex is: 0 <= nIndex< CountTools().

Return Value:

The name of the tool specified by index.

8) ForceRefresh

Prototype:

Void ForceRefresh()

Note:

This method is to improve efficiency of adding highlight. In the old version, we refresh page once adding a page highlight. If we add several highlights, it will refresh for several times. It causes the decline in efficiency. From version 5.0, in order to improve efficiency, we depart adding highlights and refreshing. That is calling ForceRefresh after user completing adding highlight.

9) *IsDualPage

Check the type of the page.

Prototype:

BOOL IsDualPage(short pageIndex);

Parameters:

pageIndex: The index of a PDF page specified.

Return Value:

Returns a bool value indicating whether the page is two layers. Two layers mean a page has an image with hidden text.

10) *Highlight

Highlight a specified rectangular region on the specified page of this document.

Prototype:

Void Highlight(long nPageIndex, float left, float top, float right, float bottom)

Parameters:

nPageIndex: Number of the page where the specified rectangular region is to be highlighted

left: X-coordinate of the top left corner of the rectangular region

top: Y-coordinate of the top left corner of the rectangular region

right: X-coordinate of the bottom right corner of the rectangular region

bottom: Y -coordinate of the bottom right corner of the rectangular region

Return Value:

[None]

11) *RemoveAllHighlight

Remove all highlights in current PDF.

Prototype:

Void RemoveAllHighlight()

Parameters:

[None]

Return Value:

[None]

14. Hyperlink

1) CountHyperLinks

Count the HyperLinks in a PDF page.

Prototype:

Short CountHyperLinks(short nPageIndex);

Parameters:

nPageIndex: The index of a PDF page.

Return Value:

Returns the link number if successful, 0 for no links, -1 for failure.

2) HighlightHyperLink

Highlight a specific HyperLink in a PDF page.

Prototype:

Void HighlightHyperLink(short nPageIndex, short nLinkIndex)

Parameters:

nPageIndex: the specific page number

nLinkIndex: index of the HyperLink

Return Value:

[None]

3) GetHyperLinkRect

Obtain the position of a specific HyperLink

Prototype:

BOOL GetHyperLinkRect(short nPageIndex, short nIndex, float* top, float* left, float* bottom, float* right)

Parameters:

nPageInde: The index of a specific page.

nLinkIndex: The index of the HyperLink.

top: Returned pointer for top coordinate

left: Returned pointer for left coordinate

bottom: Returned pointer for bottom coordinate

right: Returned pointer for right coordinate

Return Value:

Returns True if successful, False otherwise.

4) GetHyperLinkInfo

Obtain the link information of a specific hyperlink.

Prototype:

BOOL GetHyperLinkInfo(short nPageIndex, short nIndex, BSTR* linktype,BSTR* linkdata, L PDISPATCH* linkdest)

Parameters:

nPageIndex: The specific page number

nLinkIndex: Index of the HyperLink

linktype: Returned pointer containing the type of HyperLink

linkdata: Returned pointer containing the String data of the HyperLink

linkdest: Returned pointer containing the redirection destination of the HyperLink

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

Currently only the link type of GoTo, GoToR, Lanuch and URI can be obtained.

5) EnableHyperLink

Enable or disable the hyperlinks.

Prototype:

Void EnableHyperLink(BOOL bEnable)

Parameters:

bEnable: Value TRUE enables Hyperlinks, FALSE disables them.

Return Value:

[None]

15. Text

1) GetPageText

Extract text content from a PDF page of the currently loaded PDF file.

Prototype:

BSTR GetPageText (long nPageIndex)

Parameters:

nPageIndex: Specify the index of the page where you want to

extract text.

Return Value:

Return the text which was extracted.

2) *GetPageTextW

Extract text content from a page in the currently loaded PDF file.

Prototype:

Long GetPageTextW(long nPageIndex, long FAR* pBuffer, long FAR*nBuflen)

Parameters:

nPageIndex: Number of page you want to extract text from

pBuffer: The buffer to place the desired content of the page

nBuflen: The length of the buffer

Return Value:

Returns -1 if successful, otherwise 0.

3) GetSelectedRectByIndex

Get the selected area by specifying indices.

Prototype:

Long GetSelectedRectByIndex(long nIndex, long* nPageIndex, float* left, float* bottom, float* right, float* top);

Parameters:

nIndex: The index of the currently selected region, the first index is 0.

nPageIndex: The index of the page which contains the selected area.

Left: The left coordinates of the area.

Bottom: The bottom coordinates of the area.

Right: The right coordinates of the area.

Top: The top coordinates of the area.

Returned Value:

Returns TRUE if successful, otherwise FALSE.

4) GetSelectedRectsCount

Get the count of text selected region.

Prototype:

Long GetSelectedRectsCount()

Returned Value:

The number of text selected regions.

5) GetSelectedText

Get currently selected text.

Prototype:

BSTR GetSelectedText ();

Parameters:

[None]

Return Value:

The text that has been selected.

6) GetSelectedTextEx

The extension of GetSelectedText function.

Prototype:

long GetSelectedTextEx(long* pBuffer, long nBuflen);

Parameters:

pBuffer: The input buffer to receive the selected text

nBufLen: The input size of the buffer.

Return Value:

Returns the buffer length of the selected text.

16. Save

1) SaveAs

Save the currently loaded PDF document as a new file.

Prototype:

Void SaveAs (BSTR FileName)

Parameters:

FileName: Specify a local path of a new PDF which the current PDF will be saved as. Only a local path could be supported here.

Return Value:

[None]

2) Save

Save the currently loaded PDF document.

Prototype:

Void Save ()

Parameters:

[None]

Return Value:

[None]

Note:

Only the PDF stored in the local could be saved successfully. The ones opened from URL can't be saved.

3) SaveToStream

Save the currently loaded PDF document into memory.

Prototype:

IStream* SaveToStream ()

Parameters:

[None]

Return Value:

An IStream interface supports reading and writing data to stream objects which contain the PDF file data.

4) *UploadCurFileToFTP

Upload the current PDF file to a FTP server.

Prototype:

BOOL UploadCurFileToFTP(BSTR ftpName, BSTR userName, BSTR userPassword, long port, BSTR FilePath);

Parameter:

ftpName: FTP server name

username: Username for logging to FTP.

userPassword: Password for logging to FTP.

port: Port on FTP server.

filePath: FTP file path.

Return Value:

One (1) if the PDF file is successfully opened, otherwise it is zero (0).

17. Custom UI

1) ShowTitleBar

Show or hide the title bar.

Prototype:

Void ShowTitleBar(BOOL bShow);

Parameters:

bShow: Specify whether to show the title bar.

TRUE - Show the title bar.

FALSE - Hide the title bar.

Return Value:

[None]

2) ShowToolBar

Show or hide the toolbar.

Prototype:

void ShowToolBar(BOOL bShow);

Parameters:

bShow: Specify whether to show the toolbar.

TRUE - Show the toolbar.

FALSE - Hide the toolbar.

Return Value:

[None]

3) ShowToolbarButton

Show or hide the toolbar button.

Prototype:

Void ShowToolBarButton (short nIndex, BOOL bShow)

Parameters:

nIndex: The index of the button.

bShow: Specify whether to show the toolbar button.

TRUE - Show the toolbar button.

FALSE - Hide the toolbar button.

Return Value:

[None]

4) ShowStatusBar

Show or hide the status bar.

Prototype:

Void ShowStatusBar(BOOL bShow);

Parameters:

bShow: Specify whether to show the status bar.

TRUE - Show the status bar.

FALSE - Hide the status bar.

Return Value:

[None]

5) EnableToolTip

Show or hide the tooltips.

Prototype:

BOOL EnableToolTip (BOOL bEnable)

Parameters:

bEnable: Specify whether to show the tooltips.

TRUE - Show the tooltips.

FALSE - Hide the tooltips.

Return Value:

Returns TRUE if successful. Otherwise FALSE.

6) ShowNavPanelByString

Show a navigation panel specified by the name.

Prototype:

BOOL ShowNavPanelByString(LPCTSTR lpszPanelName)

Parameters:

lpszPanelName: The panel name, including “Bookmark” panel, “Pages” panel, “Layer panel”, and “Attachments” panel.

Return Value:

Returns TRUE if successful. Otherwise FALSE.

7) ShowNavigationPanels

Show or hide the navigation panel.

Prototype:

BOOL ShowNavigationPanels(BOOL bShow)

Parameters:

bShow: Specify whether to show the navigation panel.

TRUE - Show the navigation panel.

FALSE - Hide the navigation panel.

Return Value:

Returns TRUE if successful. Otherwise FALSE.

8) GetPanelStatus

Obtain the status of navigation panels.

Prototype:

BOOL GetPanelStatus()

Parameters:

[None]

Return Value:

Returns TRUE if the panel is visible. Otherwise FALSE.

9) SetLayoutShowMode

Set the page layout.

Prototype:

Void SetLayoutShowMode (BrowseMode nShowMode, short nFacingCount);

Parameters:

nShowMode: The value can be set as following:

0 - MODE_SINGLE

1 - MODE_CONTINUOUS

nFacingCount: Number of columns.

Return Value:

[None]

Note:

A PDF document can be displayed as n columns by m rows. No matter what the facing count number is, when the page layout is set to MODE_SINGLE, the ActiveX window will display one row at a time, when the page layout is set to MODE_CONTINUOUS, the window will be able to display adjacent rows at the same time.

10) GetLayoutShowMode

Obtain the current layout mode.

Prototype:

Void GetLayoutShowMode(short* pnShowMode, short* pnFacingCount)

Parameters:

pnShowMode: Current display mode.

PnFacingCount: The number of pages that are displayed horizontally.

Return Value:

[None]

11) SetFacingCoverLeft

Set cover page of a PDF to be displayed on the left when using the facing mode.

Prototype:

Void SetFacingCoverLeft (BOOL bLeft)

Parameters:

bLeft: A BOOL value indicating whether to set cover page to be rendered on the left.

Return Value:

[None]

12) *ShowContextMenu

Show or hide the context menu

Prototype:

void ShowContextMenu(BOOL bShow)

Parameters :

bShow - A value to indicate whether to show context.

Description:

If bShow is TRUE, it can show right click menu, Otherwise it can’t show right click menu.

13) *ShowFormFieldsMessageBar

Show or hide the message bar for form fields in PDF.

Prototype:

void ShowFormFieldsMessageBar(BOOL bShow)

Parameters:

bShow: A value to determine whether to show the message bar for form fields.

TRUE - Show the message bar.

FALSE - Hide the message bar.

Return Value:

[None]

18. Print

1) PrintWithDialog

Display print dialog to set for PDF printing.

Prototype:

Void PrintWithDialog();

Parameters:

[None]

Return Value:

[None]

2) OpenMemFileForPrinter

Print a memory PDF file without displaying it.

Prototype:

IPDFPrinter* OpenMemFileForPrinter(long buffer, long size)

Parameters:

Buffer: The buffer of PDF file.

Size: The size of the buffer.

Return Value:

Interface of IPDFPrinter that you can use to control the printer.

3) OpenFileForPrinter

Print a PDF file without displaying it.

Prototype:

IPDFPrinter* OpenFileForPrinter(BSTR file_path)

Parameters:

file_path: Path to the PDF file (including file extension).

Return Value:

Interface of IPDFPrinter that you can use to control the printer.

19. Search

1) FindFirst

Search the document for a string. If the function finds the first occurrence of the string, it will jump to the page, update CurPage property, highlight the occurrence and return true value. Otherwise it will return false.

Prototype:

BOOL FindFirst (BSTR search_string, BOOL bMatchCase, BOOL bMatchWholeWord)

Parameters:

SearchString:	The string you want to search.
BMatchCase:	Case sensitive or not.
BMatchWholeWord:	Search for whole word only or not.

Return Value:

Nonzero if an occurrence of the string is found, otherwise it will be zero.

2) FindFirstEx

Provide an interface to search for a string in the document. This is the extension of the FindFirst function.

Prototype:

BOOL FindFirstEx(const VARIANT FAR& search_string, BOOL bMatchCase, BOOL bMatchWholeWord)

Parameters:

Search_string:	The string you want to search.
bMatchCase:	Case sensitive or not.
bMatchWholeWord:	Search for whole word only or not.

Return Value:

Nonzero if an occurrence of the string is found, otherwise it will be zero.

3) FindNext

Search for the next occurrence of the given string in the whole document. If ActiveX finds the next occurrence, it will jump to the page, update CurPage property, highlight the occurrence and return the true value. Otherwise, it will return false. Please note that FindNext will use the same searching criteria specified in the FindFirst method, including bMatchCase, bMatchWholeWord. If bSearchDown is true, then the search goes down

the document. If bSearchDown is false, then the search goes up.

Prototype:

BOOL FindNext (BOOL bSearchDown);

Parameters:

bSearchDown: Search down (True) or up (False).

Return Value:

If next occurrence is found, the return value will be non-zero, otherwise it will be zero.

4) FindFileFirst

Search a file for a string and returns an IFindResult interface if it finds the string. Otherwise it will return NULL. This method allows you to search a file without opening it first. For example, if you want to search for a keyword in all the PDF files inside a folder, you may iterate all the PDF files inside that folder and search them one by one for the keyword. If the ActiveX find an occurrence inside a PDF file, it will return IFindResult which contains all the details of the occurrence. Then you can use GoToSearchResult to open the file, jump to the page and highlight the occurrence.

Prototype:

IFindResult* FindFileFirst(BSTR file_path, BSTR search_string,BOOL bMatchCase, BOOL bMatchWholeWord)

Parameters:

file_path: Path to the PDF file.

search_string: The string you want to search.

bmatchCase: Case sensitive or not.

BMatchWholeWord: Search for whole word only or not.

Return Value:

An IFindResult interface if an occurrence is found. Otherwise it will be NULL.

5) FindFileFirstEx

The extension of FindFileFirst function.

Prototype:

IFindResult * FindFileFirstEx(BSTR file_path, BSTR password, VARIANT search_string, boolean bMatchCase, boolean bMatchWholeWord);

Parameters:

file_path: Path to the PDF file.

Password:	The password for the searched PDF file.
search_string:	The string you want to search.
bMatchCase:	Case sensitive or not.
BMatchWholeWord:	Search for whole word only or not.

Return Value:

An IFindResult interface if an occurrence is found. Otherwise it will be NULL.

6) FindFileNext

Search for the next occurrence of the given string in the file specified by FindFileFirst.

Prototype:

IFindResult * FindFileNext ();

Parameters:

[None]

Return Value:

If the next occurrence is found, the return value will be IFindResult. Otherwise it will be NULL.

7) GoToSearchResult

Display and highlight the search result.

Prototype:

Void GoToSearchResult (IFindResult* findresult);

Parameters:

Findresult: An IFindResult interface returned by FindFileFirst, or FindFileNext .

Return Value:

[None]

8) FindPageNext

Find next occurrence on a page.

Prototype:

IFindResult* FindPageNext()

Parameters:

[None]

Return Value:

Returns IFindResult interface if successful, otherwise NULL.

Note:

Calls FindPageFirst with initial search, then calls FindPageNext() to receive the next result of the search.

9) FindClose

Release memory allocated by FindMemFileFirst

Prototype:

Void FindClose()

10) FindPageFirst

Find First Occurrence on a Page

Prototype:

IFindResult* FindPageFirst(long nPageIndex, BSTR search_string,BOOL bMatchCase, BOOL bMatchWholeWord)

Parameters:

nPageIndex:	PDF page index, the index of first page is 0.
Search_string:	The search input.
bMatchCase:	If exactly match the case of the search string
bMatchWholeWord:	If match the whole word of the search string.

Return Value:

Returns IFindResult interface if successful, otherwise NULL.

11) FindMemFileFirst

Search the file in memory.

Prototype:

IFindResult* FindMemFileFirst(VARIANT buffer, long fileSize, BSTR password, BSTR search_string, BOOL bMatchCase, BOOL bMatchWholeWord)

Parameters:

Buffer:	PDF file data
fileSize:	PDF file size
Password:	The password of PDF file
Search_string:	The search input
bMatchCase:	Case sensitive matching
bMatchWholeWord:	Match whole word

12) *SearchAndHighlightAllTextOnPage

Highlight all matches to the search keyword in a specific PDF page.

Prototype:

Void SearchAndHighlightAllTextOnPage(BSTR searchstring, BOOL bMatchCase, BOOL bMatchWholeWord,long PageNo);

Parameters:

PageNo: Number of the page where you want to search.

Return Value:

[None]

20. Bookmark

1) GetOutlineFirstChild

Get the first child item of the current outline.

Prototype:

IPDFOutline* GetOutlineFirstChild(IPDFOutline* Outline)

Parameters:

Outline: The parent item whose first child item will be returned. If you want to get the root item of the outline tree, set NULL as the parameter value.

Return Value:

If the specified item has child items, then the first child item will be returned. Otherwise NULL will be returned.

2) GetOutlineNextSibling

Get next sibling item.

Prototype:

IPDFOutline* GetOutlineNextSibling (IPDFOutline* Outline)

Parameters:

Outline: The outline item whose next sibling will be returned

Return Value:

If the next sibling item exists, it will be returned. Otherwise, NULL will be returned.

21. Security

1) GetDocPermissions

Get file permission flags of the document.

Prototype:

Long GetDocPermissions ()

Parameter:

[None]

Return Value:

A 32-bit integer indicates permission flags. Please refer to PDF Reference for detailed description. If the document is not protected, 0xffffffff will be returned.

2) * CheckOwnerPassword

Check the Owner password.

Prototype:

BOOL CheckOwnerPassword(BSTR lpszPermPsw)

Parameters:

lpszPermPsw: The owner password.

Return Value:

Returns TRUE if correct, otherwise FALSE.

3) *SetOwnerPassword

Set owner password for current PDF.

Prototype:

BOOL SetOwnerPassword(LPCTSTR lpszNewValue)

Parameters:

lpszNewValue: Password string

Return Value:

Returns True if successful, False otherwise.

4) *SetUserPermission

Set user permission for current PDF.

Prototype:

BOOL SetUserPermission(long dwPermission)

Parameters:

dwPermission: User permission flag.

Return Value:

Returns TRUE if successful, otherwise FALSE.

5) *SetUserPassword

Set user password for current PDF.

Prototype:

BOOL SetUserPassword(LPCTSTR lpszNewValue)

Parameters:

lpszNewValue: Password string.

Return Value:

Returns TRUE if successful, otherwise FALSE.

22. Annotation

1) *ExportAnnotsToFDFFile

Export comments from current PDF to a Form Data Format (FDF) file.

Prototype:

BOOL ExportAnnotsToFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName: Path of the FDF file.

Return Value:

Returns TRUE if successful. Otherwise FALSE.

2) *ImportAnnotsFromFDFFile

Import comments from a Form Data Format (FDF) file to current PDF.

Prototype:

BOOL ImportAnnotsFromFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName: Path of the FDF file.

Return Value:

Returns TRUE if successful, otherwise FALSE.

3) *SetBDrawAnnot

Set the annotation flag.

Prototype:

Void SetBDrawAnnot(BOOL bDrawAnnot);

Parameters:

bDrawAnnot: A value to determine whether to draw annotation.

Ture - Can draw annotation.

False - Can't draw annotation.

Return Value:

[None]

4) *ShowAllPopup

Show the popup for all the markups.

Prototype:

Void ShowAllPopup (BOOL bShow);

Parameters:

bShow: The value to determine whether to show the popup.

TRUE - Show the popup.

FALSE - Hide the popup.

Return Value:

[None]

5) ^GetPageAnnots

Obtain a reference to the annotations in a specific PDF page.

Prototype:

IPDFPageAnnots GetPageAnnots(long pageIndex)

Parameters:

pageIndex : The specific page number where to obtain annotations.

Return Value:

Returns an IPDFPageAnnots object that contains references to all of the annotations in the page if successful, otherwise NULL.

6) ^GetFormatTool

Obtain a reference to a FormatTool.

Prototype:

IPDFormatTool GetFormatTool ()

Parameters:

[None]

Return Value:

Returns a reference to a FormatTool if successful, otherwise NULL.

23. Form

1) *ExportFormToFDFFile

Export PDF form data to a Form Data Format (FDF) file.

Prototype:

BOOL ExportFormToFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName: The path of FDF file.

Return Value:

Returns TRUE if successful, otherwise FALSE.

2) *ImportFormFromFDFFile

Import data from a Form Data Format (FDF) file into PDF forms.

Prototype:

BOOL ImportFormFromFDFFile(BSTR FDFFileName)

Parameters:

FDFFileName: The path of FDF file.

Return Value:

Returns TRUE if successful, otherwise FALSE.

3) *FindFormFieldsTextFirst

Search a text in form fields.

Prototype:

BOOL FindFormFieldsTextFirst (BSTR searchstring, BOOL bMatchCase);

Parameters:

Searchstring: The string you want to search.

bMatchCase: Case sensitive or not.

Return Value:

Returns a value to indicate whether a match to the text required exists.

4) *FindFormFieldsTextNext

Continue searching the text in form fields.

Prototype:

BOOL FindFormFieldsTextNext ()

Parameters:

[None]

Return Value:

Returns a value to indicate whether a new match to the text required exists.

5) * SubmitForm

Submit PDF form field to the specific destination.

Prototype:

BOOL SubmitForm(BSTR csDestination)

Parameters:

csDestination: The input string contains the submit URL

Return Value:

Returns TRUE if successful, otherwise FALSE.

6) #GetCurrentForm

Obtain the pointer of the Form interface. This is required when to add new PDF Form elements.

Prototype:

IPDFForm* GetCurrentForm ()

Parameters:

[None]

Return Value:

Returns the Form interface point if successful. Otherwise NULL.

24. Page Organization

1) *InsertPage

Insert one or more pages from a specific PDF to current PDF.

Prototype:

Boolean InsertPage(long nInsertAt, BSTR lpszPDFFileName, BSTR lpszPssword, BSTR lpszPageRangeString);

Parameters:

nInsertAt:	The position to insert the pages.
lpszPDFFileName:	Path of the source PDF whose pages will be used to insert to current PDF.
lpszPssword:	The PDF password.
lpszPageRangeString:	The range of pages to be inserted from the source PDF.

Return Value:

Returns TRUE if successful, otherwise FALSE.

2) *DeletePage

Delete one or more pages from current PDF.

Prototype:

Boolean DeletePage(long pageIndex, long count);

Parameters :

pageIndex: The index of the first PDF page to be deleted.

Count: The total number of the pages to be deleted.

Return Value:

Returns TRUE if successful, otherwise FALSE.

3) *SwapPage

Exchange the positions of two pages in current PDF.

Prototype:

boolean SwapPage(long pageIndex1, long pageIndex2);

Parameters:

pageIndex1: The page index.

pageIndex2: The page index.

Return Value:

Returns TRUE if successful, otherwise FALSE.

4) *SetPageIndex

Set a new position for a specific PDF page.

Prototype:

Boolean SetPageIndex (long pageOldIndex, long pageNewIndex);

Parameters:

PageOldIndex: The current index of the PDF page.

PageNewIndex: A new index of the PDF page.

Return Value:

Returns TRUE if successful, otherwise FALSE.

5) *SetPageCropBox

Crop a PDF page by specifying the crop area.

Prototype:

Boolean SetPageCropBox(long pageIndex, float left, float top, float right,

float bottom);

Parameters:

pageIndex: The page index.

left: The horizontal coordinate of the top left corner.

top: The vertical coordinate of the top left corner.

right: The horizontal coordinate of the bottom right corner.

bottom : The vertical coordinate of the bottom right corner.

Return Value:

Returns TRUE if successful, otherwise FALSE.

6) * SetPageRotate

Rotate a page specified in current PDF.

Prototype:

Boolean SetPageRotate(long pageIndex, long rotate);

Parameters:

pageIndex: The page index.

rotate: Degree of clockwise rotation

Return Value:

Returns TRUE if successful, otherwise FALSE.

7) *FlattenPage

Flatten a page in current PDF.

Prototype:

Boolean FlattenPage(long pageIndex);

Parameters:

pageIndex: Page index

Return Value:

Returns TRUE if successful, otherwise FALSE.

8) *InsertNewPage

Insert a new blank page to the specific position of current PDF.

Prototype:

Boolean InsertNewPage(long nPageIndex, long nPageWidth, long nPageHeight);

Parameters:

nPageIndex: The page index

nPageWidth: The page width

nPageHeight: The page height

Return Value:

Returns TRUE if successful, otherwise FALSE.

9) *ExportPagesToPDF

Export some pages from current document into a specific PDF file.

Prototype:

BOOL ExportPagesToPDF(BSTR lpszPDFFileName, BSTR lpszPageRangeString);

Parameters:

lpszPDFFileName: The file path to insert the pages.

lpszPageRangeString: The range of the pages which will be exported from current PDF, such as "0", "2", "3-5". Please note that "5-2" is invalid. To sum up, the value before the dash should be less than the one after the dash.

Return Value:

A bool value to indicate whether the pages have been exported.

25. JavaScript

1) * RunJScript

Execute a Java Script.

Prototype:

Void RunJScript(LPCTSTR csJS)

Parameters:

scJS: Specifies a Java Script to be executed.

2) *ShowDocJsDialog

Pop up the document JavaScript dialog.

Prototype:

Void ShowDocJsDialog();

Parameters:

[None]

Return Value:

[None]

3) *ShowJsConsoleDialog

Pop up JavaScript Console Dialog.

Prototype:

Void ShowJsConsoleDialog();

Parameters:

[None]

Return Value:

[None]

26. Multi-Instances

1) SetCurrentWnd

By the use of this function, users can set the current instance when ActiveX runs in Multi-instance.

Prototype:

Void SetCurrentWnd(long hWnd);

Parameters:

hWnd: The HWND of a OCX instance.

Return Value:

[None]

2) GetCurrentWnd

Obtain the pointer to the current window.

Prototype:

Long GetCurrentWnd();

Parameters:

[None]

Return Value:

Returns the window point if successful, otherwise Null.

3) GetCtrlInstance

Obtain the handler to the Control Instance.

Prototype:

Long GetCtrlInstance ();

Parameters:

[None]

Return Value:

Returns the Control Instance handler if successful, otherwise NULL.

27. Signature

1) &GetPDFSignatureMgr

Prototype:

IPDFSignatureMgr* GetPDFSignatureMgr();

Return Value:

The type of PDFSignatrueMgr.

2) &GetLastSigModuleError

Get the error index of the latest Signature Module.

Prototype:

long GetLastSigModuleError()

Return Value:

The error index of the latest Signature Module

0	-	Successful
1	-	Parameters error
2	-	Status error
3	-	Execution error
4	-	Certificate does not exist

3) &GetLastSigModuleErrMsg

Get the latest error message from the signature module.

Prototype:

BSTR GetLastSigModuleErrMsg()

Return Value:

The latest error message from the signature module.

28. Others

1) *AddImageObject

Insert an image into the document.

Prototype:

BOOL AddImageObject (long nPageIndex, float left, float bottom, float

width, float height, BSTR BmpFileName, short alpha, short rotate);

Parameters:

nPageIndex: The index of the PDF page to which the image will be added.

Left: The horizontal X position in which the image is placed. The value starts from 0 at the left-most pixel.

Bottom: The horizontal y position in which the image is placed. It starts from 0 at the bottom-most scan line.

Width: The width of the bitmap.

Height: The height of the bitmap.

BmpFileName: The file path of the image.

Alpha: A number from 0 to 255, identifying the alpha value.

Rotate: The angle through which the image are to be rotated.

Return Value:

Returns value indicates whether the image is added.

2) *GetBitmap

Render contents in a page as a bitmap.

Prototype:

Long GetBitmap(short nPageIndex, long pixelWidth, long pixelHeight, float rectLeft, float rectTop, loat rectRight, float rectBottom, long PixelFormat);

Parameters:

nPageIndex: The page number of the document.

pixelWidth: The width of the bitmap.

PixelHeight: The height of the bitmap.

RectLeft: Left pixel position of the display area in the device coordination.

RectTop: Top pixel position of the display area in the device coordination

rectRight: Right pixel position of the display area in the device coordination.

RectBottom: Bottom pixel position of the display area in the device coordination.

PixelFormat: The pixel format of the bitmap.

Return Value:

The handler of the bitmap.

3) *AddWaterMark

Insert a text as watermark into the document.

Prototype:

BOOL AddWaterMark (short page, BSTR string, float left, float bottom, short fontsize, OLE_COLOR fontcolor, short textmode, short alpha, short rotate);

Parameters:

Page: The page number of the document to be Added watermark.

String: The text that is displayed as watermark.

Left: The horizontal X position in which the watermark is placed.

Bottom: The horizontal y position in which the watermark is placed

Fontsize: The text size.

Fontcolor: The text color.

Textmode: The value must be 0, 1, 2.

0 means to fill the text,

1 means to stroke the text,

2 means to fill then stroke the text.

alpha: A number from 0 to 255, identifying the alpha value.

rotate: The angle through which the watermarks are to be rotated.

Return Value:

Returns value indicates whether the watermark is added.

4) * GetBarcodeBitmap

Encode strings into barcode bitmap

Prototype:

Long GetBarcodeBitmap(BSTR contents, short format, long moduleHSize, long moduleVSize, short ecLevel);

Parameters:

Contents: Contents need to be encoded

Format: We support these coding formats

UNSPECIFY = -1,

CODABAR = 0

CODE_39 = 1

CODE_93 = 2

CODE_128 = 3
EAN_8, = 4
UPC_A, = 5
EAN_13, = 6
ITF, = 7
PDF_417, = 8
RSS_14, = 9
RSS_EXPANDED, = 10
UPC_E, = 11
DATAMATRIX, = 12
QR_CODE = 13

moduleHSize: Model unit width

moduleVSize: Model unit height

ecLevel: Merely effective to the QR Code type

ECLEVEL_L = 0
ECLEVEL_M = 1
ECLEVEL_Q = 2
ECLEVEL_H = 3

Return Value:

2D image if successful, otherwise NULL.

5) * ConvertPDFPageToImage

Convert a specified PDF page to JPG Image.

Prototype:

BOOL ConvertPDFPageToImage(LPCTSTR pdfFile, long lnPageIndex, LPCTSTR imageFilePath, long lnImageWidth, long lnImageHeight)

Parameters:

pdfFile : The path of a PDF whose page will be converted.

lnPageIndex: The index of a PDF page specified.

imageFilePath: The file path of the image file converted.

lnImageWidth: Width of the converted image.

lnImageHeight: Height of the converted image.

Return Value:

Returns TRUE if successful, otherwise FALSE.

6) *ImportImageToPdf

Import an image to a specific PDF.

Prototype:

BOOL ImportImageToPdf(LPCTSTR pdfFilePath, long index, LPCTSTR imageFilePath)

Parameters:

pdfFilePath: The path of a PDF where the image will be imported.

index: The index of the PDF page to which the image file will be imported.

imageFilePath: The path of an image file which will be imported to a PDF page.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Event

1) BeforeDraw

Triggered before the painting of the viewer contents is about to begin.

Prototype:

Void BeforeDraw (long dc)

Parameters:

Dc: Handle to a device context.

2) AfterDraw

Triggered after the painting of the viewer contents has been completed.

Prototype:

Void AfterDraw (long dc)

Parameters:

dc: Handle to a device context.

3) OnZoomChange

Triggered when the Zoomlevel property has been changed.

Prototype:

Void OnZoomChange ()

Parameters:

[None]

4) OnPageChange

Triggered when you change a page (move from one page to another).

Prototype:

Void OnPageChange ()

Parameters:

[None]

5) OnOpenPassword

Triggered when you try to open a PDF document which is password protected.

Prototype:

Void OnOpenPassword (BSTR* password, BOOL* cancel)

Parameters:

Password: Password for opening the PDF.

Cancel: If cancel is set as False, it will be triggered all the time, until the password is correct.

6) OnSearchProgress

Triggered when you search document.

Prototype:

Void OnSearchProgress (long pageNumber, long pageCount)

Parameters:

pageNumber: The page currently being searched,

pageCount : The total number of pages .

7) OnOpenFile

Event triggered when file open operation fails.

Prototype:

Void OnOpenFile(short Error);

Parameters:

Error: Returns the error code.

8) OnOpenDocument

Event triggered when you open a document.

Prototype:

Void OnOpenDocument (BSTR filepath)

Parameters:

Filepath: The file path of a PDF.

9) OnFilePathInvalidate

Event triggered when file operation fails to validate.

Prototype:

Void OnFilePathInvalidate(BSTR WarnString);

Parameters:

WarnString: The error message.

10) OnShowSavePrompt

Event triggered when closing the modified file.

Prototype:

void OnShowSavePrompt(BOOL* bShow, short * nResult);

Parameters:

bShow: The value indicates whether to show the default message box in ActiveX.

nResult: The value indicates whether to save the modified file.

11) OnCloseDocument

Event triggered when you close a document.

Prototype:

Void OnCloseDocument (BSTR filepath)

Parameters:

Filepath: The file path of a PDF.

12) OnDocumentChange

Triggered when the PDF document content changes.

Prototype:

Void OnDocumentChange ()

Parameters:

[None]

13) CustomFileGetSize

Triggered when you use OpenCustomFile method to open PDF document.

Prototype:

Void CustomFileGetSize (long* size)

Parameters:

Size: [out] Pointer to number that will receive the PDF length. Set it

to PDF file length.

14) CustomFileGetBlock

Triggered when you use OpenCustomFile method to open PDF document.

Prototype:

Void CustomFileGetBlock (long pos, long pBuf, long size)

Parameters:

Pos: [in] Byte offset from beginning of the file.

pBuf: [out]Pointer to the buffer that will receive the pdf data.

Size: [in] the buffer size.

Note:

Get a block of data from specific position. Position is specified by byte offset from beginning of the file. The position and size will never go past the range of file length.

15) OnClick

Triggered when the left button is clicked.

Prototype:

Void OnClick (long hWnd, long ClientX, long ClientY);

Parameters:

hWnd: The handle of this window.

ClientX: X coordinate in the ActiveX control window's client area.

ClientY: Y coordinate in the ActiveX control window's client area.

16) OnDbClick

Triggered when the left button is double clicked.

Prototype:

Void OnDbClick (long hWnd, long ClientX, long ClientY);

Parameters:

hWnd: The handle of this window.

ClientX: X coordinate in the ActiveX control window's client area.

ClientY: Y coordinate in the ActiveX control window's client area.

17) OnRButtonClick

Event triggered when clicking the right mouse button.

Prototype:

Void OnRButtonClick(long hWnd, long ClientX, long Client);

Parameters:

hWnd: The handle of this window.

ClientX: X coordinate in the ActiveX control window's client area.

ClientY: Y coordinate in the ActiveX control window's client area.

18) OnDownLoadFinish

Event triggered when downloading from the Internet succeeds.

Prototype:

Void OnDownLoadFinish();

Parameters:

[None]

19) OnErrorOccurred

Event triggered when errors occur on the called SDK interfaces. Only methods GetToolByIndex and ShowToolbarButton are supported now.

Prototype:

Void OnErrorOccurred(BSTR lpszErrorMsg)

Parameters:

lpszErrorMsg: The error message.

Return Value:

[None]

20) OnUploadFinish

Event triggered when errors occur on the called UploadCurFileToFTP.

Prototype:

Void OnUploadFinish(short nRetCode)

Parameters:

nRetCode: The error code.

Return Value:

[None]

21) OnTextHyperLink

Event triggered when user clicks the text link in ActiveX. It will return the link text to user and let the user to decide whether to trigger this link or not.

Prototype:

Void OnTextHyperLink(BSTR csUrl, boolean* cancel)

Parameters:

csUrl: The URL of the text link.

cancel: Whether to behavior this link

Return Value:

[None]

22) OnExcuteMenuItem

Event triggered when the user executes an action of the customized menu item that is added by the event OnAddmenuItemAction.

Prototype:

Void OnExcuteMenuItem(BSTR sMenuItem, boolean* bResult)

Parameters:

sMenuItem: Customized menu item

bResult: Obtains the result caused by the action (TRUE or FALSE)

Return Value:

[None]

23) OnDoGoToRAction

Event triggered when the user executes the action GoToR.

Prototype:

void OnDoGoToRAction(BSTR sFilePath, Link_Dest* dest)

Parameters:

sFilePath: The targeted file path

dest: The targeted object in the file

Return Value:

[None]

24) *OnHyperLink

Triggered when clicking on a hypertext.

Prototype:

Void OnHyperLink(BSTR linktype, BSTR linkdata, Link_Dest* dest, BOOL* cancel)

Parameters:

Linktype: A string containing information about the type of hyperlink.

Linktype sting are:

GoTo moves to a different page on the current document, the linkdata is None string, dest contain position information which the

control is about to navigate.

GoToR moves to a different PDF file stored on the local disk, if a new window is required for viewing the new document, the linkdata information contains the filename followed by 1, otherwise, followed by 0.

dest contain position information which the control is about to navigate.

Launch launches an external application, if a window is required for viewing the new document, the linkdata information contains the filename followed by 1, otherwise, followed by 0.

URI open an uri, linkdata contains the uri string.

Cancel: If cancel variable is set to true, the control will not follow the hyperlink.

linkData: A string contains additional information separated by characters.

25) *OnContextMenuIndex

Event triggered when clicking on an entry on the right click contextual menu. Use this event together with the SetContextMenuString interface.

Prototype:

Void OnContextMenuIndex(short nIndex);

Parameters:

nIndex: Index of the selected menu entry.

26) *OnAddMenuItemAction

Event triggered when user adds an “add a menu item” action.

Prototype:

void OnAddMenuItemAction(BSTR* pMenuItem)

Parameters:

pMenuItem: The menu string defined outside the class

Return Value:

[None]

27) #FormFieldError

Event triggered when an error occurs while configuring a PDF form field.

Prototype:

Void FormFieldError(long nErrorCode);

Parameters:

nErrorCode:	Returns the error code when configuring the PDF form field.

ILink_Dest

Method

1) GetPageIndex

Prototype:

Long GetPageIndex();

Return Value:

The page index.

2) GetZoomMode

Long GetZoomMode();

Return Value:

Returns the following definition:

2 - Fit Page

3 - Fit Width

4 - Fit Height

3) GetZoomParamCount

Get the number of Zoom Parameters.

Prototype:

Long GetZoomParamCount()

Return Value:

The number of Zoom Parameters.

4) GetZoomParam

Get the information of Paramater nIndex

Prototype:

double GetZoomParam(long nIndex)

Return Value:

The information of Paramater nIndex, which is the description for Zoom.

5) GetDestName

Get the name of Name Destination object.

Prototype:

BSTR GetDestName() ;

Return Value:

The name of Name Destination object.

IPDFAction

Method

1) GetURIPath

Get the path of URI related to Action.

Prototype:

BSTR GetURIPath();

Return Value:

The Path of URI.

2) GetFilePath

Get the path of the external files related to Action.

Prototype:

BSTR GetFilePath();

Return Value:

The path of the external files.

3) GetType

Get the type for Action.

Prototype:

Int GetType();

Return Value:

Returns the following values:

- 1 - Go to Action;
- 2 - Remote Go-To Actions;
- 4 - Launch Actions;
- 6 - URI Actions.

4) GetDest

Get the destination for Action.

Prototype:

Ilink_Dest* GetDest();

Return Value:

The destination to which action jump.

IPDFOutline

Method

1) GetOutlineDest

Get outline link destination.

Prototype:

ILink_Dest* GetOutlineDest ();

Return Value:

Outline link destination.

2) GetOutlineAction

Get outline associated action

Prototype:

IPDF_Action* GetOutlineAction ();

Return Value:

Outline action

3) GetOutlineColor

Get outline text color (RGB)

Prototype:

DWORD GetOutlineColor ();

Return Value:

The outline color

4) NavigateOutline

Navigate to the destination specified by the outline object.

Prototype:

Void NavigateOutline ();

Parameters:

[None]

Return Value:

[None]

5) GetOutlineTitle

Get the title of the outline object.

Prototype:

BSTR GetOutlineTitle()

Parameters:

[None]

Return Value:

Returns the title of the outline object.

6) GetOutLineTitle2

Get the title of the outline object as a variant.

Prototype:

VARIANT GetOutLineTitle2 ()

Parameters:

[None]

Return Value:

Returns the title of the outline object as a variant

7) GetOutlineExpandValue

Get the number of child node for the outline

Prototype:

Long GetOutlineExpandValue()

Return Value:

The number of child node

IFindResult

Method

1) GetFindRectsCount

Get the count of the find result rectangles.

Prototype:

Long GetFindRectsCount();

Return Value:

Returns positive integer if successful, otherwise 0.

2) GetFindRectByIndex

Get find result rectangle by index. Find Rectangle by Index.

Prototype:

Long GetFindRectByIndex

(longnIndex,float* left, float* bottom, float* right, float* top);

Parameters:

nIndex: The index of the find area.

left: The left vertical coordinates of the find area (PDF coordinate system)

bottom: The bottom horizontal coordinates of the find area(PDF coordinate system)

right: The right vertical coordinates of the find area(PDF coordinate system)

top: The top horizontal coordinates of the find area(PDF coordinate system)

Return Value:

Returns TRUE if successful, otherwise FALSE.

3) GetFindPageNum

Get the page index of the search result.

Prototype:

long GetFindPageNum();

Parameters:

[None]

Return Value:

The page index.

4) GetFindFileName

Get the file name of the search result.

Prototype:

BSTR GetFindFileName();

Parameters:

[None]

Return Value:

The file name.

5) GetFindString

Get the context of the search result.

Prototype:

BSTR GetFindString()

Parameters:

[None]

Return Value:

The context of the search result.

IPDFPrinter

With IPDFPrinter interface you can control the printer and send print-outs.

Properties

1) PrinterName

Type:

BSTR

Description:

Set the printer name that will be used for print-outs.

2) PrinterRangeMode

Type:

PrinterRangeMode

Description:

Set the printer name that will be used for print-outs.

Set the print range, can be set to:

PRINT_RANGE_ALL = 0,

PRINT_RANGE_CURRENT_VIEW = 1,

PRINT_RANGE_CURRENT_PAGE = 2,

PRINT_RANGE_SELECTED = 3

3) PrinterRangeFrom

Type:

short

Description:

Specify the first page number to be printed. You must first set the PrinterRangeMode to PRINT_RANGE_SELECTED

4) PrinterRangeTo

Type:

short

Description:

Specify the last page number to be printed. You must first set the

PrinterRangeMode to PRINT_RANGE_SELECTED

5) NumOfCopies

Type:

short

Description:

Specify the number of copies to be printed.

6) Scaling

Type:

short

Description:

Set the scaling of in the print dialog.

7) AutoRotate

Type:

boolean

Description:

Set the auto-rotate parameter in the print dialog.

8) AutoCenter

Type:

boolean

Description:

Set whether or not to auto-center when printing.

1 = auto-center;

0 = no auto-center.

9) Collate

Type:

boolean

Description:

Set whether or not to check the Collate option in print dialog.

1 = Collate checked;

0 = not checked.

10) Rotation

Type:

short

Description:

Set whether or not to rotate the document when printing.

11) RangeSubset

Type:

short

Description:

Set whether or not to include Subset when printing.

12) ReversePage

Type:

boolean

Description:

Set whether or not to print document in reverse order.

13) PageBorder

Type:

boolean

Description:

Set whether or not to print out page boarders.

Methods

1) PrintWithDialog

Display Windows dialog for sending print-outs.

Prototype:

void PrintWithDialog();

Parameters:

[None]

Returns:

[None]

2) PrintQuiet

Send the printouts to the specified printer without displaying the print dialog.

Prototype:

Void PrintQuiet();

Parameters:

[None]

Returns:

[None]

3) SetPaperSize

Set the paper size for the selected printer, for available paper sizes values print please read Windows SDK documentation.

Prototype:

Void SetPaperSize (long paperSize);

Parameters:

paperSize - Available paper sizes.

Returns:

[None]

4) GetSystemPrinterCount

Get the number of system printer.

Prototype:

Long GetSystemPrinterCount();

Returns:

The number of the printer in the system.

5) GetSystemPrinterNameByIndex

Get the printer name in the system by index.

Prototype:

BSTR GetSystemPrinterNameByIndex(long index);

Returns:

The printer name in the system by index

6) SetPaperSizeByPage

Check the option “Choose Paper Source by PDF Page Size” for printing,

including silent printing and the one with the print dialog.

Prototype:

void SetPaperSizeByPage(BOOL bPage);

Parameters:

bPage: TRUE if checking the option, otherwise FALSE.

IPDFDocumentInfo

This chapter gives the information about the document properties.

Properties

1) Author

Type:

BSTR

Description:

Author of the PDF document.

2) Subject

Type:

BSTR

Description:

Subject of the PDF document.

3) CreatedDate

Type:

BSTR

Description:

The date when the PDF was created.

4) ModifiedDate

Type:

BSTR

Description:

The date the last time the PDF was modified.

5) Keywords

Type:

BSTR

Description:

Keywords for the PDF document.

6) Creator

Type:

BSTR

Description:

Creator of the PDF document.

7) Producer

Type:

BSTR

Description:

Producer of the PDF document.

8) Title

Type:

BSTR

Description:

Title of the PDF document.

#IPDFForm

Method

1) #AddField

Add a new form field

Prototype:

LPDISPATCH AddField(LPCTSTR bstrFieldName, LPCTSTR bstrFieldType, long pageIndex, float left, float top, float right, float bottom);

Parameters:

bstrFieldName: The fully-qualified name of the field.

bstrFieldType: Field type for the newly created field.

Valid types are:

text

button

combobox

listbox

checkbox

radio button

pageIndex: The page number (zero-based).

Left - coordinates of the field rectangle.

top - The top coordinates of the field rectangle.

Right - The right coordinates of the field rectangle.

bottom - The bottom coordinates of the field rectangle.

Return Value:

The field object created newly.

Comment:

The field rectangle coordinate measured in rotated page space; that is, [0,0] is always at the left bottom corner regardless of page rotation.

2) #RemoveFieldsByName

Delete a specific form field

Prototype:

Void RemoveFieldsByName (LPCTSTR bstrFieldName);

Parameters:

bstrFieldName: The fully-qualified name of the field to be removed If the field has multiple child annotations, all of them are removed. If multiple fields have the same name, all are removed.

Return Value:

[None]

3) #ExportToFDF

Export certain form elements into an FDF file.

Prototype:

Void ExportToFDF (LPCTSTR bstrFullPath, BOOL bEmptyFields, const VARIANT FAR& arrFields)

Parameters:

bstrFullPath - Full path of the exported FDF file

bEmptyFields - True to export ONLY elements specified in VARIANT& arrField. False to export everything but the elements specified in VARIANT& arrFields.

ArrFields - Form element array to be exported

Return Value:

[None]

4) #ImportFromFDF

Import from an FDF file

Prototype:

void ImportFormFromFDF(LPCTSTR bstrFullPath);

Parameters:

bstrFullPath: Full path of the FDF file.

Return Value:

[None]

5) #GetFieldByIndex

Obtain the pointer for a specific form field

Prototype:

LPDISPATCH GetFieldByIndex(long index)

Parameters:

index: Index of the form field.

Return Value:

On Success, returning the pointer to the form field, NULL otherwise.

6) #GetFieldsCount

Obtain the count of all form fields

Prototype:

Long GetFieldsCount()

Parameters:

[None]

Return Value:

Returns form fields if successful, otherwise -1.

7) #RemoveField

Remove a specific form field.

Prototype:

Void RemoveField(IPDFFormField pFormField)

Parameters:

pFormField : The input form field to be removed

Return Value:

[None]

#IPDFFormField

Properties

1) #Alignment

Type:

String

Description:

Alignment of text inside a textfield(left, center, right)

Note:

For text field only.

2) #BorderStyle

Type:

String

Description:

Style of the form field border.

Note:

For all.

The border style can be set to:

- solid
- dashed
- beveled
- inset
- underline

3) #BorderWidth

Type:

short

Description:

Width of the form field

Note:

For all form types.

4) #ButtonLayout

Type:

short

Description:

The layout appearance of a button. Valid values include:

0 - Text only; the button has a caption but no icon.

1 - Icon only; the button has an icon but no caption.

2 - Icon over text; the icon should appear on top of the caption.

3 - Text over icon; the text should appear on top of the icon.

4 - Icon then text; the icon should appear to the left of the caption.

5 - Text then icon; the icon should appear to the right of the caption.

6 - Text over icon; the text should be overlaid on top of the icon.

Note:

For push button only.

5) #CalcOrderIndex

Type:

short

Description:

Index of the current form field in the CO array

Note:

For all form types.

6) #CharLimit

Type:

short

Description:

Limit of the number of characters in a textfield.

Note:

For text Field Only

7) #DefaultValue

Type:

String

Description:

Default value of the form field

Comment:

For all

8) #IsEditable

Type:

Boolean

Description:

Indicate if the Combo Box is editable

Note:

Combo Box Only

9) #Behavior

Type:

String

Description:

None, Invert, Outline, Push

N (None) - No highlighting.

I (Invert) - Invert the contents of the annotation rectangle.

O (Outline) - Invert the annotation's border.

P (Push) - Display the annotation as if it were being pushed below the surface of the page.

10) #IsHidden

Type:

Boolean

Description:

Whether the form field is hidden.

Note:

For all form types.

11) #IsMultiline

Type:

Boolean

Description:

Whether a textfield is multi-line or single-line.

Note:

Text field only

12) #IsPassword

Type:

Boolean

Description:

Whether to mask the input as password input.

Note:

For Text Field only.

13) #IsReadOnly

Type:

Boolean

Description:

Whether to set a form field as read only.

Comment:

For All

14) #IsRequired

Type:

Boolean

Description:

Whether a field has to be non-empty

Comment:

For Combo box, Radio button, Text Field

15) #Name

Type:

String

Description:

Name of the current form field

Comment:

Read only, For all

16) #NoViewFlag

Type:

Boolean

Description:

Whether or not to show the form element. 1 = hidden; 0 = show.

Note:

For all form types

17) #PrintFlag

Type:

Boolean

Description:

Whether or not to include form element in print-outs. 1 = show; 0 = hidden.

Note:

For all form types

18) #Style

Type:

CString

Description:

Set the shape of checkbox and radio button to:

1) check
2) cross
3) diamond
4) circle
5) star
6) square

Note:

For checkbox, radio button

19) #TextFont

Type:

String

Description:

Font (Reference 1.7 416)

Can be set as:

Courier

Courier-Bold

Courier-Oblique

Courier-BoldOblique

Helvetica

Helvetica-Bold

Helvetica-Oblique

Helvetica-BoldOblique

Symbol

Times-Roman

Times-Bold

Times-Italic

Times-BoldItalic

ZapfDingbats

Note:

For all form types except for Check box, radio box

20) #TextSize

Type:

Short

Description:

Size of the text in form fields

Note:

For all form types except for Check box, radio box

21) #Type

Type:

String

Description:

Type of the Form

Note:

For all form type.

It can be set as:

Text

Button

Combobox

Listbox

Checkbox

radiobutton

22) #Value

Type:

String

Description:

The current value

Note:

Form elements having this property:

1) Text field
2) combo box
3) radio button
4) check box <Yes&Off>
5) List box

23) #Tooltip

Type:

String

Description:

Displayed tooltip

Note:

For all

24) #Orientation

Type:

Short

Description:

The text rotation of the form

Note:

For all

25) #DirtyFlag

Description:

Set it as true if form field is changed, else as false.

Method

1) #PopulateListOrComboBox

Assign values to entries within a Listbox or Combobox

Prototype:

Void PopulateListOrComboBox (const VARIANT& arrItems, const VARIANT& arrExportVal);

Parameter:

arrItem: An array of strings, with each element representing an item name.

arrExportVal: An array of strings, the same size as the first parameter, with each element representing an export value.

Return Value:

[None]

2) #SetBackgroundColor

Set the background color of a form field

Prototype:

VoidSetBackgroundColor (LPCTSTR bstrColorSpace, float redC, float greenM, float blueY, float AlphaK);

Parameter:

bstrColorSpace: Can be one of the following:

transparent, gray, RGB or CMYK color space.

Use T, G, RGB and CMYK for each.

For T and G, redC is required;

For RGB, redC, greenM and blueY are required;

For CMYK, redC, greenM, blueY and AlphaK are required. redC, greenM and blueY are values from 0 to 1.

Return Value:

[None]

3) #SetBorderColor

Set the border color.

Prototype:

Void SetBorderColor (LPCTSTR bstrColorSpace, float redC, float greenM,float blueY, float AlphaK);

Parameter:

bstrColorSpace: Can be one of the following items:

transparent, gray, RGB or CMYK color space.

Use T, G, RGB and CMYK for each. For T and G, redC is required;

For RGB, redC, greenM and blueY are required;

For CMYK, redC, greenM, blueY and AlphaK are required. redC, greenM and blueY are values from 0 to 1.

Return Value:

[None]

4) #SetForegroundColor

Set the foreground color

Prototype:

Void SetForegroundColor (LPCTSTR bstrColorSpace, float redC, float greenM, float blueY, float AlphaK);

Parameter:

BstrColorSpace - Can be one of the following: transparent, gray, RGB or CMYK color space. Use T, G, RGB and CMYK for each. For T and G, redC is required; For RGB, redC, greenM and blueY are required; For CMYK, redC, greenM, blueY and AlphaK are required. redC, greenM and blueY are values from 0 to 1.

Return Value:

[None]

5) #SetButtonCaption

Set the text displayed on a button

Prototype:

Void SetButtonCaption (LPCTSTR bstrFace, LPCTSTR bstrCaption);

Parameter:

bstrFace: A string that specifies the face for which the caption will be used. Valid strings include:

N - Normal appearance

D - Down appearance

R - Appearance for rollover

bstrCaption -The caption for the button.

Return Value:

[None]

6) #SetButtonIcon

Set the icon for a button

Prototype:

Void SetButtonIcon (LPCTSTR bstrFace, LPCTSTR bstrFilePath);

Parameters:

bstrFace: A string that specifies the face for which the caption will be used. Valid strings include:

N — Normal appearance

D — Down appearance

R — Appearance for rollover

BstrFilePath: The full path of the image file to be used as the source of the appearance.

Return Value:

[None]

7) #SetExportValues

When Radio Button and Checkbox are to be exported, set the value to export for different selection (Selected, un-selected, checked, un-checked, etc)

Prototype:

Void SetExportValues (const VARIANT& arrExportVal);

Parameters:

arrExportVal: Array of the export values

Return Value:

[None]

8) #SetJavaScriptAction

Set the action of JavaScript

Ptototype:

Void SetJavaScriptAction (LPCTSTR bstrTrigger, LPCTSTRbstrJavaScript);

Parameters:

BstrTrigger: A string that specifies the trigger for the action. Valid strings include:

up

down

enter

exit

calculate

validate

format

keystroke

bstrJavaScript: The script itself.

Return Value:

[None]

9) #SetResetFormAction

Set the action of the Reset form field

Prototype:

Void SetResetFormAction (LPCTSTR bstrTrigger, long bFlags, const VARIANT& arrFields);

Parameters:

bstrTrigger : A string that specifies which trigger is used for the action.

Valid strings include:

Up - Mouse up

Down - Mouse down

Enter - Mouse enter

Exit - Mouse exit

bFlags : A collection of flags that define various characteristics of the action.

arrFields : Form element array to be exported

Return Value:

[None]

10) #SetSubmitFormAction

Set the action for the Submit form field

Prototype:

Void SetSubmitFormAction (LPCTSTR bstrTrigger, LPCTSTR bstrURL, long bFlags, const VARIANT& arrFields);

Parameters:

bstrTrigger - A string that specifies which trigger is used for the action.

Valid strings include:

up - Mouse up

down - Mouse down

enter - Mouse enter

exit - Mouse exit

bstrURL : A string containing the URL.

bFlags : A collection of flags that define various characteristics of the action.

arrFields : Form element array to be submitted

Return Value:

[None]

11) #GetPageIndex

Obtain the page index of specified field.

Prototype:

Long GetPageIndex()

Parameters:

[None]

Return Value:

Returns the page index of the field

12) #GetRectTop

Obtain the top of specified field.

Prototype:

Float GetRectTop(l)

Parameters:

[None]

Return Value:

Returns the top of the field

13) #GetRectLeft

Obtain the left of specified field.

Prototype:

Float GetRectLeft()

Parameters:

[None]

Return Value:

Returns the left of the field

14) #GetRectRight

Obtain the right of specified field.

Prototype:

Float GetRectRight()

Parameters:

[None]

Return Value:

Returns the right of the field

15) #GetRectBottom

Obtain the bottom of specified field.

Prototype:

Float GetRectBottom()

Parameters:

[None]

Return Value:

Returns the bottom of the field

^IPDFPageAnnots

Method

1) ^GetAnnot

Obtain a pointer to a specific PDF annotation.

Prototype:

CPDFAnnot GetAnnot(long AnnotIndex)

Parameters:

AnnotIndex : Index of the PDF annotation.

Return Value:

Returns a pointer to the specific PDF annotation if successful, otherwise NULL.

2) ^AddAnnot

Add a new PDF annotation

Prototype:

CPDFAnnot AddAnnot (LPDISPATCH AnnotToAddAfter, LPCTSTR SubType, float left, float top, float right, float bottom)

Parameters:

AnnotToAddAfter : Reserved. Set to NULL.

SubType : Subtype of the annotation.

Possible types:

"Cloudy", "Arrow", "Line", "Square", "Rectangle", "Circle", "Ellipse", "Polygon", "PolyLine", "Pencil", "Underline", "Highlight", "Squiggly", "StrikeOut", "Replace", "Caret", "Note", "Typewriter", "Callout", "Textbox", "FileAttachment", "Image", "Movie", "Sound", "Rectangle Link", "Quadrilateral Link".

Left : The left coordinates of the annotation rectangle.

Top : The top coordinates of the annotation rectangle.

Right : The right coordinates of the annotation rectangle.

Bottom: The bottom coordinates of the annotation rectangle.

All coordinates are based on the PDF coordinate space.

Return Value:

The new annotation object if successful, otherwise will be NULL.

3) ^RemoveAnnot

Delete a specific PDF annotation.

Prototype:

Long RemoveAnnot(LPDISPATCH AnnotToRemove)

Parameters:

AnnotToRemove : A reference to the specific PDF annotation that to be removed.

Return Value:

Returns 0 if successful, otherwise -1.

4) ^GetAnnotIndex

Obtain the index of the specific PDF annotation.

Prototype:

Long GetAnnotIndex(LPDISPATCH Annot)

Parameters:

Annot : A reference to the specific PDF annotation

Return Value:

Returns the index of the specific PDF annotation if successful, otherwise will be -1.

5) ^GetAnnotsCount

Obtain the count of all PDF annotations in the current PDF page.

Prototype:

Long GetAnnotsCount()

Parameters:

[None]

Return Value:

Returns the number of PDF annotations.

^IPDFAnnot

Properties

1) ^Thickness

Type:

short. Read and Write.

Description:

The width of PDF annotation border.

Note:

Used by Drawing Annotations, Measure Tools, and any annotations that have a border (i.e. Cloudy, Arrow, Line, Square, Rectangle, Circle, Ellipse, Polygon, Polyline, Pencil, Callout, Textbox, Link).

2) ^BorderStyle

Type:

Short. Read and Write

Description:

Style of the annotation border.

Note:

There are 7 types:

1 Solid - For all types of annotations.

2 Dashed type 1 - For Drawing Annotations only.

3 Dashed type 2 - For Drawing Annotations only.

4 Dashed type 3 - For Drawing Annotations only.

5 Dashed type 4 - For Drawing Annotations only.

6 Cloudy type 1 - For these Annotations only: Cloudy, Polygon, Circle, Square, Rectangle.

7 Cloudy type 2 - For these Annotations only: Cloudy, Polygon, Circle, Square, Rectangle.

Annotations that aren't supported:

Pencil, Underline, Highlight, Squiggly, StrikeOut, Replace, Caret Note, Typewriter, FileAttachment, Image, Movie, and Sound.

3) ^Color

Type:

OLE_COLOR. Read and Write.

Description:

The background color of PDF annotation.

Note:

Used by all annotations.

4) ^LineStartingStyle

Type:

Short. Read and Write.

Description:

The starting style of line.

Note:

Used by the following annotations: Line, Arrow, PolyLine and Callout

There are 10 types:

0 None
1 Square
2 Round
3 Diamond
4 Open
5 Close
6 Butt
7 Open(recersed)
8 Close(recersed)
9 Slash

5) ^LineEndingStyle

Type:

Short. Read and Write.

Description:

The end style of a line.

Note:

Used by the following annotations: Line, Arrow, and PolyLine.

There are 10 types:

0 None

1	Square
2	Round
3	Diamond
4	Open
5	Close
6	Butt
7	Open(recersed)
8	Close(recersed)
9	Slash

6) ^FillColor

Type:

OLE_COLOR. Read and Write.

Description:

The fill color of PDF annotation.

Note:

Used by the following annotations: Cloudy, Arrow, Line, Square, Rectangle, Circle, Ellipse, Polygon, and Polyline.

7) ^Opacity

Type:

Short. Read and Write.

Description:

The opacity value of the PDF annotation.

Note:

Annotations not supported: Link, Movie, and Sound.

8) ^Author

Type:

BSTR. Read and Write.

Description:

The author of a PDF annotation.

Note:

Annotations not supported: Image, Movie, Sound, and Link.

9) ^Subject

Type:

BSTR. Read and Write.

Description:

The subject of this PDF annotation.

Note:

Annotations not supported: Image, Movie, Sound and Link.

10) ^CreationDate

Type:

DATE. Read only.

Note:

The create time of this PDF annotation.

11) ^ModificationDate

Type:

DATE. Read only.

Description:

The date of the last modification made to the PDF annotation.

12) ^Locked

Type:

Boolean. Read and Write.

Description:

Flag indicating whether the annotation is locked.

Note:

Used by all annotations.

13) ^Print

Type:

Boolean. Read and Write.

Description:

Flag indicating whether the annotation can be printed.

Note:

Used by all annotations.

14) ^ReadOnly

Type:

Boolean. Read and Write.

Description:

Flag indicating whether the annotation is read only.

Note:

Used by all annotations.

15) ^Description

Type:

BSTR. Read and Write.

Description:

The desciption of the PDF annotation

Note:

Only can be used for FileAttachment Annotation.

Methods

1) ^GetType

Obtain the type of PDF annotation

Prototype:

BSTR GetType()

Parameters:

[None]

Return Value:

Returns the type of PDF annotation

2) ^GetSubType

Obtain the subtype of PDF annotation

Prototype:

BSTR GetSubType()

Parameters:

[None]

Return Value:

Returns the Subtype of the PDF annotation

3) ^GetContents

Obtain the contents of PDF annotation

Prototype:

BSTR GetContents()

Parameters:

[None]

Return Value:

Returns the contents of the PDF annotation

4) ^SetContents

Set the contents of the PDF annotation

Prototype:

long SetContents(LPCTSTR Contents)

Parameters:

Contents: The contents to be set.

Return Value:

Returns 0 if successful, otherwise -1

5) ^IsPopupOpen

Specify whether the popup box is opened or not.

Prototype:

BOOL IsPopupOpened()

Parameters:

[None]

Return Value:

Returns TRUE if the Popup Box is open, otherwise FALSE.

6) ^SetPopupOpen

Set to open the popup box.

Prototype:

Long SetPopupOpened(BOOL Open)

Parameters:

Open: The input value specifies whether to open the popup box.

Return Value:

Returns 0 if successful, otherwise -1.

7) ^HasPopup

Indicate whether the annotation is a markup type which has a popup box.

Prototype:

BOOL HasPopup()

Parameters:

[None]

Return Value:

Returns TRUE if the annotation has a popup box, otherwise FALSE.

8) ^GetRect

Get the rectangle of the annotation.

Prototype:

Long GetRect(float* pLeft, float* pTop, float* pRight, float* pBottom)

Parameters:

pLeft: The output value for receiving the left value of the rect.

pTop: The output value for receiving the top value of the rect.

pRight: The output value for receiving the right value of the rec.

pBottom: The output value for receiving the bottom value of the rec.

Return Value:

Returns 0 if successful, otherwise -1.

9) ^SetRect

Set the rectangle of the annotation.

Prototype:

long SetRect(float Left, float Top, float Right, float Bottom)

Parameters:

Left: The input value for setting the left value of the rect.

Top: The input value for setting the top value of the rect.

Right: The input value for setting the right value of the rec.

Bottom: The input value for setting the bottom value of the rec.

Return Value:

Returns 0 if successful, otherwise -1.

10) ^SetLinkGoToAction

Set the GoTo Action for a link annotation.

Prototype:

Void SetLinkGoToAction(long nPageIndex, float left, float top, float zoom)

Parameters

nPageIndex: The input page number.

left,top : The position of the windows' upper-left corner..

zoom: The magnified factor of page.

Return Value:

[None]

11) ^SetLinkURLAction

Set the URL Action of the link annotation.

Prototype:

Void SetLinkURLAction(LPCTSTR sURL)

Parameters:

sURL: The uniform resource identifier to go to when the action is executed.

Return Value:

[None]

12) ^DoAction

Perform the link annotation's action, if it has.

Prototype:

long DoAction()

Parameters:

[None]

Return Value:

Returns 0 if successful, otherwise -1.

13) ^HasAction

Indicates whether the link annotation has an action.

Prototype:

BOOL HasAction()

Parameters:

[None]

Return Value:

Returns TRUE if the link annotation has an action, otherwise FALSE.

14) ^GetMarkedState

Obtain the mark state of this annotation.

Prototype:

Long GetMarkedState()

Parameters

[None]

Return Value

The return value could be 0 (unmarked), 1 (marked), or -1 indicating an error.

15) ^SetMarkedState

Set the mark state of this annotation. annotations not supported:Image, Movie,

Sound, and Link.

Prototype:

long SetMarkedState(long state)

Parameters:

state: The input value to set the mark state of the annotation.

Possible values are 0 (unmarked) or 1 (marked).

Return Value:

Returns 0 if successful, otherwise -1.

16) ^GetReviewState

Get the review state of this annotation.

Prototype:

long GetReviewState()

Parameters:

[None]

Return Value:

The return value could be 0, 1, 2, 3, 4, which mean NULL, Accepted, Rejected, Canceled, and Completed, respectively. Otherwise -1 to indicate an error.

17) ^SetReviewState

Set the review state of this annotation. The following annotations are not supported: Image, Movie, Sound, and Link.

Prototype:

long SetReviewState(long state)

Parameters:

State: The input value specifies the review state of the annotation. It could be set to 0, 1, 2, 3 or 4. Each value specifies a review state as follows.

0 = NULL

1 = Accepted

2 = Rejected

3 = Canceled

4 = Completed.

Return Value:

Returns 0 if successful, otherwise -1.

18) ^GetMigrationState

Get the migration state of this annotation.

Prototype:

long GetMigrationState()

Parameters:

[None]

Return Value:

Returns a value which will indicate the migration state of the annotation.

0 = NULL

1 = Not Confirmed

2 = Confirmed

-1 = Others.

19) ^SetMigrationState

Set the migration state of this annotation. The following annotations are not supported: Image, Movie, Sound, and Link.

Prototype:

long SetMigrationState(long state)

Parameters:

state: The input value specifies the migration state of the annotation. It could be set to 0, 1, or 2. Each value specifies a state as follows.

0 = NULL

1 = Not Confirmed

2 = Confirmed.

Return Value:

Returns 0 if successful, otherwise -1.

20) ^SetStartingPoint

For Line and Arrow annotations. Set the start point of the annotation.

Prototype:

long SetStartingPoint(float PointX, float PointY)

Parameters:

PointX: The input value to set the x of the start point.

PointY: The input value to set the y of the start point.

Return Value:

Returns 0 if successful, otherwise -1.

21) ^GetStartingPoint

For Line and Arrow annotations. Get the start point of the annotation.

Prototype:

long GetStartingPoint(float* PointX, float* PointY)

Parameters:

PointX: The output value for receiving the x of the start point.

PointY: The output value for receiving the y of the start point.

Return Value:

Returns 0 if successful, otherwise -1.

22) ^SetEndingPoint

Set the end point of the annotation. Only be used for Line and Arrow.

Prototype:

long SetEndingPoint(float PointX, float PointY)

Parameters:

PointX: The input value sets the x of the end point.

PointY: The input value sets the y of the end point.

Return Value:

Returns 0 if successful, otherwise -1

23) ^GetEndingPoint

Get the end point of the annotation. Only be used for Line and Arrow.

Prototype:

Long GetEndingPoint(float* PointX, float* PointY)

Parameters:

PointX: The output value receives the x of the end point.

PointY: The output value receives the y of the end point.

Return Value:

Returns 0 if successful, otherwise -1.

24) ^SetMediaPoster

Set a poster (image) for annotations that support posters (i.e. Movie and Sound annotations)

Prototype:

long SetMediaPoster(LPCTSTR ImageFilePath)

Parameters:

ImageFilePath: The input image file path.

Return Value:

Returns 0 if successful, otherwise -1.

25) ^SetMultimedia

Set the multimedia content for annotations that support multimedia.

Prototype:

Long SetMultimedia (LPCTSTR FilePath, LPCTSTR ContentType, BOOL Embed, BOOL bShowCtrlBar)

Parameters:

FilePath: The media file path.

ContentType: The MIME type of the media data.

Embed: Indicates whether to embed media in the PDF file.

bShowCtrlBar: Indicates whether to show the control bar. .

Return Value:

Returns 0 if successful, otherwise -1.

26) ^SetLinkQuadPoints

Set Link annotation's appearance.

Prototype:

long SetLinkQuadPoints(long* PointsArray, long PointsCount)

Parameters:

PointsArray: An array of points.

PointsCount: The size of the array must be 4.

Return Value:

Returns 0 if successful, otherwise -1.

27) ^SetPolygonVertices

Set Polygon annotation's appearance.

Prototype:

long SetPolygonVertices(long* PointsArray, long PointsCount)

Parameters:

PointsArray: An array of points.

PointsCount: The size of PointArray.

Return Value:

Returns 0 if successful, otherwise -1.

28) ^SetPencilVertices

Set Pencil annotation's appearance.

Prototype:

long SetPencilVertices(long* PointsArray, long PointsCount)

Parameters:

PointsArray: An array of lines, each line is a point array.

PointCount: The size of PointArray.

Return Value:

Returns 0 if successful, otherwise -1.

29) ^AttachFile

Attach a file to a FileAttachment annotation.

Prototype:

Long AttachFile(LPCTSTR FileName)

Parameters:

FileName: The input file path.

Return Value:

Returns 0 if successful, otherwise -1.

30) ^ GetReplyList

Get annotation replies.

Prototype:

IPDFAnnotReplyList* GetReplyList()

Return Value:

Returns annotation replies if successful, NULL if there is no replies.

31) ^ UpdateAnnotReplies

Update annotation replies.

Prototype:

Void UpdateAnnotReplies(IPDFAnnotReplyList* pReplies)

Parameters:

pReplies: The annotation replies to be updated.

Note:

Update all the replies of the annotation.

Events

1) ^OnAnnotCreated

After creating the annotation, this event will be triggered and it is completed by a third-party.

Prototype:

void OnAnnotCreated(long pageIndex, long annotIndex)

Parameters:

pageIndex: Page index

annotIndex: Annotation index

2) ^OnAnnotDeleted

After the annotation is deleted, this event will be triggered and it is completed by a third-party.

Prototype:

void OnAnnotDeleted(long pageIndex, long annotIndex)

Parameters:

pageIndex: The page index

annotIndex: The annotation index

3) ^OnAnnotModified

After the annotation is edited, this event will be triggered and it is completed by a third-party.

Prototype:

void OnAnnotModified(long pageIndex, long annotIndex)

Parameters:

pageIndex: The page index

annotIndex: The annotation index

4) ^OnAnnotReplyCreated

After the annotation reply is created, this event will be triggered and it is completed by a third-party.

Prototype:

void OnAnnotReplyCreated(long pageIndex, long annotindex,
lpctstr replyNM)

Parameters:

pageIndex: The page index

annotIndex: The annotation index

pReply: The created reply

5) ^OnAnnotReplyDeleted

After the annotation reply is deleted, this event will be triggered and it is completed by a third-party.

Prototype:

void OnAnnotReplyDeleted(long pageIndex,
long annotindex, CPDFAnnotReply* pReply)

Parameters:

pageIndex: The page index

annotIndex: The annotation index

pReply: The annotation reply to delete.

6) ^OnAnnotReplyModified

After the annotation reply is modified, this event will be triggered and it is completed by a third-party.

Prototype:

Void OnAnnotReplyModified(long pageIndex, long annotindex, CPDFAnnotReply* pReply)

Parameters:

pageIndex: The page index

annotIndex: The annotation index

pReply: The modified annotation reply

7) ^OnAnnotRButtonDown

This event is triggered when you right-click the annotation.

Prototype:

void OnAnnotRButtonDown(IPDFAnnot* Annot, float x, float y, BOOL* bDefault);

Parameters:

Annot: The annotation you do right-click.

x: The horizontal coordinate of the annotation (PDF coordinate)

y: The vertical coordinate of the moved annotation (PDF coordinate)

bDefault: True means use the default action of the clicked button, otherwise it means disable button default action.

Return Value:

[None]

8) ^OnAnnotRButtonUp

This event is triggered when you release the right button.

Prototype:

void OnAnnotRButtonUp(IPDFAnnot* Annot, float x, float y, BOOL* bDefault);

Parameters:

Annot: The annotation which is right-clicked.

x: The horizontal coordinate of the moved annotation (PDF coordinate)

y: The vertical coordinate of the moved annotation (PDF coordinate)

bDefault: True means clicking button default action, otherwise it means disable button default action.

Return Value:

[None]

9) ^OnAnnotLButtonDbClick

This event is triggered when you double-click the annotation.

Prototype:

void OnAnnotLButtonDbClick(IPDFAnnot* Annot, float x, float y, BOOL* bDefault);

Parameters:

Annot: The annotation which is double-clicked.

x: The horizontal coordinate of the annotation (PDF coordinate)

y: The vertical coordinate of the moved annotation (PDF coordinate)

bDefault: True means clicking button default action, otherwise it means disable button default action.

Return Value:

[None]

10) ^OnAnnotMoving

This event is triggered while the annotation is moved.

Prototype:

Void OnAnnotMoving(IPDFAnnot* Annot, float x, float y, BOOL* bDefault);

Parameters:

Annot: The annotation which will be moved.

x: The horizontal coordinate of the annotation (PDF coordinate)

y: The vertical coordinate of the moved annotation (PDF coordinate)

bDefault: True means clicking button default action, otherwise it means disable button default action.

Return Value:

[None]

11) ^OnAnnotMouseEnter

This event is triggered while the mouse is moving to the annotation.

Prototype:

void OnAnnotMouseEnter(IPDFAnnot* Annot);

Parameters:

Annot: The entered annotation.

Return Value:

[None]

12) ^OnAnnotMouseExit

This event is triggered while the mouse is moving out of the annotation.

Prototype:

Void OnAnnotMouseExit(IPDFAnnot* Annot);

Parameters:

Annot: The exited annotation.

Return Value:

[None]

^ IPDFAnnotReplyList

Methods

1) ^ GetCount

Get the count of replies

Prototype:

long GetCount()

Return Value:

The count of replies

2) ^ GetItem

Get the reply with the specified index

Prototype:

CPDFAnnotReply GetItem(long nIndex)

Parameters:

nIndex: The reply index

Return Value:

Returns the specified reply if successful, NULL otherwise.

3) ^ Remove

Remove the specified reply

Prototype:

void Remove(long nIndex)

Parameters:

nIndex : the reply index to remove

4) ^ RemoveAll

Remove all replies in replies list

Prototype:

void RemoveAll()

5) ^ Add

Add a new reply.

Prototype:

Void Add(LPCTSTR Creator, LPCTSTR Content, DATE CreationDate, long nIndex)

Parameters:

Creator: Reply creator

Content: Reply content

CreationDate: Reply creation date

nIndex: Reply adding location(Index is from 0). If the index is -1, it will add reply at the end of reply list.

Note:

Suppose that there are N itmes replies in the original replies list, then range of nIndex is[0, N-1].

^IPDFAnnotReply

Methods

1) ^ SetCreator

Set the creator for the annotation reply.

Prototype:

Void SetCreator(LPCTSTR Creator)

Parameters:

Creator: The creator of the annotation reply.

2) ^ GetCreator

Get the creator of the annotation reply.

Prototype:

BSTR GetCreator()

Return Value:

The creator of the annotation reply.

3) ^ SetContent

Set the content for the annotation reply.

Prototype:

Void SetContent(LPCTSTR Content)

Parameters:

Content: The content of the annotation reply.

4) ^ GetContent

Get the content of annotation reply.

Prototype:

BSTR GetContent()

Return Value:

The content of the annotation reply.

5) ^ GetChildren

Get children annotation reply.

Prototype:

IPDFAnnotReplyList* GetChildren()

Return Value:

The children annotation reply if successful, NULL if there is no children reply.

6) ^ GetParent

Get parent annotation reply

Prototype:

CPDFAnnotReply GetParent()

Return Value:

The parent annotation reply if successful, NULL if there is no children reply.

7) ^ GetCreationDate

Get the annotation reply creation date

Prototype:

DATE GetCreationDate()

Return Value:

The annotation reply creation date

8) ^ SetCreationDate

Set the annotation reply creation date

Prototype:

void SetCreationDate(DATE CreationDate)

Parameters:

CreationDate: The annotation reply creation date

9) ^ SetReadonly

Set the read only flag of the annotation reply

Prototype:

void SetReadonly(BOOL bNewValue)

Parameters:

bNewValue: If it is TRUE, the annotation reply will be set as ReadOnly. If FALSE, it will be set as Read&Write

10) ^ GetReplyID

Get the annotation reply unique ID

Prototype:

long GetReplyID()

Return Value:

The unique ID of the annotation reply

^IPDFormatTool

IPDFormatTool is an interface that allows users to format free text annotations (FreeTextAnnot)in PDF applications. The Interface allows a developer to utilize the same formatting tools found in the Foxit Reader annotation toolbar. There are three FreeTextAnnots in ActiveX, Typewriter, Callout, and TextBox. Each FreeTextAnnot maintains independent format data. Please note that when you change the current FreeTextAnnot, the data of IPDFormatTool will also be modified. IPDFormatTool works only for formatting FreeTextAnnot objects and not any other type of annotations.

Method

1) ^SetFontName

Set current font.

Prototype:

Void SetFontName(BSTR FontName)

Parameters:

FontName The input font name.

Return Value:

[None]

2) ^GetFontName

Get current font.

Prototype:

BSTR GetFontName()

Parameters:

[None]

Return Value:

The current font name.

3) ^SetFontSize

Set the font size.

Prototype:

Void SetFontSize(float FontSize)

Parameters:

FontSize: The input font size.

Return Value:

[None]

4) ^GetFontSize

Get current font size.

Prototype:

Float GetFontSize.

Parameters:

[None]

Return Value:

Returns current font size.

5) ^SetFontColor

Set current text color.

Prototype:

Void SetFontColor(OLE_COLOR FontColor)

Parameters:

FontColor: The input font color.

Return Value:

[None]

6) ^GetFontColor

Obtain the current font color.

Prototype:

OLE_COLOR GetFontColor()

Parameters:

[None]

Return Value:

Returns the current font color.

7) ^SetBorderColor

Set the border color.

Prototype:

Void SetBorderColor(OLE_COLOR color)

Parameters:

Color: The input border color.

Return Value:

[None]

8) ^GetBorderColor

Obtain the border color.

Prototype:

OLE_COLOR GetBorderColor()

Parameters:

[None]

Return Value:

Returns the border color.

9) ^SetFillColor

Set fill color.

Prototype:

Void SetFillColor(OLE_COLOR FillColor)

Parameters:

FillColor: The input fill color to be set.

Return Value:

[None]

10) ^GetFillColor

Get fill color.

Prototype:

OLE_COLOR GetFillColor()

Parameters:

[None]

Return Value:

Returns fill color.

11) ^SetFontBold

Set whether to display the text in bold.

Prototype:

Void SetFontBold(BOOL FontBold)

Parameters:

FontBold: The input value specifies whether to display the text in bold .

Return Value:

[None]

Note:

The bold and italic properties can be set when the font is one of the following: Courier\Helvetica\Times Roman.

12) ^GetFontBold

Indicates if the text is bold.

Prototype:

BOOL GetFontBold()

Parameters:

[None]

Return Value:

Returns TRUE if the current is bold , otherwise FALSE.

13) ^GetFontBoldEnable

Indicates if bold can be enabled by the format tool.

Prototype:

BOOL GetFontBoldEnalbe()

Parameters:

[None]

Return Value:

Returns TRUE if bold can be enabled, otherwise FALSE.

14) ^SetFontItalic

Set whether to display the text in italic.

Prototype:

Void SetFontItalic(BOOL FontItalic)

Parameters:

FontItalic: TRUE to display text in italic, otherwise FALSE.

Return Value:

[None]

15) ^GetFontItalic

Indicates if the text is italic.

Prototype:

BOOL GetFontItalic()

Parameters:

[None]

Return Value:

Returns TRUE if the text is italic, otherwise FALSE.

16) ^GetFontItalicEnable

Indicates if the font italic can be set.

Prototype:

BOOL GetFontItalicEnable()

Parameters:

[None]

Return Value:

Returns TRUE if font italic is available, otherwise FALSE.

17) ^SetAlign

Set the alignment of text.

Prototype:

Void SetAlign(AlignStyle Style)

Parameters:

Style: The input alignment style. Can be set as

ASLEFT=0,

ASMIDDLE=1,

ASRIGHT=2

Return Value:

[None]

18) ^GetAlign

Get the alignment style of the text.

Prototype:

AlignStyle GetAlign()

Parameters:

[None]

Return Value:

Returns the alignment style of the text. Possible values are:

ASLEFT =0,

ASMIDDLE =1,

ASRIGHT =2

19) ^SetCharSpace

Set the spacing between characters.

Prototype:

Void SetCharSpace(float CharSpace)

Parameters:

CharSpace: Space between characters.

Return Value:

[None]

20) ^GetCharSpace

Get the spacing between characters.

Prototype:

float GetCharSpace()

Parameters:

[None]

Return Value:

Returns the space between characters.

21) ^SetCharHorzScale

Set the horizontal scale of characters.

Prototype:

Void SetCharHorzScale(float CharHorzScale)

Parameters:

CharHorzScale: The input scale to be set.

Return Value:

[None]

22) ^GetCharHorzScale

Get the horizontal scale of characters.

Prototype:

float GetCharHorzScale()

Parameters:

[None]

Return Value:

Returns the horizontal scale.

&IPDFSignatureMgr

Method

1) &Add

Add a new unsigned signature field.

Prototype:

IPDFSignatureField* Add(long pageIndex, float left, float top, float right, float bottom);

Parameters:

pageIndex: The index of a PDF page where you add a signature field.

left: The starting horizontal coordinate of the signature field rectangle.

top: The starting vertical coordinate of the signature field rectangle.

right: The end horizontal coordinate of the signature field rectangle.

bottom: The end vertical coordinate of the signature field rectangle.

Return Value:

Return a newly created IPDFSignatureField object.

Note:

The original point of PDF file is the left bottom corner.

2) &SignDocument

Sign the unsigned signature field, when the file is signed successfully, it will close and reopen.

Prototype:

boolean SignDocument(LPDISPATCH pSigField, BSTR signedFilePath, boolean bDefault);

Parameters:

pSigField: IPDFSignatureField object to be signed

signedFilePath: File path of signed PDF file

bDefault: Whether to use default signature

Return Value:

Whether the signed is successfully or not

Note:

If the user signs the file with default arithmetic, call this interface to complete signing the file; otherwise, call the following interface to sign PDF file with third-party signing Operation:

1. Get content flow of unsigned file with interface GetSourceBuffer and GetSourceBufferLen, Then sign it with third-party tool. Lastly, call CreateSignedDoc to complete signing.
2. When signed successfully with standard arithmetic the PDF file will reopen, and in the process make the IPDFSignatureField object invalid. you need to get the object once more if you intend to continue operating on the object.

3) &Verify

Verify signature field.

Prototype:

Boolean Verify(LPDISPATCH pSigField, boolean bDefaultVerified);

Parameters:

pSigField:	Specify the signature field to be verified.
bDefaultVerified:	Specify whether to verify the signature field with the default arithmetic. If the value is TRUE, use the default one.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

If user signed the file with default arithmetic, then call this interface to complete verification of the file.

If it is signed with third-party signing arithmetic, then after calling this interface, call the following interfaces to verify PDF file:

First verify the signed file with interface GetSourceBuffer and GetSignedBuffer. Then call SetVerifyResult to pass verification result to ActiveX. Obtain verification result with GetState interface of IPDFSignatureField .

4) &GetCounts

Get the count of signature in the pdf document .

Prototype:

long GetCounts();

Return Value:

The total number of signature fields , including signed fields and unsigned fields.

5) &Get

Get the signature field by specifying the Index.

Prototype:

LPDISPATCH Get(long index);

Parameters:

index: Get the signature field by specifying the Index.

Return Value:

The index associated IPDFSignatureField object

6) &Clear

Clear signature field and associated information, leaving an unsigned signature field.

Prototype:

boolean Clear(LPDISPATCH pSigField);

Parameters:

pSigField: The IPDFSignatureField object needs to cleared.

Return Value:

Whether cleared successfully or not

7) &Remove

Remove unsigned signature field (remove the object)

Prototype:

boolean Remove(LPDISPATCH pSigField);

Parameters:

pSigField: PDFSignatField object which will be removed.

Return Value:

Whether removed successfully or not.

8) &VerifyAll

Verify all signed signature fields.

Prototype:

boolean VerifyAll();

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

On opening a file, if it asks to verify all signature fields, by default there will be no status dialogues.

9) &InitStraddleValue

Initialize Straddle Sign with some initial values

Prototype:

BOOL InitStraddleValue(IPDFSignatureField* pSigField)

Parameters:

boolean: InitStraddleValue(IPDFSignatureField* pSigField);

pSigField: IPDFSignatureField object needs to be signed.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

After calling this interface, call the normal signature procedure to complete straddle sign. It can be a standard signature or a third-party signature.

&IPDFSignatureField

Properties

1) &Reason

Type:

String

Description:

Signature reason

Operation:

Read and Write

2) &Location

Type:

String

Description:

Signature physical location

Operation:

Read and Write

3) &Signer

Type:

String

Description:

Signer

Operation:

Read and Write

4) &Filter

Type:

String

Description:

The filter name of signature field object arithmetic, its default arithmetic is Adobe.PPKLite.

Operation:

Read and Write

5) &SubFilter

Type:

String

Description:

The subFilter name of signature field object arithmetic, its default arithmetic is adbe.pkcs7.detached in foxit.

6) &State

Type:

Short

Description:

Current signature status (if it is signed by default, the status is decided by ActiveX, else it is decided by definition and send it to ActiveX with SetVerifyResult)

0: Unknown signature

1: Pass verification

2: Did not pass verification

3: Not signed

Operation:

Read only

Methods

1) &SetAPOptions

Set appearance display options value of the customized appearance

Prototype:

boolean SetAPOptions(long opts);

Parameters:

Opts: Set appearance display options value of the customizing appearance (0-511)

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The following options are for users to customize appearance, assign them before calling SignDocument

Completely not to show	- 0
Show All	- 511
Show Text	- 0x100L
Show Image	- 0x080L
Show Signer	- 0x040L
Show Location	- 0x020L
Show DN	- 0x010L
Show Time	- 0x008L
Show Reason	- 0x004L
Show Tag	- 0x002L
Show FoxitFlag	- 0x001L

2) &SetAPText

Set text showed in Signature field

Prototype:

boolean SetAPText(BSTR text);

Parameters:

text: Text which will be displayed.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

Set text before calling SignDocument

3) &SetAPImage

Set Image showed in Signature field

Prototype:

boolean SetAPImage(BSTR imageFilePath, boolean bSetMask, OLE_COLOR clrMask);

Parameters:

imageFilePath: Specify the path of the image for signature field.

bSetMask: The value specifies whether to set mask for background image. Please note that the background image must be pure.

clrMask: Set the value for Mask color.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

Set image before calling SignDocument

4) &IsSigned

Check if the signature field is signed or not.

Prototype:

boolean IsSigned();

Return Value:

Returns TRUE if it is signed, otherwise FALSE.

5) &SetSignerDN

Set Signer Distinguished Name

Prototype:

boolean SetSignerDN(BSTR dn);

Parameters:

dn: Signer Distinguished Name

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

Set the name before calling SignDocument.

6) &SetStatusImage

Set Status Icon

Prototype:

boolean SetStatusImage(BSTR imagePath, short sState, short sMode, boolean bRotate, boolean bSetMask, OLE_COLOR clrMask);

Parameters:

imagePath: Status image path.

sState: Signature status. Its value is decided by attribute Status.

sMode: The image display mode, 0 means default image display location (that is top left Corner); 1 means image cover the whole signature field.

bRotate: Specify whether to rotate image when rotating the signature object. It is set not to rotate by default.

bSetMask: Specify whether to set mask for the background image. Please note that the background image must be pure.

clrMask: Set Mask Color Value

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

This icon can't be saved into PDF file. You can only set the image before the status of Signature is changed.

7) &GetPageIndex

Get the signature page index.

Prototype:

Long GetPageIndex();

Return Value:

Get the signature page index

8) &GetSourceBuffer

Get source file which will be signed or validated.

Prototype:

VARIANT GetSourceBuffer();

Return Value:

Source file buffer

Note:

A. Please call SignDocument or Verify and get TRUE first, and then call this interface.

B. The content flow is saved in parray field of VARIVANT. Please get the content flow pointer with parray->pvData in VC. And get content flow array with getArray in JS.

9) &GetSourceBufferLen

The length of source file content flow which will be signed or verified.

Prototype:

long GetSourceBufferLen();

Return Value:

Buffer length

Note:

Please call SignDocument or Verify and get TRUE first, and then call this interface.

10) &GetSignedBuffer

Get signed buffer

Prototype:

VARIANT GetSignedBuffer();

Return Value:

The signed buffer.

Note:

Please call Verify and get TRUE first, and then call this interface.

The content flow is saved in parray field of VARIVANT. Please get the content flow pointer with parray->pvData in VC. And get source array with getArray in JS.

11) &GetSignedBufferLen

Get the length of signed content flow

Prototype:

long GetSignedBufferLen();

Return Value:

The length of content flow

Note:

Please call Verify and get TRUE first, and then call this interface.

12) &CreateSignedDoc

Create signed document which is signed by a third-party (The path is in SignDocument interface)

Prototype:

boolean CreateSignedDoc(VARIANT signedBuf, long length);

Parameters:

signedBuf: Signed content flow

length: The length of signed content flow.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

Please call SignDocument to set save path for signed document and do not use default signature first, and then call this interface. The signedBuf pointer is inpbVal field of VARIANT in VC, and use array as parameters in JS.

After signing successfully, the PDF document will be opened again. The IPDFSignatureField object will become invalid and then it needs to be obtained once more.

13) &SetVerifyResult

Verify third-party signature result

Prototype:

boolean SetVerifyResult(short sResult);

Parameters:

sResult: Customization verify result

0: Un-know signature

1: Pass verify

2: Not pass verify

3: Not be signed

Return Value:

Returns TRUE if successful, otherwise FALSE.

14) &SetCertPath

Set certification content

Prototype:

boolean SetCertPath(BSTR certPath, BSTR pfxPsw);

Parameters:

certPath - Certification path

pfxPsw - Certification password

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The pCertData pointer is inpbVal field of VARIANT in VC, and use array as parameters in JS.

15) &SetCertData

Set certification Data

Prototype:

boolean SetCertData(VARIANT pCertData, long length, BSTR pfxPsw);

Parameters:

pCertData: Certification data

length: The length of certification data

pfxPsw: Certification password

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The pCertData pointer is inpbVal field of VARIANT in VC, and use array as parameters in JS.

16) &SetCertContext

Set certification context

Prototype:

boolean SetCertContext(long pCertContext);

Parameters:

pCertContext: The type of certification context PCCERT_CONTEXT

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The certification is effective when one of these three interfaces is set.

17) &SetAPImageData

Set image data which will be showed in signature field AP.

Prototype:

Boolean SetAPImageData(VARIANT imageDataBuffer, BSTR imageType, long dataSize, boolean bSetMask, OLE_COLOR clrMask);

Parameters:

imageDataBuffer: Image data buffer

imageType: Image type

dataSize: Image Size

bSetMask: Whether set mask or not

clrMask: Mask color

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The pCertData pointer is inpbVal field of VARIANT in VC, and use array as parameters in JS.

18) &SetStatusImageData

Prototype:

Boolean SetStatusImageData(VARIANT imageDataBuffer, BSTR imageType, long dataSize, short sState, short sMode, boolean bRotate, boolean bSetMask, OLE_COLOR clrMask);

Parameters:

imageDataBuffer: Image data buffer

imageType: mage types can be supported, including bmp, jpg, png, gif.

dataSize: Image size.

sState: Signature status, which is decide by Status property.

sMode : Image show mode, 0 means default image display location (that is top left Corner of signature); 1 means image cover the whole signature field.

bRotate: Specify whether to rotate image while rotating signature object. It is set not to rotate by default.

bSetMask: Specify whether to set mask for the background image. Please note that the background image must be pure.

clrMask: Set Mask Color Value

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

The imageDataBuffer pointer is inpbVal field of VARIANT in VC, and use array as parameters in JS.

19) &TurnGray

Turn the signature appearance to gray.

Prototype:

boolean TurnGray(boolean bGray,boolean bCanModify);

Parameters:

bGray: Specify whether to turn the signature appearance to gray.

bCanModify: Specify whether the signature can be modified

Return Value:

Returns TRUE if successful, otherwise FALSE.

20) &TurnBlur

Signature turn blur

Prototype:

boolean TurnBlur(boolean bBlur);

Parameters:

bBlur: Whether turn to blur

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

This interface changes the view of Signature instead of PDF content.

21) &SetVisible

Signature is visible or not

Prototype:

boolean SetVisible(boolean bVisible);

Parameters:

bVisible: visible or not

Return Value:

Returns TRUE if successful, otherwise FALSE.

22) &GrayPrint

Turn the appearance to gray when to print the signature.

Prototype:

boolean GrayPrint(boolean bGrayPrint);

Parameters:

bGrayPrint: Specify whether to turn the signature to gray for printing.

Return Value:

Returns TRUE if successful, otherwise FALSE.

Note:

It will only change the appearance of Signature instead of PDF content.

23) &SetStraddleType

Set the type of Straddle Sign

Prototype:

BOOL SetStraddleType(short nType)

Parameters:

nType: The value must be 0, 1, 2, 3, 4;

0 means Middle Straddle Sign,

1 means Left Straddle Sign,

2 means Right Straddle Sign,

3 means Top Straddle Sign,

4 means Bottom Straddle Sign.

Return Value:

Returns TRUE if successful, otherwise FALSE.

24) &SetStraddlePos

Set the position of Straddle Sign.

Prototype:

BOOL SetStraddlePos(float fPos)

Parameters:

fPos: If it is Top Straddle Sign or Bottom Straddle Sign, the value will be the horizontal middle point of the signature field. Otherwise, it will be the vertical middle point of the signature field.

Return Value:

Returns TRUE if successful, otherwise FALSE.

25) &SetStraddleBitmap

Set the image for Straddle Sign appearance according to the state.

Prototype:

BOOL SetStraddleBitmap(short nState, BSTR sFilePath)

Parameters:

nState: The value must be 0, 1, 2, 3;

0 means Unknown,

1 means Unsigned,

2 means Valid,

3 means Invalid.

sFilePath: The path of the image file.

Return Value:

Returns TRUE if successful, otherwise FALSE.

26) &SetStraddlePages

Set the page range in which the signature is displaying.

Prototype:

BOOL SetStraddlePages(BSTR sRange)

Parameters:

sRange: The value is the page range in which the Straddle Sign is displaying and the format can be “0-10” or “0-2,3-10”;

Return Value:

Returns TRUE if successful, otherwise FALSE.

27) &SetStraddleFirstPagePercent

Set the percent of Straddle Sign appearance for the first signed PDF page.

Prototype:

BOOL SetStraddleFirstPagePercent(float fPercent)

Parameters:

fPercent: The range of this value is 0 to 1;

Return Value:

Returns TRUE if successful, otherwise FALSE.

Events

1) &OnSetSignatureInfo

Triggered when you click the unsigned signature field or right click the field to select the signature. It asks to set the properties for IPDFSignatureField object to complete the signing.

Prototype:

void OnSetSignatureInfo(IPDFSignatureField* pSignature);

Parameters:

pSignature: The signature field object which is clicked or right clicked.

2) &OnSigning

Triggered when the signature field is unsigned. It will be signed by the default or by a third-party.

Prototype:

void OnSigning(IPDFSignatureField* pSignature);

Parameters:

pSignature: The signature field object which is clicked.

3) &OnVerifying

Triggered when you click the signed signature field object or right click to select “Verify”. It will be verified by default or by third-party.

Prototype:

void OnVerifying(IPDFSignatureField* pSignature);

Parameters:

pSignature: The signature field object to be clicked.

4) &OnShowSignaturePropertyDialog

Triggered when you right click the signature field and then choose “Property” to check the signature properties.

Prototype:

Void OnShowSignaturePropertyDialog (IPDFSignatureField*pSignature, boolean* bShowProperty);

Parameters:

pSignature: The signature field object which is right clicked.

bShowProperty: Specify whether to pop up signature field property dialog.

Contact Us

Feel free to contact us should you need any information or have any problems with our products. We are always here, ready to serve you better.

- ***Office Address:***

  Foxit Corporation

  42840 Christy Street. Suite 201

  Fremont CA 94538

  USA

- ***Mailing Address:***

  Foxit Corporation

  42840 Christy Street. Suite 201

  Fremont CA 94538

  USA

- ***Sales:***

  1-866-680-3668 (24/7)

- ***Support:***

  1-866-MYFOXIT or 1-866-693-6948 (24/7)

- ***Fax:***

  530-535-9288

- ***Website:***

  www.foxitsoftware.com

- ***E-mail:***

  Sales and Information - sales@foxitsoftware.com

  Technical Support - support@foxitsoftware.com