User Manual

Foxit® PDF SDK(ActiveX)

Programming Guide

For Windows

Microsoft® Partner

Gold Independent Software Vendor (ISV)

Contents

Contents 2

Overview 18

Tutorials 19

Setup 19

1) Get trial verion 19

2) Download MSI pack to install. 19

3) Run demo. 19

Create project with ActiveX 21

1) Register 21

2) Create a Project 21

3) Select Foxit ActiveX SDK 22

4) Open a PDF document 23

5) How to call functions 23

6) How to create your own projects 24

Getting Started: Basic PDF Functions 24

1) Unlocking the SDK 24

2) Opening a PDF File 25

3) Go to a specific page 26

4) Zoom in and out of a page 27

5) Rotate a Page 29

6) Print a PDF document 29

7) Hide/Show UI elements 31

8) Iterate the tree outline of the PDF document 32

9) Search 33

IFoxitPDFSDK 35

Properties 35

1) FilePath 35

2) Password ... 35
3) PageCount ... 35
4) CurPage ... 35
5) Rotate ... 36
6) Zoomlevel ... 36
7) CurrentTool ... 36
8) Printer ... 38
9) DocumentInfo ... 38
10) ActiveXVersion ... 38
11) *bHasFormFields ... 38
12) *bHighlightFormFields ... 39
13) *FormFieldsHighlightAlpha ... 39
14) *FormFieldsHighlightColor ... 39
Method ... 39
1. Unlock ... 39
2. Global Settings ... 41
3. Open and Close PDF ... 44
4. Save ... 47
5. Navigation ... 48
6. View ... 51
7. Hyperlink ... 57
8. Text ... 59
9. Custom UI ... 62
10. Print ... 68
11. Search ... 69
12. Security ... 74
13. JavaScript ... 75
14. Multi-Instances ... 77
15. Edit and Page Organize ... 77
16. Asynchronous feature ... 82

17. Bookmark 83
18. Form 84
19. Annotation 86
20. Signature 89
21. Others 90

Event 94

1) BeforeDraw 94
2) AfterDraw 94
3) OnZoomChange 95
4) OnPageChange 95
5) OnOpenPassword 95
6) OnSearchProgress 95
7) OnOpenFile 95
8) OnOpenDocument 96
9) OnFilePathInvalidate 96
10) OnShowSavePrompt 96
11) OnCloseDocument 96
12) OnDocumentChange 96
13) CustomFileGetSize 97
14) CustomFileGetBlock 97
15) OnClick 97
16) OnDbClick 97
17) OnRButtonClick 98
18) OnDownLoadFinish 98
19) OnErrorOccurred 98
20) OnUploadFinish 98
21) OnTextHyperLink 99
22) OnExcuteMenuItem 99
23) *OnAddMenuItemAction 99
24) OnDoGoToRAction 100

25) OnDoGoToEAction 100

26) *OnHyperLink 100

27) *OnContextMenuIndex 101

28) *OnFetchAsyncFileData 101

29) *OnCurPageIndexChanged 101

30) *OnSigContextMenuIndex 102

31) OnPagesContextMenuIndex 102

32) OnBookmarkContextMenuIndex 102

33) *OnFormFieldClick 103

34) *OnFormFieldKeyDown 103

35) *OnFormFieldKeyUp 103

36) *OnSetFocus 103

37) *OnKillFocus 104

38) *OnDbClickEx 104

39) *OnRButtonClickEx 104

40) #FormFieldError 105

IPDFPrinter 106

Properties 106

1) PrinterName 106

2) PrinterRangeMode 106

3) PrinterRangeFrom 106

4) PrinterRangeTo 106

5) NumOfCopies 106

6) Scaling 107

7) AutoRotate 107

8) AutoCenter 107

9) Collate 107

10) Rotation 107

11) RangeSubset 108

12) ReversePage 108

13) PageBorder 108

14) PrintWhat 108

Methods 108

1) PrintWithDialog 108

2) PrintQuiet 109

3) SetPaperSize 109

4) GetSystemPrinterCount 109

5) GetSystemPrinterNameByIndex 109

6) SetPaperSizeByPage 110

7) SetDuplexMode 110

IPDFDocumentInfo 111

Properties 111

1) Author 111

2) Subject 111

3) CreatedDate 111

4) ModifiedDate 111

5) Keywords 111

6) Creator 111

7) Producer 112

8) Title 112

IFindResult 113

Method 113

1) GetFindRectsCount 113

2) GetFindRectByIndex 113

3) GetFindPageNum 113

4) GetFindFileName 113

5) GetFindString 114

IPDFOutline 115

Method 115

1) GetOutlineDest 115

2) GetOutlineAction 115

3) GetOutlineColor 115

4) NavigateOutline 115

5) GetOutlineTitle 115

6) GetOutLineTitle2 116

7) GetOutlineExpandValue 116

ILink_Dest 117

Method 117

1) GetPageIndex 117

2) GetZoomMode 117

3) GetZoomParamCount 117

4) GetZoomParam 117

5) GetDestName 118

IPDFAction 119

Method 119

1) GetURIPath 119

2) GetFilePath 119

3) GetType 119

4) GetDest 119

#IPDFForm 121

Method 121

1) #ImportFromFDF 121

2) #ExportToFDF 121

3) #AddField 121

4) #GetSelectedField 122

5) #RemoveField 122

6) #RemoveFieldsByName 123

7) #GetFieldsCount 123

8) #GetFieldByIndex 123

9) #GetSelectedField 123

#IPDFFormField 125

Properties 125

1) #Alignment 125
2) #BorderStyle 125
3) #BorderWidth 125
4) #ButtonLayout 125
5) #CalcOrderIndex 126
6) #CharLimit 126
7) #DefaultValue 126
8) #IsEditable 126
9) #Behavior 127
10) #IsHidden 127
11) #IsMultiline 127
12) #IsPassword 127
13) #IsReadOnly 128
14) #IsRequired 128
15) #Name 128
16) #NoViewFlag 128
17) #PrintFlag 129
18) #Style 129
19) #TextFont 129
20) #TextSize 130
21) #Type 130
22) #Value 130
23) #Tooltip 131
24) #Orientation 131
25) #DirtyFlag 131
26) #ID 131

Method 131

1) #PopulateListOrComboBox 131

2) #SetBackgroundColor 132

3) #SetBorderColor 132

4) #SetForegroundColor 133

5) #SetButtonCaption 133

6) #SetButtonIcon 134

7) #SetExportValues 134

8) #SetJavaScriptAction 134

9) #SetResetFormAction 135

10) #SetSubmitFormAction 135

11) #GetPageIndex 135

12) #GetRectTop 136

13) #GetRectLeft 136

14) #GetRectRight 136

15) #GetRectBottom 136

^IPDFPageAnnots 138

Method 138

1) ^GetAnnot 138

2) ^GetLTAnnot 138

3) ^ReleaseLTAnnot 138

4) ^AddAnnot 138

5) ^AddLTAnnot 139

6) ^ SetStampParam 140

7) ^RemoveAnnot 140

8) ^GetAnnotIndex 140

9) ^GetAnnotsCount 141

^IPDFAnnot 142

Properties 142

1) ^Thickness 142

2) ^BorderStyle 142

3) ^Color 143

4) ^LineStartingStyle 143
5) ^LineEndingStyle 144
6) ^FillColor 144
7) ^Opacity 144
8) ^Author 145
9) ^Subject 145
10) ^CreationDate 145
11) ^ModificationDate 145
12) ^Locked 145
13) ^Print 146
14) ^ReadOnly 146
15) ^Description 146

Methods 147

1) ^GetType 147
2) ^GetSubType 147
3) ^GetContents 147
4) ^SetContents 147
5) ^IsPopupOpen 148
6) ^SetPopupOpen 148
7) ^HasPopup 148
8) ^GetRect 148
9) ^SetRect 149
10) ^SetLinkGoToAction 149
11) ^SetLinkURLAction 149
12) ^DoAction 150
13) ^HasAction 150
14) ^GetMarkedState 150
15) ^SetMarkedState 150
16) ^GetReviewState 151
17) ^SetReviewState 151

18) ^GetMigrationState 152
19) ^SetMigrationState 152
20) ^GetStartingPoint 152
21) ^SetStartingPoint 153
22) ^GetEndingPoint 153
23) ^SetEndingPoint 153
24) ^SetMediaPoster 154
25) ^SetMultimedia 154
26) ^SetLinkQuadPoints 154
27) ^SetPolygonVertices 155
28) ^SetPencilVertices 155
29) ^AttachFile 155
30) ^ GetReplyList 155
31) ^ UpdateAnnotReplies 156
Events 156
1) ^OnAnnotCreated 156
2) ^OnAnnotDeleted 156
3) ^OnAnnotModified 156
4) ^OnAnnotReplyCreated 157
5) ^OnAnnotReplyDeleted 157
6) ^OnAnnotReplyModified 157
7) ^OnAnnotRButtonDown 157
8) ^OnAnnotRButtonUp 158
9) ^OnAnnotLButtonDbClick 158
10) ^OnAnnotLButtonDown 158
11) ^OnAnnotLButtonUp 159
12) ^OnAnnotPosChanged 159
13) ^OnAnnotMoving 159
14) ^OnAnnotMouseEnter 160
15) ^OnAnnotMouseExit 160

16) ^GetRectTop 160

17) ^GetRectLeft 160

18) ^GetRectRight 161

19) ^GetRectBottom 161

^ IPDFAnnotReplyList 162

Methods 162

1) ^ GetCount 162

2) ^ GetItem 162

3) ^ Remove 162

4) ^ RemoveAll 162

5) ^ Add 163

^IPDFAnnotReply 164

Methods 164

1) ^ GetCreator 164

2) ^ SetCreator 164

3) ^ GetContent 164

4) ^ SetContent 164

5) ^ GetChildren 165

6) ^ GetParent 165

7) ^ GetCreationDate 165

8) ^ SetCreationDate 165

9) ^ SetReadonly 166

10) ^ GetReplyID 166

^IPDFormatTool 167

Method 167

1) ^SetFontName 167

2) ^GetFontName 167

3) ^SetFontSize 167

4) ^GetFontSize 167

5) ^SetFontColor 168

6) ^GetFontColor 168
7) ^SetBorderColor 168
8) ^GetBorderColor 168
9) ^SetFillColor 169
10) ^GetFillColor 169
11) ^SetFontBold 169
12) ^GetFontBold 170
13) ^GetFontBoldEnable 170
14) ^SetFontItalic 170
15) ^GetFontItalic 170
16) ^GetFontItalicEnable 171
17) ^SetAlign 171
18) ^GetAlign 171
19) ^SetCharSpace 172
20) ^GetCharSpace 172
21) ^SetCharHorzScale 172
22) ^GetCharHorzScale 172

&IPDFSignatureMgr 173

Method 173

1) &Add 173
2) &SignDocument 173
3) &Verify 174
4) &VerifyAll 174
5) &GetCounts 174
6) &Get 175
7) &Clear 175
8) &Remove 175
9) &InitStraddleValue 175
10) &CreatePatternSigField 176
11) &SetCurPattenSigField 176

12) &CountPatternSigFiel 176
&IPDFSignatureField........ 178
Properties........ 178
1) &Reason 178
2) &Location 178
3) &Signer........ 178
4) &Filter........ 178
5) &SubFilter........ 179
6) &State........ 179
Methods 179
1) &SetAPOptions........ 179
2) &SetAPText........ 180
3) &SetAPImage........ 180
4) &SetAPImageData 181
5) &IsSigned........ 181
6) &SetSignerDN........ 181
7) &SetStatusImage 181
8) &SetStatusImageData........ 182
9) &GetPageIndex........ 183
10) &GetSourceBuffer 183
11) &GetSourceBufferLen........ 183
12) &GetSignedBuffer........ 184
13) &GetSignedBufferLen........ 184
14) &CreateSignedDoc........ 184
15) &SetVerifyResult........ 185
16) &SetCertPath 185
17) &SetCertData........ 185
18) &SetCertContext........ 186
19) &TurnGray 186
20) &TurnBlur 186

21) &SetVisible 187

22) &GrayPrint 187

23) &SetStraddleType 187

24) &SetStraddlePos 188

25) &SetStraddleBitmap 188

26) &SetStraddlePages 188

27) &SetStraddleFirstPagePercent 189

28) &SetSigFieldAlpha 189

29) &Refresh 189

30) &SetDefaultContentsLength 189

Events 190

1) &OnSetSignatureInfo 190

2) &OnSigning 190

3) &OnVerifying 190

4) &OnShowSignaturePropertyDialog 190

*IPDFTextDoc 192

Methods 192

1) *ReleasePDFTextDoc 192

2) *LoadPDFTextPage 192

*IPDFTextPage 193

Methods 193

1) *ReleasePDFTextPage 193

2) *CountChars 193

3) *GetChars 193

4) *GetCharInfo 193

5) *GetCharIndexAtPos 194

6) *GetNextCharIndexByDirection 194

7) *SelectByRange 194

8) *SelectByRectangle 195

9) *StartSearch 195

10) *ExtractLinks 196

11) *ExtractPageText 196

*IPDFCharInfo 197

Properties 197

1) *state 197

2) *fontSize 197

3) *originX 197

4) *originY 197

5) *fontName 197

6) *fontAscent 198

7) *fontDescent 198

8) *displayFontSize 198

Methods 198

1) *GetBBox 198

2) *GetMatrix 198

*IPDFTextSearch 200

Methods 200

1) *ReleaseTextSearch 200

2) *FindNext 200

3) *FindPrev 200

4) *GetSelection 200

*IPDFTextSelection 202

Methods 202

1) *ReleaseTextSelection 202

2) *GetBBox 202

3) *GetBBoxEx 202

4) *GetChars 203

5) *CountPieces 203

6) *GetPieceRect 203

7) *GetPieceCharStart 203

8) *GetPieceCharCount 204

*IPDFTextLink 205

Methods 205

1) *CountLinks 205

2) *GetLink........ 205

3) *GetSelection 205

Contact Us 206

Overview

Foxit PDF SDK ActiveX is a visual programming component that offers PDF displaying capability with lowest 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.

The ActiveX version is much easier to use and has much more 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.

From version 5.0, ActiveX incorporates Signature module and many advanced PDF features, such as asynchronous feature. Signature module 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 print control and more.

From version 5.2, ActiveX provides native 64-bit support.

Foxit offers two versions of ActiveX – standard version and professional version, 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: page organization, setting password and permission, printing pop-up annotations, importing/exporting annotation data, importing/exporting form data, running JavaScript, converting PDF to text, etc.

Based on your requirements you are able to pick and choose which ActiveX version and which ActiveX module would be best for your application. Please Scaling

that the three modules can only be added to Professional version as the add-ons. Please contact sales@foxitsoftware.com for license details.

In this developer's guide,
All the functions marked with an asterisk (*) are available only in the professional version
All the functions marked with a hash sign (#) are available only in the Form Module
All the functions marked with a caret (^) are available only in the Annotation Module
All the functions marked with an ampersand (&) are only available in Signature Module.

Foxit PDF SDK ActiveX runs on Windows XP 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.

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 functions once inside your program before you call ANY other functions of the ActiveX. 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

This tutorials part is targeted towards C/C++ developers using the Foxit PDF ActiveX SDK. It assumes the developer is familiar with C/C++ and Microsoft Foundation Classes (MFC).

This tutorials part covers how to use the Foxit PDF ActiveX SDK. It uses the demo provided by Foxit Corporation as reference for explanation.

Setup

1) Get trial verion.

Get the trial version of ActiveX from Foxit website:

http://www.foxitsdk.com/products/pdf-sdk-activex/

2) Download MSI pack to install.

You will receive an email with two download links: *FoxitPDFSDKActiveX_Std.msi* and *FoxitPDFSDKActiveX_Pro.msi*. Then download the MSI pack as you need and double click on it to start the installation.

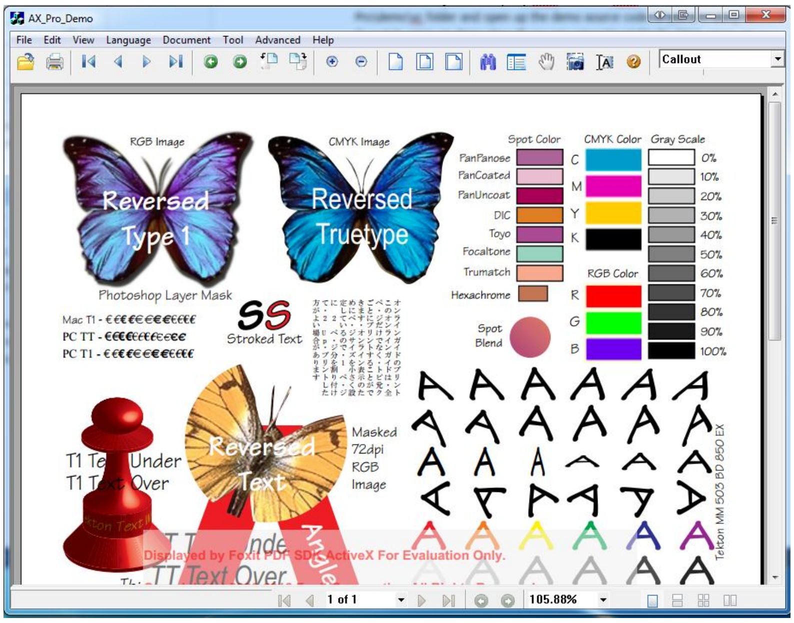
Please note that it will automatically register the OCX files, including the 32bit OCX and the 64bit OCX, to Windows when installing with the MSI pack.

3) Run demo.

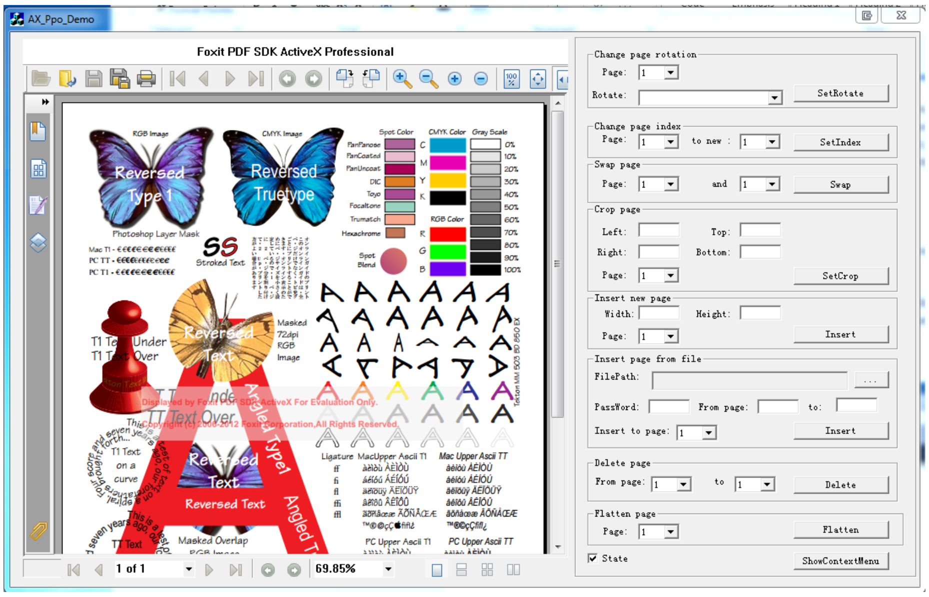
Here take ActiveX professional version.

1. The files are installed under C:\Program Files (x86)\Foxit Software\Foxit PDF SDK ActiveX Pro. Go to the C:\Program Files (x86)\Foxit Software\Foxit PDF SDK ActiveX Pro\demo\vc folder and open up the demo source code.
   - If you are using Visual C++ 6.0, open up demo.dsw.
   - If you are using Visual Studio 2010, open up demo.sln. The instructions and screenshots in this tutorial will reference Visual Studio 2010.
2. The demo.sln file contains 2 demos.

   AX_Pro_Demo – A full featured PDF reader with a drop down list of annotation tools.


AX_Pro_Demo
File Edit View Language Document Tool Advanced Help
Callout
1 of 1
105.88%
Displayed by Foxit PDF SDK ActiveX For Evaluation Only.

AX_Ppo_Demo – A full featured PDF reader with a panel for page manipulation functions.



AX_Annot_Demo – A full featured PDF reader with a panel for annotation manipulation functions.

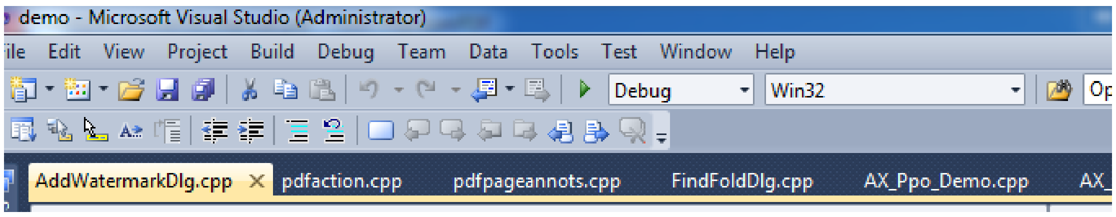
AX_Form_Demo – A full featured PDF reader with a panel for form processing functions.

AX_Signature_Demo – A full featured PDF reader with a panel for signature manipulation functions.

3. The remainder of this demo will examine the basic functions developers will start with when using the SDK. Make sure to set AX_Pro_Demo as the startup project.

   In Visual Studio 2010, go to the Solution Explorer > right click on AX_Pro_Demo > Set as Startup Project.

4. Select to build a Debug version of the demo so you will be able to step through the code.


demo - Microsoft Visual Studio (Administrator)
ile Edit View Project Build Debug Team Data Tools Test Window Help
Debug
Win32
Op
AddWatermarkDlg.cpp
pdfaction.cpp
pdfpageannots.cpp
FindFoldDlg.cpp
AX_Ppo_Demo.cpp
AX_

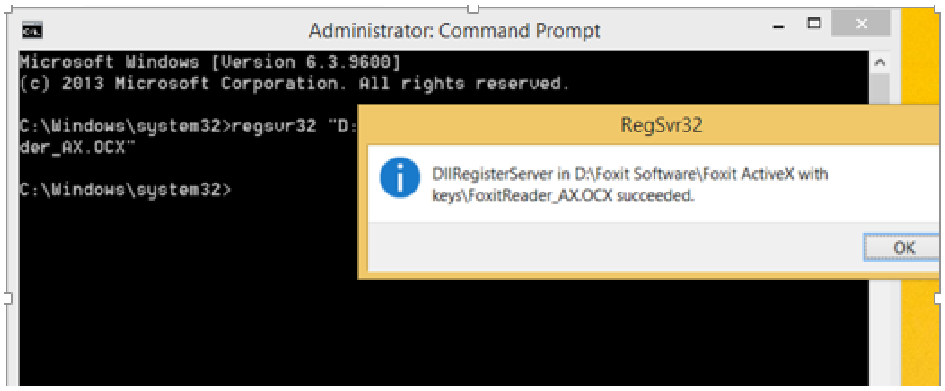
5. Build the demo: Go to Build > Build Solution.
6. Make sure there are no errors in the build output.
7. Run the demo by going to Debug > Start Debugging.

Create project with ActiveX

1) Register

To register an OCX file to Windows:

Run your command line > type: regsvr32 "Filepath\FoxitReader_AX.OCX" > hit enter to register > get the prompt once it is successfully registered.



Administrator: Command Prompt

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>regsvr32 "D:
der_AX.OCX"

C:\Windows\system32>
```

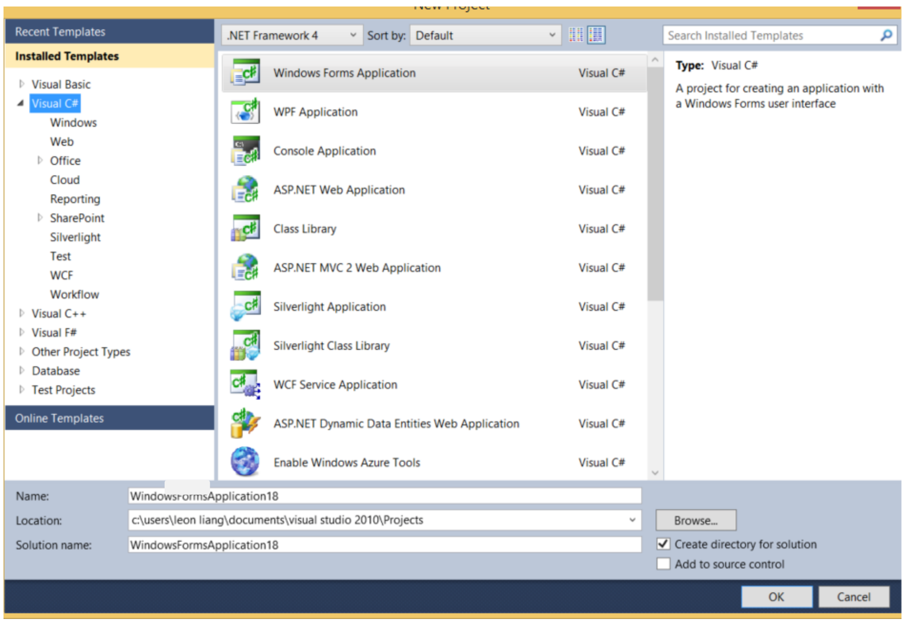
RegSvr32

DllRegisterServer in D:\Foxit Software\Foxit ActiveX with keys\FoxitReader_AX.OCX succeeded.

OK

2) Create a Project

Go to visual studio > File > New > Project > Windows Forms Application



Recent Templates
Installed Templates
- Visual Basic
- Visual C#
  - Windows
  - Web
  - Office
  - Cloud
  - Reporting
  - SharePoint
  - Silverlight
  - Test
  - WCF
  - Workflow
- Visual C++
- Visual F#
- Other Project Types
- Database
- Test Projects

Online Templates

.NET Framework 4 Sort by: Default

Search Installed Templates

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Application	Visual C#
Class Library	Visual C#
ASP.NET MVC 2 Web Application	Visual C#
Silverlight Application	Visual C#
Silverlight Class Library	Visual C#
WCF Service Application	Visual C#
ASP.NET Dynamic Data Entities Web Application	Visual C#
Enable Windows Azure Tools	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

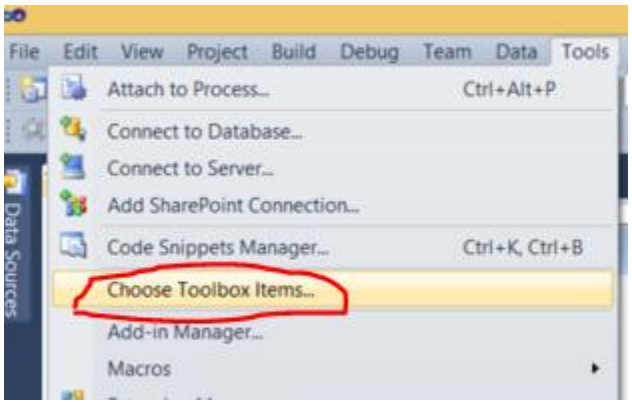
Name: WindowsFormsApplication18
Location: c:\users\leon liang\documents\visual studio 2010\Projects
Solution name: WindowsFormsApplication18

Browse...
☑ Create directory for solution
☐ Add to source control

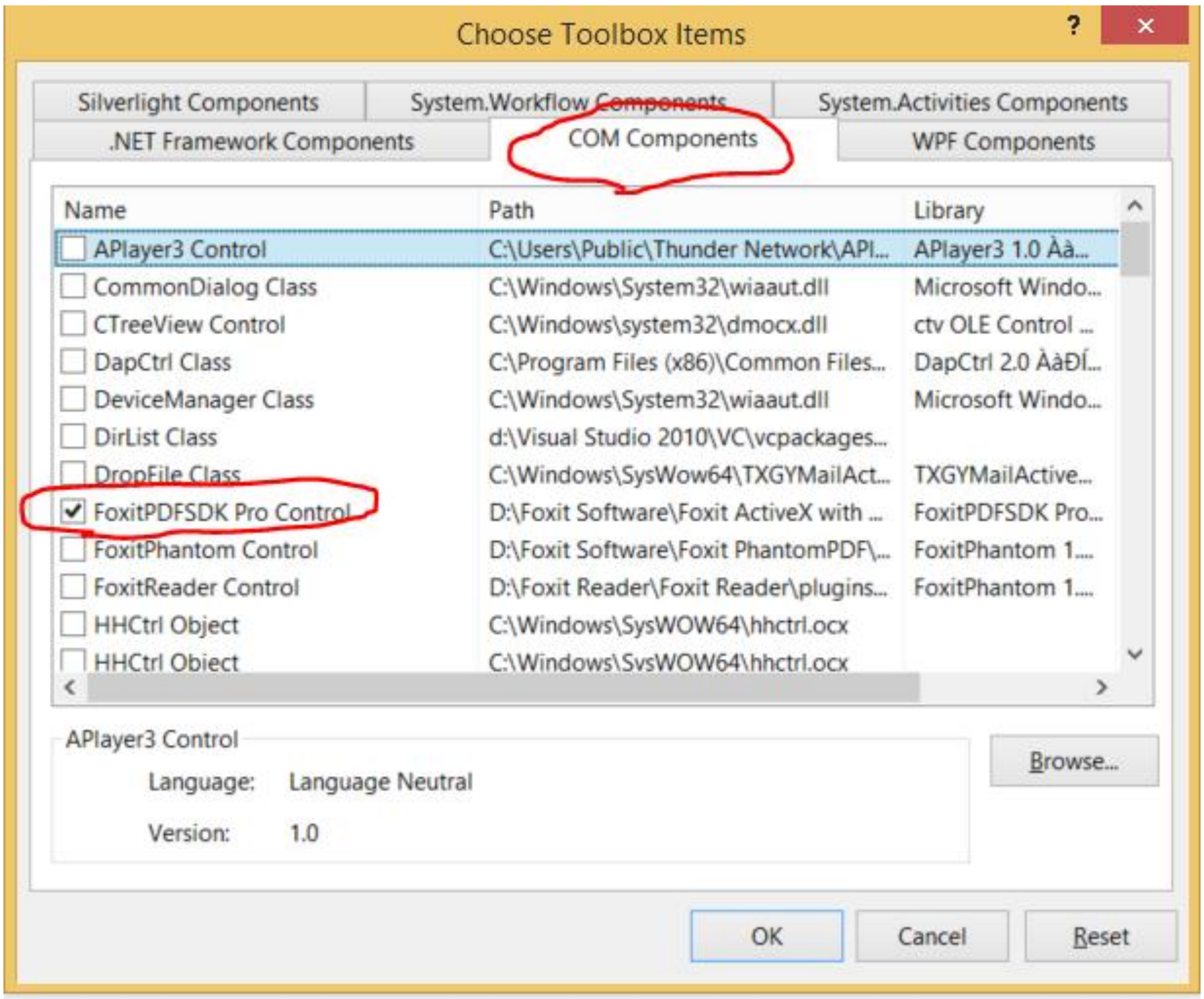
OK Cancel

3) Select Foxit ActiveX SDK

Click Tool > Choose Toolbox Items



Under "COM Components" tab, check "FoxitPDFSDK Pro Control", then hit OK



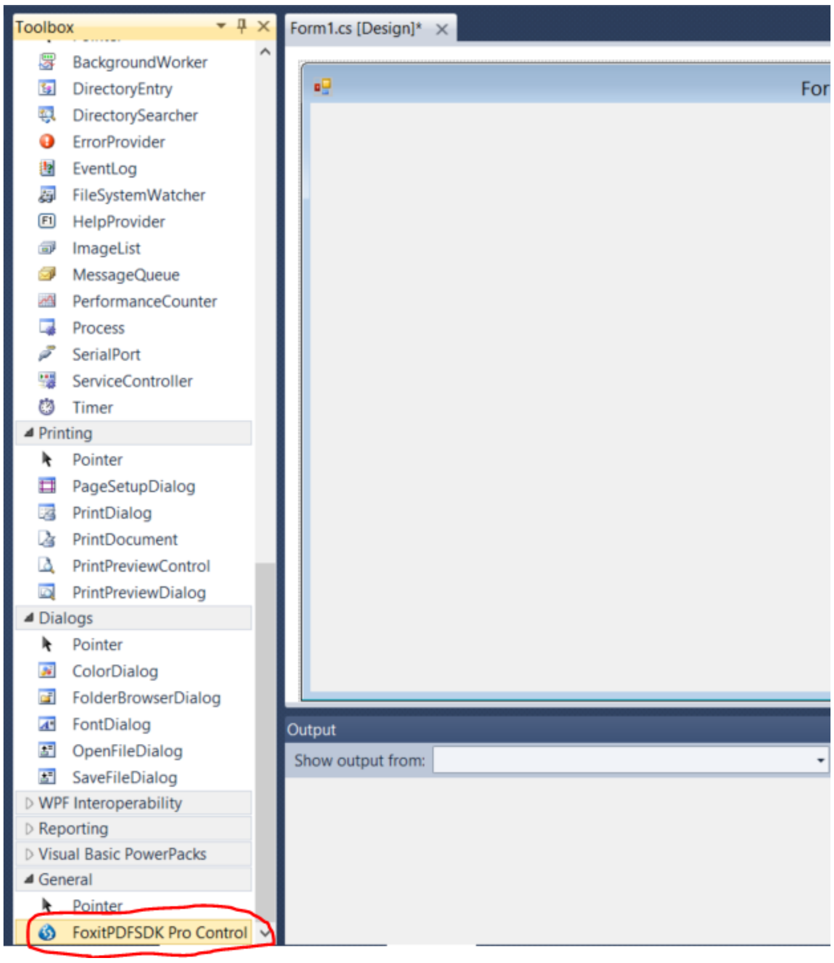
Name	Path	Library
☐ APlayer3 Control	C:\Users\Public\Thunder Network\APl...	APlayer3 1.0 Àà...
☐ CommonDialog Class	C:\Windows\System32\wiaaut.dll	Microsoft Windo...
☐ CTreeView Control	C:\Windows\system32\dmocx.dll	ctv OLE Control ...
☐ DapCtrl Class	C:\Program Files (x86)\Common Files...	DapCtrl 2.0 ÀàÐÍ...
☐ DeviceManager Class	C:\Windows\System32\wiaaut.dll	Microsoft Windo...
☐ DirList Class	d:\Visual Studio 2010\VC\vcpackages...	
☐ DropFile Class	C:\Windows\SysWow64\TXGYMailAct...	TXGYMailActive...
✓ FoxitPDFSDK Pro Control	D:\Foxit Software\Foxit ActiveX with ...	FoxitPDFSDK Pro...
☐ FoxitPhantom Control	D:\Foxit Software\Foxit PhantomPDF\...	FoxitPhantom 1....
☐ FoxitReader Control	D:\Foxit Reader\Foxit Reader\plugins...	FoxitPhantom 1....
☐ HHCtrl Object	C:\Windows\SysWOW64\hhctrl.ocx	
☐ HHCtrl Object	C:\Windows\SysWOW64\hhctrl.ocx	

APlayer3 Control

Language: Language Neutral

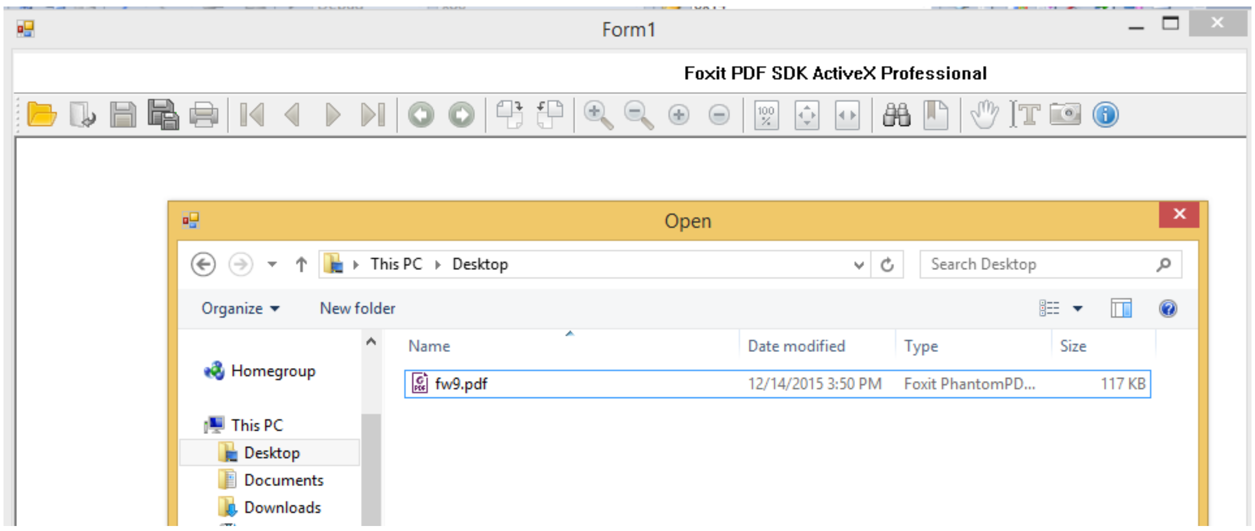
Version: 1.0

Drag FoxitPDFSDK Pro Control into Form window



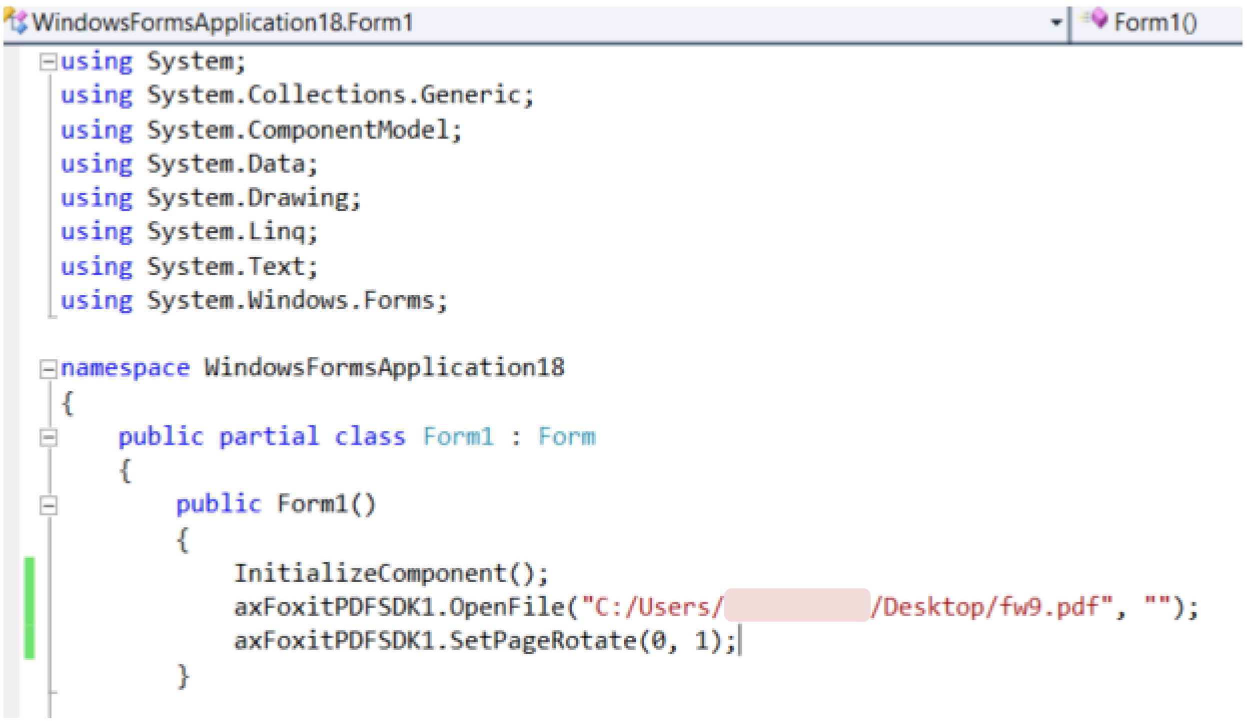
4) Open a PDF document

Now you can run this basic program and open PDF documents by click the yellow folder



5) How to call functions

To call other functions, they should be used by following axFoxitPDFSDK1.

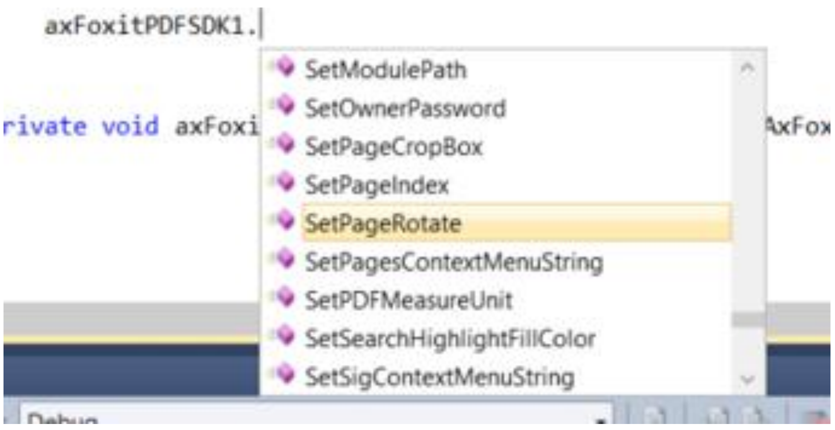


```
WindowsFormsApplication18.Form1                                   Form1()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication18
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            axFoxitPDFSDK1.OpenFile("C:/Users/[redacted]/Desktop/fw9.pdf", "");
            axFoxitPDFSDK1.SetPageRotate(0, 1);
        }
```

In this example, these two lines are to open fw9.pdf on your desktop, and then rotate the first page to right by 90 degree.

6) How to create your own projects

You can see more options in this list and create your own project from there.



```
axFoxitPDFSDK1.

rivate void axFoxi                                    AxFox
```

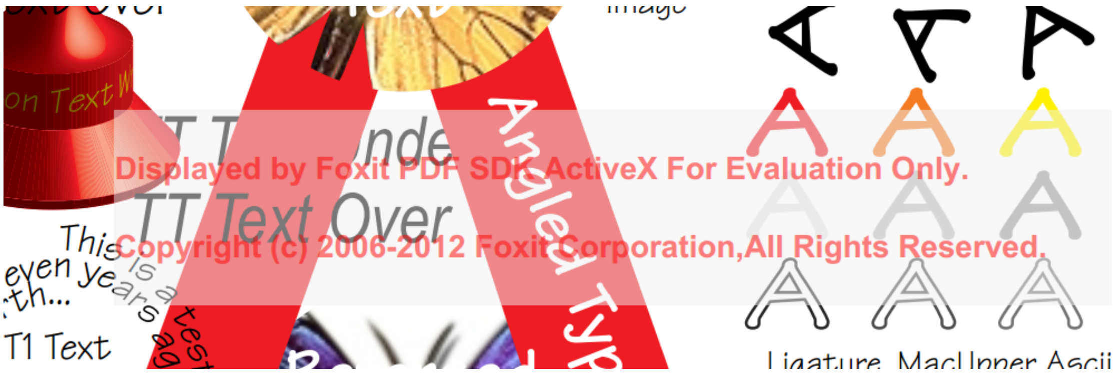
Getting Started: Basic PDF Functions

The following part contains references to the AX_Pro_Demo. To get a full understanding of how the functions works, the user should set breakpoints and step through the code. File names and line numbers for specific function calls are provided.

1) Unlocking the SDK

The evaluation version of the ActiveX SDK will apply a watermark across any rendered PDF page.

Customers who pay for the SDK will receive a license key to remove the watermark.


Displayed by Foxit PDF SDK ActiveX For Evaluation Only.
Copyright (c) 2006-2012 Foxit Corporation,All Rights Reserved.
TT Text Over
T1 Text
Angled Typ
Ligature MacUpper Ascii

The license key will contain a license id and an unlock code that you will pass in as parameters to the UnLockActiveX function. There is an example of this in AX_Pro_DemoDlg.cpp.

```
BOOL CAX_Pro_DemoDlg::OnInitDialog()

{

  CDialog::OnInitDialog();


  // code


  m_AX.UnLockActiveX(_T("Licence_ID"),_T("Unlock_code"));


  // code

}
```

2) Opening a PDF File

Call the OpenFile function to open a PDF file for viewing. There is a working example in AX_Pro_DemoDlg.cpp.

```
void CAX_Pro_DemoDlg::OnFileOpen()

{

  //code


  m_AX.OpenFile(fdg.GetPathName() ,NULL );
```

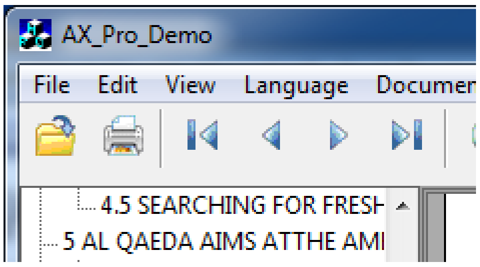
```
    // code
  }
```

The user can trigger the OnFileOpen event by going to File > Open > browse to a PDF file > click on Open or by using the file open folder icon.

3) Go to a specific page

Use the GoToPage function to navigate directly to the page number specified as the parameter. Page indexing is zero based. So call GoToPage(0) to get to the first page.

The AX_Pro_Demo has arrows for navigating to the first, previous, next, and last page of a multipage document. To see these functions in action, run the demo and open a multiple page PDF.



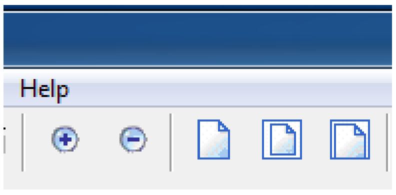
In AX_Pro_DemoDlg.cpp, you will see the corresponding events that are triggered when the arrow buttons for navigating pages are pressed. You will see the GoToPage function is called with the appropriate index value.

```
void CAX_Pro_DemoDlg::OnFirstPage()
{
  m_AX.GoToPage(0);
}



void CAX_Pro_DemoDlg::OnLastPage()
{
  int count = m_AX.GetPageCount();
  m_AX.GoToPage(count-1);
}
```

```
void CAX_Pro_DemoDlg::OnNextPage()
{   int count = m_AX.GetPageCount();
    int current = m_AX.GetCurPage();

    if(current+1 <= count-1)
       m_AX.GoToPage(current+1);
}

void CAX_Pro_DemoDlg::OnPrevPage()
{
  int count = m_AX.GetPageCount();
    int current = m_AX.GetCurPage();

    if(current-1 >= 0)
       m_AX.GoToPage(current-1);
}
```

4) Zoom in and out of a page

Call SetZoomLevel to zoom in and out of the PDF page. The AX_Pro_Demo has menu buttons for zoom in, zoom out, actual size, fit page, and fit width. All of these buttons trigger events that call SetZoomLevel.

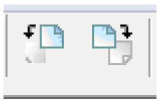


```
void CAX_Pro_DemoDlg::OnZoomIn()
{
  zoomfactor = m_AX.GetZoomLevel();

  if(zoomfactor + 10 == 6400)
      zoomfactor = 6400;
  else
      zoomfactor = zoomfactor + 10;

  m_AX.SetZoomLevel( zoomfactor );
}

void CAX_Pro_DemoDlg::OnZoomOut()
{
   zoomfactor = m_AX.GetZoomLevel();

  if(zoomfactor - 10 > 0)
      zoomfactor = zoomfactor - 10;

  m_AX.SetZoomLevel( zoomfactor );
}

void CAX_Pro_DemoDlg::OnActualSize()
{
  m_AX.SetZoomLevel(0);
}

void CAX_Pro_DemoDlg::OnFitPage()
{
  m_AX.SetZoomLevel(1);
}
```

```
void CAX_Pro_DemoDlg::OnFitWidth()

{

  m_AX.SetZoomLevel(2);

}
```

5) Rotate a Page

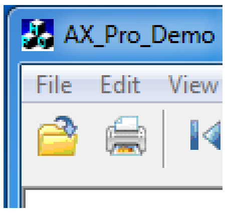
Call SetRotate to rotate the PDF page. The AX_Pro_Demo has menu buttons for rotate right and rotate left. These buttons trigger events that call SetRotate.



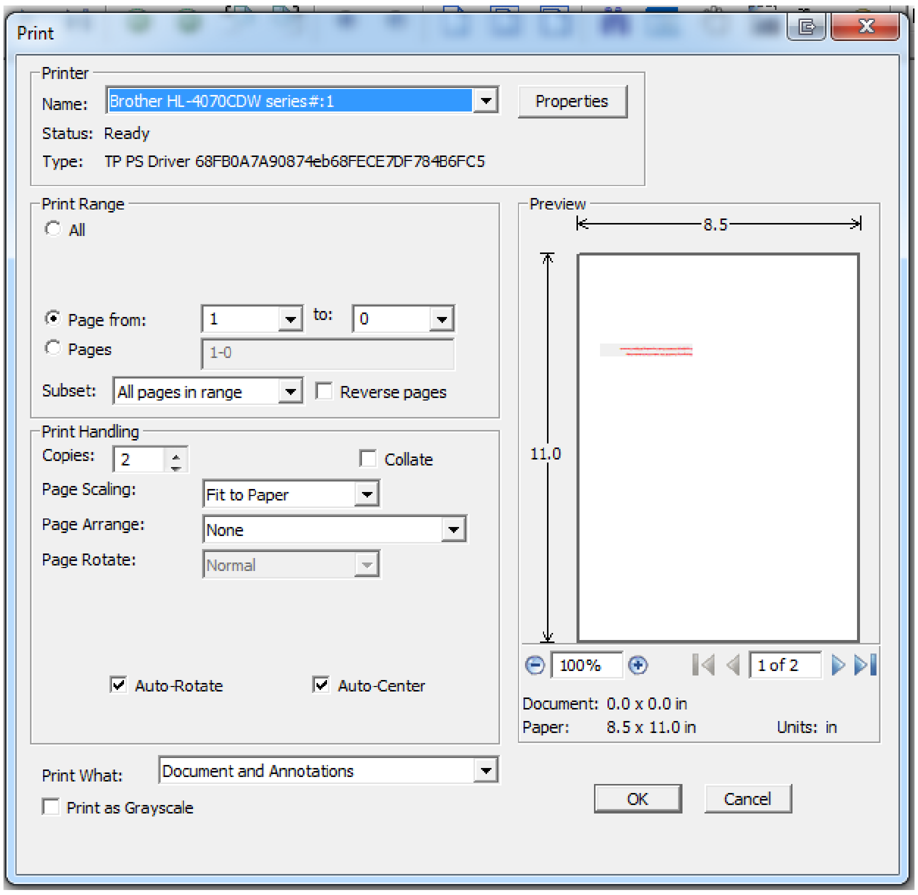
```
void CAX_Pro_DemoDlg::OnRotateRight()

{

    if(rotatefactor == 3)

       rotatefactor = 0;

  else

       rotatefactor++;

  m_AX.SetRotate(rotatefactor);

}


void CAX_Pro_DemoDlg::OnRotateLeft()

{

  if(rotatefactor == 0) rotatefactor=3;

    else rotatefactor--;

  m_AX.SetRotate( rotatefactor );

}
```

6) Print a PDF document

Call PrintWithDialog to bring up the print dialog page. On that page you can manually adjust printer settings. Using the ActiveX SDK you can also programmatically set printer settings. Clicking on the printer icon.



or going to File > Print will trigger the OnFilePrint() event that calls the PrintWithDialog function to bring up the print dialog settings.


Print

Printer
Name: Brother HL-4070CDW series#:1
Properties
Status: Ready
Type: TP PS Driver 68FB0A7A90874eb68FECE7DF784B6FC5

Print Range
All
[✓ Page from:] 1 to: 0
Pages 1-0
Subset: All pages in range
Reverse pages

Print Handling
Copies: 2
Collate
Page Scaling: Fit to Paper
Page Arrange: None
Page Rotate: Normal
[✓ Auto-Rotate]
[✓ Auto-Center]

Preview
8.5
11.0
100%
1 of 2
Document: 0.0 x 0.0 in
Paper: 8.5 x 11.0 in
Units: in

Print What: Document and Annotations
Print as Grayscale

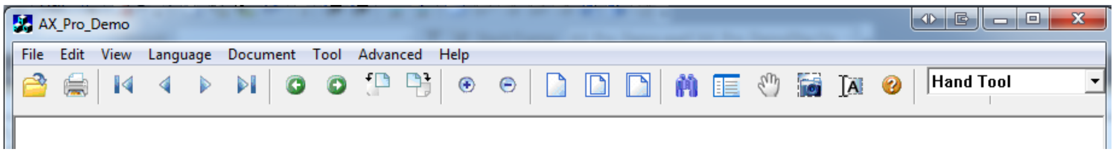
OK
Cancel

```
void CAX_Pro_DemoDlg::OnFilePrint()
{
  CPDFPrinter printer = m_AX.GetPrinter();
  if (!printer)
  {
      return;
  }
```

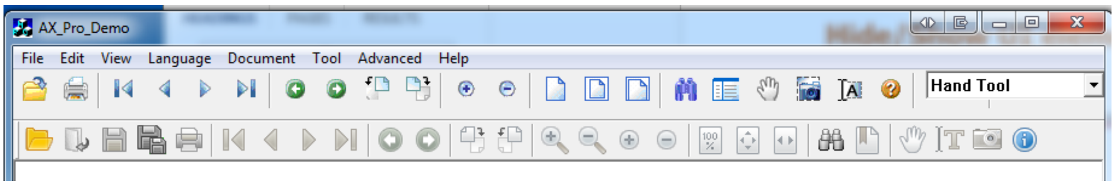
```
        //printer.SetPrinterName( "foxit PDF" );

        printer.SetPrinterRangeMode( 3 );

        printer.SetPrinterRangeFrom( 2 );

        printer.SetPrinterRangeTo( 3 );

        printer.SetPaperSize( 8 );

        printer.SetNumOfCopies( 2 );

        printer.PrintWithDialog();

    }
```

7) Hide/Show UI elements

The ActiveX SDK provides functions for modifying the user interface. For example, the ShowToolBar() function takes a boolean parameter. This is what the user interface looks like when ShowToolBar(FALSE) is called by the application code.



ShowToolBar(TRUE) adds the toolbar to the user interface.

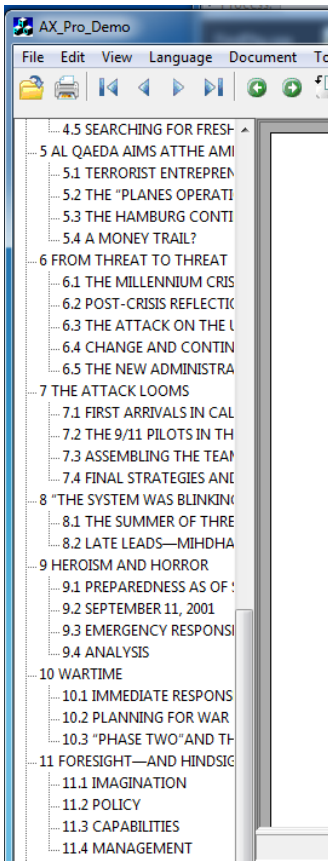


In AX_Pro_DemoDlg.cpp, the ShowToolBar function is called with FALSE as the default setting. As an exercise, the user can change m_ToolState to be TRUE to enable the toolbar when the demo is run.

```
    BOOL CAX_Pro_DemoDlg::OnInitDialog()

    {

    // code


        m_ToolState = FALSE;


        // code
        m_AX.ShowToolBar( m_ToolState );


      // code

    }
```

8) Iterate the tree outline of the PDF document

If the user wants to build a table of contents for a PDF document, they can use the GetOutlineFirstChild() and GetOutlineNextSibling() functions to iterate through the outline tree of the PDF. Here's an example of a TOC using the AX_Pro_Demo.



```
  4.5 SEARCHING FOR FRESH
5 AL QAEDA AIMS ATTHE AMI
  5.1 TERRORIST ENTREPREN
  5.2 THE "PLANES OPERATI
  5.3 THE HAMBURG CONTI
  5.4 A MONEY TRAIL?
6 FROM THREAT TO THREAT
  6.1 THE MILLENNIUM CRIS
  6.2 POST-CRISIS REFLECTIC
  6.3 THE ATTACK ON THE U
  6.4 CHANGE AND CONTIN
  6.5 THE NEW ADMINISTRA
7 THE ATTACK LOOMS
  7.1 FIRST ARRIVALS IN CAL
  7.2 THE 9/11 PILOTS IN TH
  7.3 ASSEMBLING THE TEAM
  7.4 FINAL STRATEGIES AND
8 "THE SYSTEM WAS BLINKING
  8.1 THE SUMMER OF THRE
  8.2 LATE LEADS—MIHDHA
9 HEROISM AND HORROR
  9.1 PREPAREDNESS AS OF
  9.2 SEPTEMBER 11, 2001
  9.3 EMERGENCY RESPONSI
  9.4 ANALYSIS
10 WARTIME
  10.1 IMMEDIATE RESPONS
  10.2 PLANNING FOR WAR
  10.3 "PHASE TWO"AND TH
11 FORESIGHT—AND HINDSIG
  11.1 IMAGINATION
  11.2 POLICY
  11.3 CAPABILITIES
  11.4 MANAGEMENT
```

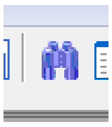
In AX_Pro_DemoDlg.cpp in the CAX_Pro_DemoDlg::OnFileOpen() function there is code that shows how the TOC is generated when the file is first open in the demo. To get a closer understanding, it is recommended that the developer set a breakpoint at the beginning of this function and step through. See where GetOutlineFirstChild and GetOutlineNextSibling are called.

```
  void CAX_Pro_DemoDlg::OnFileOpen()
```

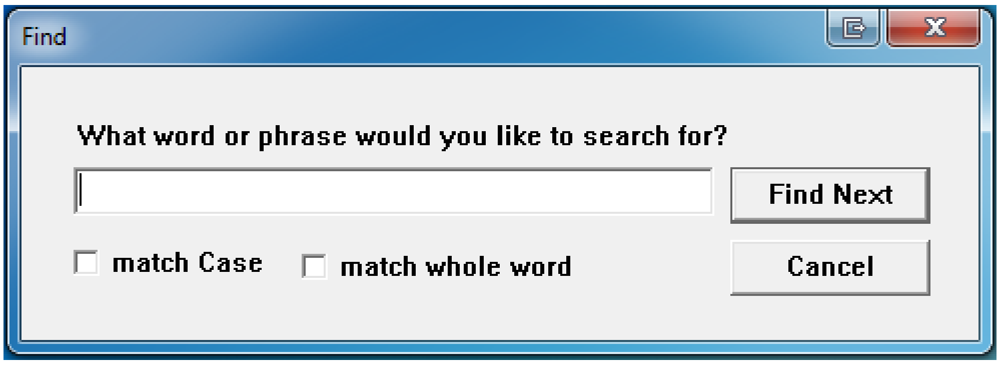
```
{
  // code


  CPDFOutline outline;

  outline = m_AX.GetOutlineFirstChild( outline.m_lpDispatch );


  if (outline != NULL)

  {

      outline = NULL;

      ProcOutline(outline,NULL);

  }


  OnMENUITEMexpandtree();

    if (m_outlinetree.GetCount()>0)

  {

      ShowBookMark();

      m_bhidebookmark = TRUE;

  }

  else

  {

      HideBookMark();

      m_bhidebookmark = FALSE;

  }

}
```

9) Search

The AX_Pro_Demo has a search icon in the toolbar that allows a user to search through a PDF for specific words or phrases.



This Find dialog box appears when the icon is clicked on,



The FindFirstEx() function finds the first occurrence of the string in the PDF document. The parameters are the search string, a boolean to indicate if the user wants to match case, a Boolean to indicate if the user wants to match the entire word. In FindDlg.cpp in the CFindDlg::OnOK() function there is an working example of FindFirstEx() and FindNext().

```
void CFindDlg::OnOK()
{
  UpdateData(true);
  If (TRUE == m_bFirst)
    {
    // code to initialize the var parameter

    m_pAX->FindFirstEx(var, m_check1, m_check2);

      // code
    }
    else
      //FindNext(BOOL) TRUE:down search, FALSE:up search
      m_pAX->FindNext(TRUE);
}
```

IFoxitPDFSDK

This section describes all the properties, methods and events exposed by ActiveX. They can be used for the basic PDF operations, such as viewing, printing, navigation, text search and so on.

Please note that this reference shows everythings in C syntax. If you use a programming language other than C/C++, please follow the corresponding syntax.

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.
The functions marked with a caret (^) are available only in the Annotation Module.

Properties

1) FilePath

Type:

BSTR, read-only

Description:

File path of current PDF.

Note:

If no PDF file is opened or the file is opened from the buffer or stream, the value will be NULL.

2) Password

Type:

BSTR, read-only

Description:

Password for opening a PDF.

3) PageCount

Type:

long, read-only

Description:

Total page number of current PDF.

4) CurPage

Type:

long, read-only

Description:

The index of current PDF page. It starts from 0.

5) Rotate

Type:

short, read and write

Description:

Page rotation for PDF viewing. It can be set to:

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:

Page zoom level for PDF viewing. It can be between 10 and 1600. And can also be set to the following special values.

0 - Displaying PDF page in actual page size, which is the same as setting zoom level to 100%.

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

2 - Displaying PDF page with proper zoom level so that the page width can be fit to the client window.

7) CurrentTool

Type:

BSTR, read and write

Description:

The current tool of ActiveX. Possible values are:

"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"

* "ESignature Tool"

And so on.

Note:

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

8) Printer

Type:

IPDFPrinter, read-only

Description:

Returns an IPDFPrinter object, which can be used to specify the printer and do settings for PDF printing.

9) DocumentInfo

Type:

IPDFDocumentInfo*, read-only

Description:

Returns an IPDFDocumentInfo object with which you can use to retrieve PDF properties information, including Author, Subject, Creator, CreatedDate, Keywords, ModifiedDate, Producer and Title.

10) ActiveXVersion

Type:

BSTR, read-only

Description:

The version information of current ActiveX control.

11) *bHasFormFields

Type:

BOOL, read-only

Description:

A flag indicating whether current PDF contains form fields. It will be TRUE if the PDF contains form fields, otherwise FALSE.

12) *bHighlightFormFields

Type:

BOOL, read and write

Description:

A flag indicating whether to highlight all the interactive forms in PDF. All the interactive form fields will be highlighted when to set it to TRUE. The highlight could help the users to get better visual to the interactive form fields.

13) *FormFieldsHighlightAlpha

Type:

short, read and write

Description:

Transparency of the highlight color when highlighting the form filed in PDF. The value could be from 0 to 255. 0 means transparent and 255 means opaque.

14) *FormFieldsHighlightColor

Type:

OLE_COLOR, read and write

Description:

The highlight color of the form fields.

Method

1. Unlock

• UnLockActiveX

Unlock the ActiveX using license key received from Foxit.

Prototype:

Void UnLockActiveX(BSTR license_id, BSTR unlock_code)

Parameters:

license_id:	A string in the key file sent by Foxit after purchasing a license.
unlock_code:	A string in the key file sent by Foxit after purchasing a license.

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.

- **UnLockActiveXEx**

Unlock the ActiveX using license key received from Foxit.

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 UnlockActiveX.

- ***IsUnLocked**

Check if the ActiveX control is unlocked.

Prototype

boolean IsUnLocked()

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***RemoveEvaluationMark**

Remove the evaluation marks created by Foxit PDF ActiveX after your version has been unlocked by a license key.

Prototype:

BOOL RemoveEvaluationMark()

Parameters:

None

Return Value:

Return TRUE if successful, otherwise FALSE.

2. Global Settings

- **SetCurrentLanguage**

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

Prototype:

void SetCurrentLanguage(short LanguageID)

Parameters:

LanguageID: The identifier of a language which can be supported by ActiveX. The value could be from 1 to 34 and every value represents a language. Please refer to the following language list.

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 34 languages dynamically. This requires a corresponding XML language file to be in the same folder of ActiveX. To get a specific language file, please contact us at sales@foxitsoftware.com.

- **SetCurrentLanguageByString**

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

Prototype:

void SetCurrentLanguageByString(BSTR FileName)

Parameters:

FileName : File path of a XML language file. Please refer to the following language file list.

"Arabic"	-	"lang_ar_ae.xml"
"Bulgarian"	-	"lang_bg_bg.xml"
"Hungarian"	-	"lang_hu_hu.xml"
"Catalan"	-	"lang_ca_es.xml"
"Czech"	-	"lang_cz_cz.xml"
"Chinese-Simplified"	-	"lang_zh_cn.xml"
"Chinese-Traditional"	-	"lang_tw_cn.xml"
"Danish"	-	"lang_da_dk.xml"
"Dutch"	-	"lang_nl_nl.xml"
"English"	-	"lang_en_us.xml"
"Estonian"	-	"lang_et_ee.xml"
"Finnish"	-	"lang_fi_fi.xml"
"French"	-	"lang_fr_fr.xml"

"Galician"	-	"lang_gl_es.xml"
"German"	-	"lang_de_de.xml"
"Greek"	-	"lang_el_gr.xml"
"Italian"	-	"lang_it_it.xml"
"Korean"	-	"lang_ko_kr.xml"
"Latvian"	-	"lang_lv_lv.xml"
"Lithuanian"	-	"lang_lt_lt.xml"
"Norwegian"	-	"lang_nb_no.xml"
"Polish"	-	"lang_pl_pl.xml"
"Portuguese"	-	"lang_pt_pt.xml"
"Portuguese_Brazilian"	-	"lang_pt_br.xml"
"Romanian"	-	"lang_ro_ro.xml"
"Russian"	-	"lang_ru_ru.xml"
"Slovenian"	-	"lang_sl_si.xml"
"Spanish"	-	"lang_es_es.xml"
"Swedish"	-	"lang_sv_se.xml"
"Turkish"	-	"lang_tr_tr.xml"
"Hebrew"	-	"lang_he_il.xml"
"Japanese"	-	"lang_jp_jp.xml"
"Thai"	-	"lang_th_th.xml"
"Valencian"	-	"lang_va_es.xml"

Return Value:

None

Note:

The user interface of ActiveX can be switched to one of the 34 languages dynamically. This requires a corresponding XML language file in the same folder of ActiveX. If you want a specific language file, please contact us at sales@foxitsoftware.com.

- **SetModulePath**

Specify the path of CJK add-on (fpdfcjk.bin), which is used to help render the Chinese, Japanese and Korean text in a PDF.

Prototype:

void SetModulePath(LPCTSTR lpFolderName)

Parameters:

lpFolderName : File path of fpdfcjk.bin.

Return Value:

None

Note:

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

- **SetLogFile**

Specify a log file to record all the functions called by your ActiveX.

Prototype:

BOOL SetLogFile(BSTR filepath)

Parameters:

Filepath:	File path of a log which is used to record all the functions called by your ActiveX.

If the specified log does not exist, a new one will be created in the specified folder automatically.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **AboutBox**

Pop up About window of ActiveX.

Prototype:

Void AboutBox()

Parameters:

None

Return Value:

None

3. Open and Close PDF

- **OpenFile**

Open a PDF from a local disk or an HTTP server.

Prototype:

BOOL OpenFile (BSTR FilePath, BSTR Password)

Parameters:

FilePath:	File path of a local PDF file or URL of a PDF in HTTP server.
Password:	Password for opening the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

A PDF can still be operated by another application when it is opened by OpenFile. Because OpenFile will not lock the PDF.

- **OpenMemFile**

Open a PDF that is stored in memory.

Prototype:

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

Parameters:

pBuffer:	A pointer to a buffer specifying the PDF data .
Size:	The size of the buffer.
Password:	Password for opening PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **OpenBuffer**

Open a PDF from the buffer.

Prototype:

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

Parameters:

Buffer:	A pointer to the buffer(byte array) specifying the PDF data.
Size:	The size of the buffer.
Password:	Password for opening the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **OpenStream**

Open a PDF from the IStream interface.

Prototype:

BOOL OpenStream (IStream* Stream, BSTR Password)

Parameters:

Stream:	An IStream interface to the PDF data.
Password:	Password for opening the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **SetFileStreamOption**

Set the file stream option for the current PDF if it is frequently accessed.

Prototype:

Void SetFileStreamOption(BOOL bFileStream)

Parameter:

bFileStream:	A flag specifying whether to set the file stream option.

Return Value:

None

Note:

It will improve performance of loading the stream contents into memory if a PDF is frequently accessed. However it will consume more memory.

- **OpenCustomFile**

Open a PDF from a custom access descriptor.

Prototype:

BOOL OpenCustomFile(BSTR Password)

Parameter:

Password:	Password for opening the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

ActiveX will trigger CustomFileGetSize event and CustomFileGetBlock event when to call OpenCustomFile. With the events, the program will open a PDF from a custom way, returning the file size and a block of data. Please refer to CustomFileGetSize and CustomFileGetBlock for details.

- **OpenFtpFile**

Open a PDF 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 a PDF in FTP.
filePassword:	Password for opening the PDF.
Passive:	A flag specifying whether it is passive or active connection.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **CloseFile**

Close a PDF.

Prototype:

Void CloseFile()

Parameter:

None

Return Value:

None

4. Save

- **SaveAs**

Save the current PDF as a new local file.

Prototype:

Void SaveAs (BSTR FileName)

Parameters:

FileName: The local path of a new PDF which the current PDF will be saved as. It requires a local path.

Return Value:

None

- **Save**

Save the current PDF.

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.

- **SaveToStream**

Save the current PDF into memory.

Prototype:

IStream* SaveToStream ()

Parameters:

None

Return Value:

Return a pointer to an IStream to which the PDF data will be saved.

- ***SetShowSavePrompt**

Set to pop up the Save window when a PDF is to be closed.

Prototype:

void SetShowSavePrompt(boolean bShow, short Result)

Parameters:

bShow: A value indicating whether to pop up the window for saving.

Result: A value specifying to save PDF in background when bShow is set to FALSE. Valid values are 1 and 6.

Return Value:

NULL

5. Navigation

- **ExistForwardStack**

Check if the next view exists.

Prototype:

BOOL ExistForwardStack ()

Parameters:

None

Return Value:

Return TRUE if the next view exists, otherwise FALSE.

Note:

When navigating in a PDF, usually one may want to go back to the previous view. These methods help users to jump among different views conveniently.

View is a reading point or displaying status. The user actions will create new views. For example, if one goes to a new page and then zooms in the page, the two actions create two new views.

- **GoForwardStack**

Go to the next view.

Prototype:

Void GoForwardStack ()

Parameters:

None

Return Value:

None

- **ExistBackwardStack**

Check if the previous view exists.

Prototype:

BOOL ExistBackwardStack ()

Parameters:

None

Return Value:

Return TRUE if the previous view exists, otherwise FALSE.

- **GoBackwardStack**

Go to previous view.

Prototype:

Void GoBackwardStack ()

Parameters:

None

Return Value:

None

- **GetVisibleLeftTopPage**

Get the index of a PDF page which is at the top left.

Prototype:

LongGetVisibleLeftTopPage ()

Parameter:

None

Return Value:

Return the index of a PDF page which is displayed on the left top of ActiveX window. This can help get a page when viewing PDF in facing mode.

- **GoToNextPage**

Go to the next page of the current PDF.

Prototype:

Void GoToNextPage ()

Parameters:

None

Return Value:

None

- **GoToPrevPage**

Go to the previous page of the current PDF.

Prototype:

Void GoToPrevPage ()

Parameters:

None

Return Value:

None

- **GoToPage**

Go to a specific page of the current PDF.

Prototype:

Void GoToPage(long page_index)

Parameters:

page_index:	The index of a PDF page.

Return Value:

None

- **GoToPagePos**

Go to a specified position in current PDF.

Prototype:

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

Parameters:

NPageIndex:	The index of a PDF page.
PageX:	X coordinate (of PDF coordinate) in the specified PDF page.
PageY:	Y coordinate (of PDF coordinate) in the specified PDF page.

Return Value:

None

- **GotoPageDest**

Go to a specified location in a PDF document.

Prototype:

Void GotoPageDest (ILink_Dest * link_dest)

Parameters:

link_dest:	A pointer to an ILink_Dest object you get from event *OnHyperLink .

Return Value:

None

6. View

- **ShowDocumentInfoDialog**

Pop up Document Properties window of current PDF.

Prototype:

Void ShowDocumentInfoDialog()

Parameter:

None

Return Value:

None

- **SetPDFMeasureUnit**

Set the measurement unit for current PDF.

Prototype:

BOOL SetPDFMeasureUnit(short nType);

Parameters:

nType: The measurement unit. It can be set to:

0	-	Point
1	-	Inch
2	-	Centimeter
3	-	Pixel

Return Value:

Return TRUE if successful, otherwise FALSE.

- **GetPageWidth**

Get the width of a specific page in current PDF.

Prototype:

Float GetPageWidth(short nPageIndex)

Parameters:

nPageIndex:	The index of a PDF page.

Return Value:

Return the page width.

- **GetPageHeight**

Get the height of a specific page in current PDF.

Prototype:

Float GetPageHeight(short nPageIndex)

Parameters:

nPageIndex:	The index of a PDF page.

Return Value:

Return the page height.

- **CountTools**

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

Prototype:

Short CountTools()

Parameters:

None

Return Value:

Return total number of the tools in current ActiveX version.

- **GetToolByIndex**

Get the tool name by index.

Prototype:

BSTR GetToolByIndex (short nIndex)

Parameters:

nIndex:	The index of a tool. It can be set to: 0 <= nIndex< CountTools().

Return Value:

Return the name of an ActiveX tool specified by the index.

- **ScrollView**

Scroll the current view by dx and dy, using the device coordinate.

Prototype:

Void ScrollView (long dx, long dy)

Parameters:

dx:	The horizontal distance of scrolling.
dy :	The vertical distance of scrolling.

Return Value:

None

- **GetScrollLocation**

Get the current scroll location in current page.

Prototype:

Void GetScrollLocation (long *dx, long *dy)

Parameters:

dx:	[out]X coordinate of the current scroll location.
dy:	[out]Y coordinate of the current scroll location.

Return Value:

None

- **GetScrollLocationEx**

The extension of GetScrollLocation.

Prototype:

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

Parameters:

HPos:	[out]X coordinate of the current scroll location.
VPos:	[out]Y coordinate of the current scroll location.

Return Value:

None

- **SetViewRect**

Display a rectangle area of current PDF page.

Prototype:

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

Parameters:

Left:	X coordinate of the top left corner.
Top:	Y coordinate of the top left corner.
Width:	The rectangle width.
Height:	The rectangle height.

Return Value:

None

Note:

This method help display a specific rectangle area of current PDF page. It uses PDF coordinate, not device coordinate. And the unit is a PDF point. The function will not change the position and the size of current ActiveX window. It will only adjust the position and zoom factor for viewing so that the specified rectangle area of current PDF page can be fit to the ActiveX window.

Here is a typical use case. An end user uses the mouse to click and drag a rectangle area and then releases the mouse. The program calls ConvertClientCoodToPageCood to convert the device coordinates to PDF coordinates and then call SetViewRect to display the specified area of a PDF page to fit the ActiveX window.

- **ConvertClientCoordToPageCoord**

Converts device coordinate(inside client area of ActiveX window) to PDF coordinate.

Prototype:

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

Parameters:

nClientX:	X coordinate of device coordinate in the ActiveX window, in pixels.
nClientY:	Y coordinate of device coordinate in the ActiveX window, in pixels.
pnPageIndex:	[out]The index of PDF page where the point falls on.
pPageX:	[out]X coordinate of PDF coordinate.
pPageY:	[out]Y coordinate of PDF coordinate.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

The specified area may contain some ActiveX background area. And then it will fail to convert.

- **ConvertClientCoordToPageCoordEx**

Converts device coordinate(inside client area of ActiveX window) to PDF coordinate. This is the extension of ConvertClientCoordToPageCoord.

Prototype:

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

Parameters:

nClientX:	X coordinate of device coordinate in the ActiveX window, in pixels.
nClientY:	Y coordinate of device coordinate in the ActiveX window, in pixels.
pnPageIndex:	[out]The index of PDF page where the point falls on.
pPageX:	[out]X coordinate of PDF coordinate.
pPageY:	[out]Y coordinate of PDF coordinate.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

The client area may contain some ActiveX background area. And then it will fail to convert.

- **ConvertPageCoordToClientCoord**

Converts PDF coordinate to device coordinate(inside client area of ActiveX window).

Prototype:

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

Parameters:

nPageIndex:	The index of a PDF page.
dPageX:	X coordinate of PDF coordinate.
dPageY:	Y coordinate of PDF coordinate.
pnClientX:	[out]X coordinate of device coordinate in the ActiveX window. If the coordinate is out of the ActiveX window, it fails to convert.
pnClientY:	[out]Y coordinate of device coordinate in the ActiveX window. If the coordinate is out of the ActiveX window, it fails to convert.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

If no PDF is opened or the page index is incorrect, it will return FALSE.

- **ConvertCoordFromClientToPage**

Converts device coordinate (inside client area of ActiveX window) to PDF coordinate. This can be called by Javascript.

Prototype:

VARIANT ConvertCoordFromClientToPage(long nClientX, long nClientY);

Parameters:

nClientX:	X coordinate of device coordinate in the ActiveX window, in pixels.
nClientY:	Y coordinate of device coordinate in the ActiveX window, in pixels.

Return Value:

Returns an array, which includes 3 values. The first is the index of current page. The second is X coordinate of page coordinate and the third is Y coordinate of page coordinate.

- **ConvertCoordFromPageToClient**

Converts PDF coordinate to device coordinate (inside client area of ActiveX window). This can be called by Javascript.

Prototype:

VARIANT ConvertCoordFromPageToClient(long nPageIndex, float PageX, float PageY);

Parameters:

nPageIndex:	The index of a PDF page.
PageX:	X coordinate of PDF coordinate.
PageY:	Y coordinate of PDF coordinate.

Return Value:

Returns an array, which includes 2 values. The first is X coordinate of device coordinate and the second is Y coordinate of device coordinate in the ActiveX window.

- **SetBackgroudColor**

Set background color for PDF viewing.

Prototype

boolean SetBackgroudColor(OLE_COLOR color)

Parameters:

color:	The background color for viewing a PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **ShowPageShadow**

Show or hide page shadow for PDF viewing.

Prototype:

void ShowPageShadow(bool bShow);

Parameters:

bShow:	A value to indicate whether to show page shadow for PDF viewing.

Return Value:

None

- **ShowPageBorder**

Show or hide page border for PDF viewing.

Prototype:

void ShowPageBorder(bool bShow);

Parameters:

bShow:	A value to indicate whether to show page border for PDF viewing.

Return Value:

None

- **ShowPagesSpace**

Show or hide page space for PDF viewing in Facing or Continuous Facing mode.

Prototype:

void ShowPagesSpace (bool bShow);

Parameters:

bShow:	A value to indicate whether to show page space for PDF viewing in Facing mode.

Return Value:

None

7. Hyperlink

- **CountHyperLinks**

Count the hyperlink in a specific PDF page.

Prototype:

Short CountHyperLinks(short nPageIndex)

Parameters:

nPageIndex:	The index of a PDF page.

Return Value:

Return the hyperlink number if successful.

Return 0 if there is no hyperlink.

Return -1 if failing.

- **HighlightHyperLink**

Highlight a specific hyperlink.

Prototype:

Void HighlightHyperLink(short nPageIndex, short nLinkIndex)

Parameters:

nPageIndex:	The index of a PDF page.
nLinkIndex:	The index of a hyperlink.

Return Value:

None

- **GetHyperLinkRect**

Get the position information 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 PDF page.
nLinkIndex:	The index of a hyperlink.
top:	[out]The top coordinate of the hyperlink rectangle.
left:	[out]The left coordinate of the hyperlink rectangle.
bottom:	[out]The bottom coordinate of the hyperlink rectangle.
right:	[out]The right coordinate of the hyperlink rectangle.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **GetHyperLinkInfo**

Get the information of a specific hyperlink.

Prototype:

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

Parameters:

nPageIndex:	The index of a PDF page.
nLinkIndex:	The index of a hyperlink.
linktype:	[out]The hyperlink type.
linkdata:	[out]The string data of the hyperlink.

linkdest:	[out]The hyperlink destination.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

Currently linktype parameter only supports GoTo, GoToR, Lanuch and URI.

- **EnableHyperLink**

Enable the hyperlink.

Prototype:

EnableHyperLink(boolean bEnable)

Parameters:

bEnable: A flag specifying whether to enable hyperlink. It can be set to:

TRUE	-	Enable the hyperlink.
FALSE	-	Disable the hyperlink.

Return Value:

None

8. Text

- **GetPageText**

Extract the text from a specific PDF page.

Prototype:

BSTR GetPageText (long nPageIndex)

Parameters:

nPageIndex:	The index of a PDF page.

Return Value:

Return the text which is extracted.

- ***GetPageTextW**

Extract the text from a specific PDF page.

Prototype:

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

Parameters:

nPageIndex:	The index of a PDF page.
pBuffer:	[out]A pointer to the buffer which receives the text data.
nBuflen:	[out]The length of the text data.

Return Value:

Returns -1 if successful, otherwise 0.

- **GetSelectedRectsCount**

Count the text selection rectangles.

Prototype:

Long GetSelectedRectsCount()

Returned Value:

Return the total number of text selection rectangles.

- **GetSelectedRectByIndex**

Get a specific text selection rectangle.

Prototype:

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

Parameters:

nIndex:	The index of a text selection rectangle.
nPageIndex:	[out]The index of the page where the specified text selection rectangle is.
Left:	[out]The left coordinate of the text selection rectangle.
Bottom:	[out]The bottom coordinate of the text selection rectangle.
Right:	[out]The right coordinate of the text selection rectangle.
Top:	[out]The top coordinate of the text selection rectangle.

Returned Value:

Return TRUE if successful, otherwise FALSE.

- **GetSelectedText**

Get the text which is currently selected.

Prototype:

BSTR GetSelectedText ();

Parameters:

None

Return Value:

Return the text which is currently selected.

- **GetSelectedTextEx**

This is the extension of GetSelectedText.

Prototype:

long GetSelectedTextEx(long* pBuffer, long nBuflen)

Parameters:

pBuffer:	[out]A pointer to the buffer which receives the selected text.
nBufLen:	The size of the buffer.

Return Value:

Return the buffer length of the selected text.

- ***SetSelectTextHighlightColor**

Set the highlight color for the text selection.

Prototype

void SetSelectTextHighlightColor(OLE_COLOR HighLihgtColor, short Alpah)

Parameters:

HighLihgtColor:	The highlight color for the selected text.
Alpah:	The transparency for the selected text. The value could be from 0 to 255. 0 means transparent and 255 means opaque.

Return Value:

NULL

- ***LoadPDFTextDoc**

Load a text document from a PDF.

Prototype:

IPDFTextDoc*LoadPDFTextDoc(BSTR filePath, BSTR filePass)

Parameters:

filePath:	The path of a PDF. It can be a URL.
filePass:	The password for opening PDF.

Return Value:

Return a pointer to an IPDFTextDoc object which contains the text document data.

- ***ExtractTextFromPDF**

Convert a PDF to a TXT File.

Prototype:

BOOL ExtractTextFromPDF(BSTR sourcePath, BSTR sourcePass, BSTR destPath)

Parameters:

sourcePath:	File path of a source PDF.
sourcePass:	The password for accessing PDF.

destPath: The path of a TXT file to be converted. If the TXT file does not exist, a new one will be created.

Return Value:

Return TRUE if successful, otherwise FALSE.

9. Custom UI

- **ShowTitleBar**

Show or hide the title bar.

Prototype:

void ShowTitleBar(BOOL bShow)

Parameters:

bShow: A flag specifying specify whether to show the title bar.

TRUE	-	Show the title bar.
FALSE	-	Hide the title bar.

Return Value:

None

- **ShowStatusBar**

Show or hide the status bar.

Prototype:

void ShowStatusBar(BOOL bShow)

Parameters:

bShow: A flag specifying whether to show the status bar.

TRUE	-	Show the status bar.
FALSE	-	Hide the status bar.

Return Value:

None

- **ShowToolBar**

Show or hide the toolbar.

Prototype:

void ShowToolBar(BOOL bShow);

Parameters:

bShow: A flag specifying whether to show the toolbar.

TRUE	-	Show the toolbar.

FALSE - Hide the toolbar.

Return Value:

None

- **ShowToolbarButton**

Show or hide a specific toolbar button.

Prototype:

void ShowToolbarButton(short nIndex, BOOL bShow)

Parameters:

nIndex:	The index of a toolbar button.
bShow:	A flag specifying whether to show the toolbar button.

TRUE - Show the toolbar button.

FALSE - Hide the toolbar button.

Return Value:

None

- **EnableToolTip**

Show or hide the tooltips.

Prototype:

boolean EnableToolTip(BOOL bEnable)

Parameters:

bEnable: A flag specifying whether to show the tooltips.

TRUE - Show the tooltips.

FALSE - Hide the tooltips.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **GetPanelStatus**

Get the status of navigation panel.

Prototype:

BOOL GetPanelStatus()

Parameters:

None

Return Value:

Returns TRUE if the panel is shown, otherwise FALSE.

- **ShowNavigationPanels**

Show or hide the navigation panel.

Prototype:

BOOL ShowNavigationPanels(BOOL bShow)

Parameters:

bShow: A flag specifying whether to show the navigation panel.

TRUE	-	Show the navigation panel.
FALSE	-	Hide the navigation panel.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **ShowNavPanelByString**

Show a navigation panel specified by the name.

Prototype:

BOOL ShowNavPanelByString(LPCTSTR lpszPanelName)

Parameters:

lpszPanelName: The panel name. There are four panels, including:

"Bookmarks"

"Pages"

"Layer"

"Attachments"

Return Value:

Return TRUE if successful, otherwise FALSE.

- **SetLayoutShowMode**

Set the page layout for PDF viewing.

Prototype:

Void SetLayoutShowMode (BrowseMode nShowMode, short nFacingCount);

Parameters:

nShowMode: The displaying mode of the PDF pages. It can be set to:

0	-	MODE_SINGLE
1	-	MODE_CONTINUOUS

nFacingCount: The number of pages that are displayed horizontally.

Return Value:

None

- **GetLayoutShowMode**

Get the current layout mode.

Prototype:

Void GetLayoutShowMode(short* pnShowMode, short* pnFacingCount)

Parameters:

pnShowMode:	[out]The current displaying mode.
PnFacingCount:	[out]The number of pages that are displayed horizontally.

Return Value:

None

- **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 flag specifying whether to render cover page to be on the left.

Return Value:

None

- **SetDisplayBackgroundColor**

Set the background color of ActiveX control.

Prototype:

void SetDisplayBackgroundColor(long clrArgb)

Parameters:

clrArgb:	The background color of ActiveX control. It must be decimal ARGB format.

Return Value:

NULL

- ***ShowContextMenu**

Show or hide the context menus.

Prototype:

Void ShowContextMenu(BOOL bShow)

Parameters :

bShow:	A flag specifying whether to show the context menus.

Return Value:

None

Note:

If setting bShow to TRUE, it will show the context menus. Otherwise it will not.

- **ShowTextSelectionMenu**

Show or hide the context menus for text selection tool.

Prototype:

Void ShowTextSelectionMenu(BOOL bShow);

Parameters:

bShow:	A flag specifying whether to show context menus for text selection tool.

Return Value:

None

Note:

ShowTextSelectionMenu is used by text selection tool and ShowContextMenu is used by hand tool.

- ***ShowFormFieldsMessageBar**

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

Prototype:

void ShowFormFieldsMessageBar(BOOL bShow)

Parameters:

bShow: A flag specifying whether to show the message bar for form fields.

TRUE	-	Show the message bar.
FALSE	-	Hide the message bar.

Return Value:

None

- ***SetSigContextMenuString**

Set context menus for a signature field.

Prototype:

void SetSigContextMenuString(BSTR string)

Parameters:

string: The context menu(s). If there are multiple context menus, the two items could be separated by a comma.

Return Value:

None

- ***EnableBookmarkEdit**

Enable bookmark editing.

Prototype:

void EnableBookmarkEdit(boolean bEdit)

Parameters:

bEdit:	A flag specifying whether the bookmark is editable.

Return Value:

NULL

- **SetPagesContextMenuString**

Set context menu(s) for thumbnail in Page panel.

Prototype:

void SetPagesContextMenuString(BSTR string)

Parameters:

string:	The context menu(s) and two values could be separated by a comma.

Return Value:

NULL

- **SetBookmarkContextMenuString**

Set context menu(s) for bookmark editing in Bookmark panel.

Prototype:

void SetBookmarkContextMenuString(BSTR string)

Parameters:

string:	The context menu(s) and two values could be separated by a comma.

Return Value:

NULL

- **ShowStdBookmarkContextMenu**

Show or hide a default bookmark context menu.

Prototype:

void ShowStdBookmarkContextMenu(short nIndex, boolean bShow)

Parameters:

nIndex:	The index of a default bookmark context menu. The value is from 1.
bShow:	A flag specifying whether to show the context menu.

Return Value:

NULL

- **ShowStdPagesContextMenu**

Show or hide a default page context menu.

Prototype:

void ShowStdPagesContextMenu(short nIndex, boolean bShow)

Parameters:

nIndex:	The index of a default page context menu. The value is from 1.
bShow:	A flag specifying whether to show the context menu.

Return Value:

NULL

10. Print

- **PrintWithDialog**

Pop up print dialog for PDF printing.

Prototype:

Void PrintWithDialog()

Parameters:

None

Return Value:

None

- **OpenMemFileForPrinter**

Print a memory PDF file without displaying it.

Prototype:

IPDFPrinter* OpenMemFileForPrinter(long buffer, long size)

Parameters:

Buffer:	A pointer to the buffer specifying the PDF data.
Size:	The size of the buffer.

Return Value:

Return a pointer to an IPDFPrinter object that you can use to set for PDF printing.

- **OpenFileForPrinter**

Print a PDF file without displaying it.

Prototype:

IPDFPrinter* OpenFileForPrinter(BSTR file_path)

Parameters:

file_path:	File path of a PDF to be printed.

Return Value:

Return a pointer to an IPDFPrinter object that you can use to set for PDF printing.

- ***PrintPopupAnnot**

Set whether to print out the content of pop-up annotations in a current PDF.

Prototype

void PrintPopupAnnot(boolean bPrint)

Parameters:

bPrint:	A flag specifying whether to print out the content of pop-up annotations.

Return Value:

NULL

11. Search

- **FindFirst**

Get the first match for text search in current PDF.

Prototype:

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

Parameters:

SearchString:	The text keyword you want to search.
BMatchCase:	A flag specifying whether the search is case sensitive.
BMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

It will jump to the PDF page where the first match is found. And then update CurPage, highlight the match and return TRUE. Otherwise FALSE.

- **FindFirstEx**

Get the first match for text search in current PDF. This is the extension of FindFirst.

Prototype:

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

Parameters:

search_string:	The text keyword you want to search.
BMatchCase:	A flag specifying whether the search is case sensitive.
BMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return TRUE if successful, otherwise FALSE.

- **FindNext**

Get the next match for text search specified by Findfirst.

Prototype:

BOOL FindNext (BOOL bSearchDown);

Parameters:

bSearchDown:	A flag specifying to search down or search up.		
	TRUE	-	Search down.
	FALSE	-	Search up.

Return Value:

Return TRUE if the next match is found, otherwise FALSE.

Note:

It will go to the PDF page where the next match is found. And then update CurPage, highlight the match and return TRUE. Otherwise FALSE. Please note that FindNext uses the same search settings, including bMatchCase and bMatchWholeWord, as FindFirst.

- **FindFileFirst**

Get the first match for text search in a specific PDF.

Prototype:

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

Parameters:

file_path:	File path of a PDF.
search_string:	The text keyword you want to search.
bmatchCase:	A flag specifying whether the search is case sensitive.
BMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return a pointer to an IFindResult object if the first match is found, otherwise NULL.

Note:

This method helps to search a text in a PDF without opening it. For example, to search for a keyword in all the PDFs in a given folder, you may iterate all the PDFs in the folder and then search for the keyword in them one by one. Once ActiveX find a match in a PDF, it returns IFindResult which includes the detailed information of the match. Then one may call GoToSearchResult to open the PDF, go to the destination page and highlight the match.

- **FindFileFirstEx**

Get the first match for text search in a specific PDF. This is the extension of FindFileFirst.

Prototype:

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

Parameters:

file_path:	File path of a PDF.
Password:	Password for opening the PDF where the first match is.
search_string:	The text keyword you want to search.
bMatchCase:	A flag specifying whether the search is case sensitive.
BMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return a pointer to an IFindResult object if the first match is found, otherwise NULL.

- **FindFileNext**

Get the next match for text search in the PDF specified by FindFileFirst.

Prototype:

IFindResult *FindFileNext ();

Parameters:

None

Return Value:

Return a pointer to an IFindResult object if the next match is found, otherwise NULL.

- **GoToSearchResult**

Display and highlight the search result.

Prototype:

Void GoToSearchResult (IFindResult* findresult);

Parameters:

Findresult:	[out]A pointer to an IFindResult object returned by FindFileFirst or FindFileNext.

Return Value:

None

- **FindPageFirst**

Get the first match for text search on a PDF page.

Prototype:

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

Parameters:

nPageIndex:	The index of a page in current PDF.
Search_string:	The text keyword you want to search.
bMatchCase:	A flag specifying whether the search is case sensitive.
bMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return a pointer to an IFindResult object, otherwise NULL.

- **FindPageNext**

Get the next match for text search specified by FindPageFirst.

Prototype:

IFindResult* FindPageNext()

Parameters:

None

Return Value:

Return a pointer to an IFindResult object if successful, otherwise NULL.

Note:

Call FindPageFirst to initialize text search, and then call FindPageNext to get the next match.

- **FindMemFileFirst**

Get the first match for text search in a PDF which is in memory.

Prototype:

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

Parameters:

Buffer:	A pointer to a buffer specifying the PDF data.
fileSize:	The size of PDF data.
Password:	Password for opening PDF.
Search_string:	The text keyword you want to search.

bMatchCase:	A flag specifying whether the search is case sensitive.
bMatchWholeWord:	A flag specifying whether to search whole word only.

Return Value:

Return a pointer to an IFindResult object if successful, otherwise NULL.

- **FindClose**

Release memory allocated by FindMemFileFirst.

Prototype:

Void FindClose()

Parameter:

None

Return:

None

- ***SetSearchHighlightFillColor**

Set the highlight color for the search result.

Prototype

void SetSearchHighlightFillColor(OLE_COLOR FillColor, short Alpha)

Parameters:

FillColor:	The highlight color of a current search result.
Alpah:	The transparency of a current search result. The value could be from 0 to 255. 0 means transparent and 255 means opaque.

Return Value:

None

- ***SearchAndHighlightAllTextOnPage**

Highlight all matches for a text search in a specific PDF page.

Prototype:

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

Parameters:

searchstring:	The text keyword you want to search.
bMatchCase:	A flag specifying whether the search is case sensitive.
bMatchWholeWord:	A flag specifying whether to search whole word only.
PageNo:	The index of a PDF page.

Return Value:

None

- **ShowSearchNotFoundPrompt**

Show or hide the message for text search.

Prototype:

Void ShowSearchNotFoundPrompt (boolean bShow)

Parameters:

bShow:	A value indicating whether to show the message for text search.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This only shows or hides "No matches were found" for FindFirst.

12. Security

- **GetDocPermissions**

Get file permission of current PDF.

Prototype:

Long GetDocPermissions ()

Parameter:

None

Return Value:

- Return a 32bit integer which indicates the PDF permissions. Please refer to PDF Reference document to get the detailed description about document permission.
- Return 0xffffffff if the PDF is not protected.

- *** CheckOwnerPassword**

Check the owner password of the PDF.

Prototype:

BOOL CheckOwnerPassword(BSTR lpszPermPsw)

Parameters:

lpszPermPsw:	The owner password.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetUserPassword**

Set user password for current PDF.

Prototype:

BOOL SetUserPassword(LPCTSTR lpszNewValue)

Parameters:

lpszNewValue:	A user password for the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetOwnerPassword**

Set owner password for current PDF.

Prototype:

BOOL SetOwnerPassword(LPCTSTR lpszNewValue)

Parameters:

lpszNewValue:	An owner password for the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetUserPermission**

Set user permission for current PDF.

Prototype:

BOOL SetUserPermission(long dwPermission)

Parameters:

dwPermission:	User permission flag.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

PDF document permissions could be 32bit integer. Please refer to PDF Reference document to get the detailed description about document permission.

13. JavaScript

- *** RunJScript**

Execute a JavaScript.

Prototype:

Void RunJScript(LPCTSTR csJS)

Parameters:

scJS: The JavaScript to be executed.

Return Value:

None

- ***ShowDocJsDialog**

Pop up the document JavaScript dialog.

Prototype:

Void ShowDocJsDialog();

Parameters:

None

Return Value:

None

- ***ShowJsConsoleDialog**

Pop up JavaScript Console Dialog.

Prototype:

Void ShowJsConsoleDialog();

Parameters:

None

Return Value:

None

- **DisableJavaScript**

Disable JavaScript actions.

Prototype:

void DisableJavaScript(bool bDisable);

Parameters:

bDisable: A value to indicate whether to disable JavaScript actions.

Return Value:

None

- **EnableSafeMode**

Enable safe mode.

Prototype:

void EnableSafeMode(bool bEnable);

Parameters:

bEnable: A value to indicate whether to enable safe mode.

Return Value:

None

14. Multi-Instances

- **SetCurrentWnd**

Set a current instance when ActiveX runs multi-instance.

Prototype:

Void SetCurrentWnd(long hWnd)

Parameters:

hWnd: The handle of an ActiveX instance.

Return Value:

None

- **GetCurrentWnd**

Get the handle of the current ActiveX window.

Prototype:

Long GetCurrentWnd()

Parameters:

None

Return Value:

Return the handle of the current ActiveX window if successful, otherwise Null.

- **GetCtrlInstance**

Get the handle of the current ActiveX instance.

Prototype:

Long GetCtrlInstance ()

Parameters:

None

Return Value:

Return the handle of the current ActiveX instance if successful, otherwise NULL.

15. Edit and Page Organize

- **GetPageRotation**

Get the rotation of a specific PDF page.

Prototype:

long GetPageRotation(long page_index)

Parameters:

page_index: The index of a PDF page.

Return Value:

Return a value indicating the page rotation. Possible values are:

0	-	0
1	-	90
2	-	180
3	-	270

- ***InsertNewPage**

Insert a new blank page to current PDF.

Prototype:

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

Parameters:

nPageIndex:	The index of a page before which a new blank page will be inserted.
nPageWidth:	Page width of the new blank page.
nPageHeight:	Page height of the new blank page.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***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 index of a page before which the new PDF page(s) will be inserted.
lpszPDFFileName:	Path of a source PDF where the new PDF pages are from.
lpszPssword:	Password for accessing the source PDF.
lpszPageRangeString:	The range of pages which are from the source PDF and will be inserted to current PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***DeletePage**

Delete one or more pages from current PDF.

Prototype:

Boolean DeletePage(long pageIndex, long count)

Parameters :

pageIndex:	The index of the first page to be deleted.
Count:	The total number of the pages to be deleted.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SwapPage**

Swap two PDF pages.

Prototype:

boolean SwapPage(long pageIndex1, long pageIndex2)

Parameters:

pageIndex1:	The index of the first page to be swapped.
pageIndex2:	The index of the second page to be swapped.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetPageIndex**

Set a new position for a specific PDF page.

Prototype:

Boolean SetPageIndex (long pageOldIndex, long pageNewIndex);

Parameters:

PageOldIndex:	The current index of a PDF page.
PageNewIndex:	A new index for the PDF page.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***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 index of a page in current PDF.
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:

Return TRUE if successful, otherwise FALSE.

- *** SetPageRotate**

Rotate a PDF page.

Prototype:

Boolean SetPageRotate(long pageIndex, long rotate);

Parameters:

pageIndex:	The index of a PDF page.
rotate:	PDF page rotation.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***FlattenPage**

Flatten a PDF page.

Prototype:

Boolean FlattenPage(long pageIndex);

Parameters:

pageIndex:	The index of a PDF page.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ExportPagesToPDF**

Export some pages from current PDF to get a new PDF.

Prototype:

BOOL ExportPagesToPDF(BSTR lpszPDFFileName, BSTR lpszPageRangeString);

Parameters:

lpszPDFFileName: File path for a new PDF which will be created with the exported pages.

lpszPageRangeString: The range of pages to be exported from current PDF. The value could be in format of "0”, “2”, “3-5". Please note that "5-2" is invalid. To sum up, the number before the dash should be less than the one after the dash.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ImportImageToPdf**

Add an image to a PDF page.

Prototype:

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

Parameters:

pdfFilePath:	The path of a PDF where the image will be added.
index:	The index of a PDF page where the image file will be added.
imageFilePath:	Path of an image which will be imported to the PDF.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

It supports six image types, including bmp, jpg, png, tif, gif and jpx.

- ***AddImageObject**

Add an image to a PDF page.

Prototype:

BOOL AddImageObject (long nPageIndex, float left, float bottom, float width, float height, BSTR BmpFileName, short alpha, short rotate)

Parameters:

nPageIndex:	The index of a PDF page where the image will be added.
Left:	X coordinate of the image to be added.
Bottom:	Y coordinate of the image to be added.
Width:	The width of the image to be rendered in the page.
height:	The height of the image to be rendered in the page.
BmpFileName:	File path of an image to be added.
Alpha:	Alpha value for the image. It can be from 0 to 255.
Rotate:	Rotation of the image.

Return Value:

Return TRUE if successful, otherwise FALSE.

- *** ConvertPDFPageToImage**

Convert a PDF page to an image.

Prototype:

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

Parameters:

pdfFile :	Path of a PDF whose page will be converted to image.
lnPageIndex:	The index of a page which will be converted to image.
imageFilePath:	File path for the image which will be generated from the specific PDF page. It supports six image types, including bmp, jpg, png, tif, gif and jpx.
lnImageWidth:	The width of the image to be converted.
lnImageHeight:	The height of the image to be converted.

Return Value:

Return TRUE if successful, otherwise FALSE.

16. Asynchronous feature

- **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.
strUserName:	Username for connecting to FTP.
strUserPassword:	Password for connecting to FTP.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetAsyncFileLen**

Set HTTP file length which will be loaded by the asynchronous way.

Prototype:

Void SetAsyncFileLen(long lnSize)

Parameters:

lnSize:	The file size.

Return Value:

None

Note:

*SetAsyncFileLen should be used together with *SetAsyncFileData, *OpenAsyncFile and *OnFetchAsyncFileData.

- ***SetAsyncFileData**

Set the asynchronous file data.

Prototype:

Void SetAsyncFileData(long lnFileBuffer, long offset, long size)

Parameters:

lnFileBuffer:	A pointer to the buffer specifying the request data.
offset:	Offset of the requested data.
Size:	Byte size of the requested data.

Return Value:

None

Note:

*SetAsyncFileData should be used together with *SetAsyncFileLen, *OpenAsyncFile and *OnFetchAsyncFileData.

- ***OpenAsyncFile**

Start to open the http file in asynchronous way.

Prototype:

Void OpenAsyncFile(LPCTSTR strPDFPassword)

Parameters:

strPDFPassword:	A password for opening a http PDF file.

Return Value:

None

Note:

*OpenAsyncFile should be used together with *SetAsyncFileData, *SetAsyncFileLen and *OnFetchAsyncFileData.

17. Bookmark

- **GetOutlineFirstChild**

Get the first child item of an outline.

Prototype:

IPDFOutline*GetOutlineFirstChild(IPDFOutline* Outline)

Parameters:

Outline: A pointer to an IPDFOutline object whose first child item will be returned. It will return the root item when to set this to NULL.

Return Value:

Return a pointer to the first child item if the specified outline has child item, otherwise NULL.

- **GetOutlineNextSibling**

Get next sibling item.

Prototype:

IPDFOutline*GetOutlineNextSibling(IPDFOutline* Outline)

Parameters:

Outline: A pointer to an IPDFOutline object whose next sibling item will be returned.

Return Value:

Return a pointer to the next sibling item if it exists, otherwise NULL.

18. Form

- ***ExportFormToFDFFile**

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

Prototype:

BOOL ExportFormToFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName: Path of a FDF to which the form data will be exported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ImportFormFromFDFFile**

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

Prototype:

BOOL ImportFormFromFDFFile(BSTR FDFFileName)

Parameters:

FDFFileName: Path of a FDF from which the form data will be imported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***FindFormFieldsTextFirst**

    Get the first match for text search in form fields.

Prototype:

    BOOL FindFormFieldsTextFirst (BSTR searchstring, BOOL bMatchCase)

Parameters:

Searchstring:	The text keyword you want to search.
bMatchCase:	A flag specifying whether the search is case sensitive.

Return Value:

    Return TRUE if successful, otherwise FALSE.

- ***FindFormFieldsTextNext**

    Get the next match for text search specified by FindFormFieldsTextFirst.

Prototype:

    BOOL FindFormFieldsTextNext ()

Parameters:

    None

Return Value:

    Return TRUE if successful, otherwise FALSE.

- *** SubmitForm**

    Submit PDF form data to a specific destination.

Prototype:

    BOOL SubmitForm(BSTR csDestination)

Parameters:

csDestination:	The destination for submitting the form data.

Return Value:

    Return TRUE if successful, otherwise FALSE.

- **#GetCurrentForm**

    Get the current form field in PDF.

Prototype:

    IPDFForm*GetCurrentForm ()

Parameters:

    None

Return Value:

Return a pointer to an IPDFForm object if successful, otherwise NULL.

19. Annotation

- **ForceRefresh**

Refresh after adding highlights.

Prototype:

Void ForceRefresh()

Note:

This is used to increase the efficiency of adding highlights. In the old version, ActiveX refresh page once a highlight is added. So it will refresh many times if several highlights are added. From version 5.0, ActiveX separate refresh from adding highlights. It requires to call ForceRefresh to refresh the page after adding highlights.

- ***Highlight**

Highlight a specific rectangle area on a PDF page.

Prototype:

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

Parameters:

nPageIndex:	The index of a PDF page.
left:	X coordinate of the top left corner in a rectangle area.
top:	Y coordinate of the top left corner in a rectangle area
right:	X coordinate of the bottom right corner in a rectangle area.
bottom:	Y coordinate of the bottom right corner in a rectangle area.

Return Value:

None

- ***RemoveAllHighlight**

Remove all highlights in current PDF.

Prototype:

Void RemoveAllHighlight()

Parameters:

None

Return Value:

None

- ***ImportAnnotsFromFDFFile**

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

Prototype:

BOOL ImportAnnotsFromFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName:	File path of a FDF from which the comment will be imported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ImportAnnotsFromFDFFileEx**

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

Prototype:

boolean ExportAnnotsToFDFFileEx(long PageIndex, BSTR FDFFileName)

Parameters:

PageIndex:	The index of a PDF page to which the comments will be imported.
FDFFileName:	File path of a FDF from which the comment will be imported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ExportAnnotsToFDFFile**

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

Prototype:

BOOL ExportAnnotsToFDFFile (BSTR FDFFileName)

Parameters:

FDFFileName:	File path of a FDF to which the comments will be exported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***ExportAnnotsToFDFFileEx**

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

Prototype:

boolean ExportAnnotsToFDFFileEx(long PageIndex, BSTR FDFFileName)

Parameters:

PageIndex:	The index of a PDF page whose comments will be exported.
FDFFileName:	File path of a FDF to which the comments will be exported.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***SetBDrawAnnot**

Set to show the PDF annotations or not.

Prototype:

Void SetBDrawAnnot(BOOL bDrawAnnot);

Parameters:

bDrawAnnot: A flag specifying whether to show the PDF annotations.

TURE	-	Show the PDF annotations.
FALSE	-	Not to show the PDF annotations.

Return Value:

None

- ***ShowAllPopup**

Show popup for all the markup annotations.

Prototype:

Void ShowAllPopup (BOOL bShow);

Parameters:

bShow: A flag specifying whether to show the popup.

TRUE	-	Show the popup.
FALSE	-	Hide the popup.

Return Value:

None

- ***ReleasePageAnnots**

Release the annotation object returned by GetPageAnnots.

Prototype

void ReleasePageAnnots(IPDFPageAnnots* pageAnnots)

Parameters:

pageAnnots: A pointer to a page annotation object.

Return Value:

NULL

- ***SetAnnotDefaultProperties**

Set default properties for adding Pencil annotation.

Prototype:

void SetAnnotDefaultProperties(BSTR string);

Parameters:

string:	The properties information of a Pencil annotation object.

Return Value:

None

- **^GetPageAnnots**

Get all the annotation objects from a PDF page.

Prototype:

IPDFPageAnnots*GetPageAnnots(long pageIndex)

Parameters:

pageIndex :	The index of a PDF page.

Return Value:

Return a pointer to an IPDFPageAnnots that can access and process the annotations in a specified PDF page if succeed, otherwise NULL.

- **^GetFormatTool**

Get the format tool for the PDF annotations.

Prototype:

IPDFormatTool*GetFormatTool()

Parameters:

None

Return Value:

Return a pointer to an IPDFormatTool object if successful, otherwise NULL.

20. Signature

- **&GetPDFSignatureMgr**

Get all the signatures.

Prototype:

IPDFSignatureMgr* GetPDFSignatureMgr();

Parameters:

None

Return Value:

Return a pointer to an IPDFSignatureMgr object.

- **&GetLastSigModuleError**

Get the last error of the signature.

Prototype:

long GetLastSigModuleError()

Parameters:

None

Return Value:

Return a value indicating the last error of the signature.

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

- **&GetLastSigModuleErrMsg**

Get the last error of the signature.

Prototype:

BSTR GetLastSigModuleErrMsg()

Parameters:

None

Return Value:

Return the error message of the signature.

21. Others

- ***AddPrintMark**

Add text watermarks to a PDF for printing.

Prototype:

boolean AddPrintMark(BSTR string, float center_x, float center_y, BSTR fontname, short lfCharSet, short fontsize, OLE_COLOR fontcolor, short textmode, short alpha, short rotate)

Parameters:

string:	The text of a watermark which will be printed in the PDF.
center_x:	X coordinate in PDF page coordinate system.
center_y:	Y coordinate in PDF page coordinate system.
fontname:	The font of the text watermark to be printed.

lfCharSet: The character encoding of the text watermark. Here is a list of pre-defined values.

ANSI_CHARSET,

BALTIC_CHARSET,

CHINESEBIG5_CHARSET,

DEFAULT_CHARSET(the one which is based on the OS),

EASTEUROPE_CHARSET,

GB2312_CHARSET,

GREEK_CHARSET,

HANGUL_CHARSET,

MAC_CHARSET,

OEM_CHARSET(the one which depends on the OS),

RUSSIAN_CHARSET,

SHIFTJIS_CHARSET,

SYMBOL_CHARSET,

TURKISH_CHARSET,

fontsize:	The font size of the text watermark to be printed.
fontcolor:	The font color of the text watermark to be printed.
textmode:	The mode of the text watermark. The value must be 0, 1 or 2.

0	-	fill the text
1	-	stroke the text
2	-	fill then stroke the text

alpha: The transparency of the text watermark. The value could be from 0 to 255. 0 means transparent and 255 means opaque.

rotate: The rotation of the text watermark.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***UploadCurFileToFTP**

Upload a PDF 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:	Path of a PDF to be uploaded to FTP.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***IsDualPage**

Check if a specific PDF page contains two layers.

Prototype:

BOOL IsDualPage(short pageIndex)

Parameters:

pageIndex:	The index of a PDF page.

Return Value:

Return TRUE if the specified PDF page contains two layers.

Note:

Two layers mean a page has an image with hidden text.

- ***AddWaterMark**

Insert a text watermark of a specific PDF page.

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 index of a page in current PDF where the watermark will be added.
String:	The text content of the watermark.
Left:	X coordinate for placing the text watermark.
Bottom:	Y coordinate for placing the text watermark.
Fontsize:	Font size of the watermark's text.
Fontcolor:	Text color of the text watermark.
Textmode:	It can be set to 0, 1 or 2.
	0 - fill the text
	1 - stroke the text
	2 - fill then stroke the text
alpha:	Alpha value of the text watermark. It can be from 0 to 255.
rotate:	Rotation of the text watermark.

Return Value:

Return TRUE if successful, otherwise FALSE.

- ***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 index of a page to be rendered as a bitmap.
pixelWidth:	The width of the bitmap.
PixelHeight:	The height of the bitmap.
RectLeft:	The left coordinate of the display rectangle area in device coordinate.
RectTop:	The top coordinate of the display rectangle area in device coordinate.
rectRight:	The right coordinate of the display rectangle area in device coordinate.
RectBottom:	The bottom coordinate of of the display rectangle area in device coordinate.
PixelFormat:	The pixel format of the bitmap to be rendered.

Return Value:

Return a handle for the bitmap.

- ***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 :	Coding format. It supports the following ones.

UNSPECIFY	=	-1,
CODE_39	=	1
CODE_128	=	3
EAN_8	=	6
UPC_A	=	7
EAN_13	=	8
ITF	=	9

When choosing ITF, length of content must be one of the following values:8,10,12,16,20,24,44.

QR_CODE	=	15

moduleHSize:	The width of the barcode.
moduleVSize:	The height of the barcode.
ecLevel:	Merely effective to the QR Code type.

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

Return Value:

Return a 2D image if successful, otherwise NULL.

- ***SetContextMenuString**

Set the context menu.

Prototype:

Void SetContextMenuString(BSTR string);

Parameters:

string:	Context menus to be added.

Return Value:

None

Note:

This method could be used together with OnContextMenuIndex.

Event

1) BeforeDraw

Triggered before rendering the contents.

Prototype:

Void BeforeDraw (long dc)

Parameters:

dc:	Handle to a device context.

2) AfterDraw

Triggered after rendering the contents is completed.

Prototype:

Void AfterDraw (long dc)

Parameters:

dc:	Handle to a device context.

3) OnZoomChange

Triggered once Zoomlevel has been changed.

Prototype:

Void OnZoomChange ()

Parameters:

None

4) OnPageChange

Triggered once the current page has been changed.

Prototype:

Void OnPageChange()

Parameters:

None

5) OnOpenPassword

Triggered when to open a PDF which is password protected.

Prototype:

Void OnOpenPassword (BSTR* password, BOOL* cancel)

Parameters:

Password:	Password for opening the PDF.
Cancel:	A flag specifying whether to cancel PDF opening if the input password is incorrect.

6) OnSearchProgress

Triggered when to do text search.

Prototype:

Void OnSearchProgress (long pageNumber, long pageCount)

Parameters:

pageNumber:	The index of a page from which it starts to search.
pageCount:	Total number of PDF pages to be searched.

7) OnOpenFile

Triggered when failing to open a PDF.

Prototype:

Void OnOpenFile(short Error)

Parameters:

Error:	Error code to be returned.

8) OnOpenDocument

Triggered when to open a PDF.

Prototype:

Void OnOpenDocument (BSTR filepath)

Parameters:

Filepath:	File path of a PDF to be opened.

9) OnFilePathInvalidate

Triggered when failing to open a PDF by file path.

Prototype:

Void OnFilePathInvalidate(BSTR WarnString);

Parameters:

WarnString:	Error message for PDF opening.

10) OnShowSavePrompt

Triggered when to close a PDF which has been modified.

Prototype:

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

Parameters:

bShow:	A flag specifying whether to show the default message box.
nResult:	A flag specifying whether to save the modified PDF. It will not save by default.

11) OnCloseDocument

Triggered when to close a PDF.

Prototype:

Void OnCloseDocument (BSTR filepath)

Parameters:

Filepath:	File path of a PDF to be closed.

12) OnDocumentChange

Triggered when the current PDF has been modified.

Prototype:

Void OnDocumentChange ()

Parameters:

None

13) CustomFileGetSize

Triggered when to call OpenCustomFile to open PDF.

Prototype:

Void CustomFileGetSize (long* size)

Parameters:

Size:	A pointer to a value specifying the size of the PDF.

14) CustomFileGetBlock

Triggered when to call OpenCustomFile to open PDF.

Prototype:

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

Parameters:

Pos:	Byte offset from beginning of the PDF.
pBuf:	A pointer to the buffer which will receive the pdf data.
Size:	The size of the buffer.

Note:

Get a block of data from the position specified by byte offset, which is from the beginning of a PDF. The offset and buffer size must be in the PDF.

15) OnClick

Triggered when to click the left button.

Prototype:

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

Parameters:

hWnd:	The handle of a ActiveX window.
ClientX :	X coordinate in the client area of ActiveX window.
ClientY :	Y coordinate in the client area of ActiveX window.

16) OnDbClick

Triggered when to double click the left button.

Prototype:

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

Parameters:

hWnd:	The handle of a ActiveX window.
ClientX :	X coordinate in the client area of ActiveX window.
ClientY:	Y coordinate in the client area of ActiveX window.

17) OnRButtonClick

Triggered when to click the right mouse.

Prototype:

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

Parameters:

hWnd:	The handle of a ActiveX window.
ClientX :	X coordinate in the client area of ActiveX window.
ClientY :	Y coordinate in the client area of ActiveX window.

18) OnDownLoadFinish

Triggered once PDF downloading from Internet is completed.

Prototype:

Void OnDownLoadFinish();

Parameters:

None

Note:

It can work with OpenFile/OpenFileAsync/OpenAsyncFile when to access PDF from Internet.

19) OnErrorOccurred

Triggered once there is any error when to call the ActiveX interfaces. It only supports GetToolByIndex and ShowToolbarButton.

Prototype:

Void OnErrorOccurred(BSTR lpszErrorMsg)

Parameters:

lpszErrorMsg:	Error message.

Return Value:

None

20) OnUploadFinish

Triggered once there exist errors when to call UploadCurFileToFTP.

Prototype:

Void OnUploadFinish(short nRetCode)

Parameters:

nRetCode:	The error code.

Return Value:

None

21) OnTextHyperLink

Triggered when to click the text link.

Prototype:

Void OnTextHyperLink(BSTR csUrl, boolean* cancel)

Parameters:

csUrl:	The URL of the text link.
cancel:	A flag specifying whether to cancel executing the link's action.

Return Value:

None

22) OnExcuteMenuItem

Triggered when to execute an action of the customized menu item that is added by OnAddmenuItemAction.

Prototype:

Void OnExcuteMenuItem(BSTR sMenuItem, boolean* bResult)

Parameters:

sMenuItem:	The custom menu item.
bResult:	A flag specifying whether to get the result of the action.

Return Value:

None

23) *OnAddMenuItemAction

Triggered when to add an action of “add a menu item”.

Prototype:

void OnAddMenuItemAction(BSTR* pMenuItem)

Parameters:

pMenuItem:	A menu item.

Return Value:

None

24) OnDoGoToRAction

Triggered when to execute the GoToR action.

Prototype:

void OnDoGoToRAction(BSTR sFilePath, Link_Dest* dest)

Parameters:

sFilePath:	File path of a PDF the GoToR action will go to.
dest:	The destination of the GoToR action.

Return Value:

None

25) OnDoGoToEAction

Triggered when to open an annotation link that points to the PDF that is attached current file.

Prototype:

Void OnDoGoToEAction(BSTR sFilePath, Link_Dest* dest) ;

Parameters:

sFilePath:	The temp path to place the attached PDF.
dest:	The destination of the annotation link.

Return Value:

None

26) *OnHyperLink

Triggered when to click on a text hyperlink.

Prototype:

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

Parameters:

Linktype: The hyperlink type. It can be set to:

GoTo - goes to a different page on the current PDF. For this, linkdata parameter must be set to NULL and dest parameter must be set to the destination position where ActiveX should go to.

GoToR - goes to a different local PDF. If it requires a new window to view the document, linkdata parameter should contain the filename followed by 1. Otherwise, the filename should be followed by 0. Dest parameter must be set to the destination position where ActiveX should go to.

Launch - launches an external application. If it requires a new window to view the document, linkdata parameter should contain the filename followed by 1. Otherwise, the filename should be followed by 0.

URI - opens an URI. For this, linkdata parameter must be set to include URI information.

linkData:	The message for the hyperlink.
dest:	The destination position where ActiveX should go to.
Cancel:	A flag specifying whether to disable the hyperlink.

Return Value:

None

27) *OnContextMenuIndex

Triggered when to choose a context menu item by right-clicking.

Prototype:

Void OnContextMenuIndex(short nIndex);

Parameters:

nIndex:	The index of a context menu item.

Return Value:

None

Note:

This can be used together with SetContextMenuString.

28) *OnFetchAsyncFileData

Triggered when to open an asynchronous file. This will be used to request for the file data.

Prototype:

void OnFetchAsyncFileData(long offset, long size)

Parameters:

offset:	The offset of the requested PDF data.
size:	The size of the requested PDF data.

Return Value:

None

Note:

*OnFetchAsyncFileData should be used together with *SetAsyncFileData, *OpenAsyncFile and *SetAsyncFileLen.

29) *OnCurPageIndexChanged

Triggered once the current PDF page has been changed.

Prototype:

VoidOnCurPageIndexChanged (long curPageIdx, long newPageIdx)

Parameters:

curPageIdx:	The old index of current PDF page.
newPageIdx:	The new index of current PDF page.

Return Value:

None

30) *OnSigContextMenuIndex

Triggered when to click a context menu of a signature filed.

Prototype:

void OnSigContextMenuIndex(short nIndex, IDispatch* SignatureField)

Parameters:

nIndex:	The index of a context menu.
SignatureField:	A pointer to a signature filed.

Return Value:

None

31) OnPagesContextMenuIndex

Triggered when to click a context menu of a page thumbnail in Page panel.

Prototype:

void OnPagesContextMenuIndex(short nIndex,VARIANT* pageArray)

Parameters:

nIndex:	The index of a page context menu.
pageArray:	The page array.

Return Value:

NULL

32) OnBookmarkContextMenuIndex

Triggered when to click a context menu of a bookmark.

Prototype:

void OnBookmarkContextMenuIndex(short nIndex)

Parameters:

nIndex:	The index of a bookmark context menu to be clicked.

Return Value:

NULL

33) *OnFormFieldClick

Triggered when to click a form field.

Prototype:

void OnFormFieldClick(PDFFormField* pClickedField)

Parameters:

nIndex:	A pointer to a PDF form field.

Return Value:

NULL

34) *OnFormFieldKeyDown

Triggered when to press key down to a form field.

Prototype:

void OnFormFieldKeyDown(IPDFFormField* pFormField, long* nKey)

Parameters:

pFormField:	A pointer to a PDF form field object.
nKey:	Virtual-key code.

Return Value:

NULL

35) *OnFormFieldKeyUp

Triggered when to release the key (key up) to a form field.

Prototype:

void OnFormFieldKeyUp(PDFFormField* pFormField, long* nKey)

Parameters:

pFormField:	A pointer to a PDF form field object.
nKey:	Virtual-key code.

Return Value:

NULL

36) *OnSetFocus

Triggered when to set focus to an ActiveX instance.

Prototype:

void OnSetFocus(long hwnd)

Parameters:

hwnd:	The handle of an ActiveX instance.

Return Value:

NULL

37) *OnKillFocus

Triggered when to lose focus from a control instance.

Prototype:

void OnKillFocus(long hwnd)

Parameters:

hwnd:	The handle of an ActiveX instance.

Return Value:

NULL

38) *OnDbClickEx

Triggered when to double click the left mouse button.

Prototype:

void OnDbClickEx(long hWnd, long ClientX, long ClientY, boolean* bRet)

Parameters:

hWnd:	The handle of a current ActiveX instance.
ClientX:	X coordinate in the client area of ActiveX window.
ClinetY:	Y coordinate in the client area of ActiveX window.
bRet:	A value indicating whether to go with the default ActiveX workflow or a custom workflow.

TRUE	-	Use a custom workflow.
FALSE	-	Use a default ActiveX workflow.

Return Value:

NULL

39) *OnRButtonClickEx

Event triggered when to click the right mouse button.

Prototype:

void OnRButtonClickEx(long hWnd, long ClientX, long ClientY, boolean* bRet)

Parameters:

hWnd:	The handle of the ActiveX control instance.
ClientX :	X coordinate in the client area of ActiveX window.
ClientY :	Y coordinate in the client area of ActiveX window.
bRet:	A value indicating whether to go with the default ActiveX workflow or a custom

workflow.

TRUE	-	Use a custom workflow.
FALSE	-	Use a default ActiveX workflow.

Return Value:

NULL

40) #FormFieldError

Triggered when an error occurs for configuring a PDF form field.

Prototype:

Void FormFieldError(long nErrorCode);

Parameters:

nErrorCode:	The error code of configuring the PDF form field.

Return Value:

None

IPDFPrinter

IPDFPrinter helps to choose the printer and do settings for PDF printing.

Properties

1) PrinterName

Type:

BSTR

Description:

The name of a printer for PDF printing.

2) PrinterRangeMode

Type:

PrinterRangeMode

Description:

Set the printing range. It 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:

The first PDF page to be printed. You must first set PrinterRangeMode to PRINT_RANGE_SELECTED.

4) PrinterRangeTo

Type:

short

Description:

The last PDF page to be printed. You must first set PrinterRangeMode to PRINT_RANGE_SELECTED.

5) NumOfCopies

Type:

short

Description:

Print copies.

6) Scaling

Type:

short

Description:

Scaling for printing.

Note:

ActiveX will print with “Fit to Paper” when setting Scaling to 0.

7) AutoRotate

Type:

boolean

Description:

A flag indicating whether to check Auto-Rotate option for printing or not.

8) AutoCenter

Type:

boolean

Description:

A flag indicating whether to check Auto-center option for printing or not.

1 = auto-center

0 = not auto-center

9) Collate

Type:

boolean

Description:

A flag indicating whether to check Collate option for printing or not.

1 = Collate

0 = not Collate

10) Rotation

Type:

short

Description:

Page rotation for printing.

11) RangeSubset

Type:

short

Description:

A value indicating whether to include subset for printing or not.

12) ReversePage

Type:

boolean

Description:

A flag indicating whether to print the PDF in reverse order or not.

13) PageBorder

Type:

boolean

Description:

A flag indicating whether to print out page borders.

14) PrintWhat

Type:

Short

Description:

PDF contents to be printed out. It can be set to:

0	-	Document
1	-	Document and Annotations
2	-	Annotations

Methods

1) PrintWithDialog

Pop up print dialog when to print a PDF.

Prototype:

void PrintWithDialog()

Parameters:

None

Return Value:

None

2) PrintQuiet

Print a PDF without setting in the print dialog.

Prototype:

Void PrintQuiet()

Parameters:

None

Return Value:

None

3) SetPaperSize

Set paper size of a selected printer for printing.

Prototype:

Void SetPaperSize (long paperSize)

Parameters:

paperSize:	Paper size for printing.

Return Value:

None

Note:

Please refer to Windows SDK documentation to get the information about the available paper sizes.

4) GetSystemPrinterCount

Count the system printers.

Prototype:

LongGetSystemPrinterCount()

Return Value:

Return the total number of the system printers.

5) GetSystemPrinterNameByIndex

Get the printer name in the system by index.

Prototype:

BSTR GetSystemPrinterNameByIndex(long index)

Return Value:

Return the name of a system printer by index.

6) SetPaperSizeByPage

Set to check “Choose Paper Source by PDF Page Size” option for PDF printing.

Prototype:

void SetPaperSizeByPage(BOOL bPage);

Parameters:

bPage:	A flag specifying whether to check the option.

Return Value:

None

7) SetDuplexMode

Set duplex mode.

Prototype:

Boolean SetDuplexMode(short nDuplexMode)

Parameters:

short nDuplexMode: A value specifying the duplex mode. Possible values are 1, 2 or 3.

1	=	DMDUP_SIMPLEX (Simplex (non duplex) mode.)
2	=	DMDUP_VERTICAL (Duplex mode that flips the page vertically.)
3	=	DMDUP_HORIZONTAL (Duplex mode that flips the page horizontally.)

Return Value:

Return TRUE if successful, otherwise FALSE.

IPDFDocumentInfo

IPDFDocumentInfo helps to get the properties information of a PDF.

Properties

1) Author

Type:

BSTR

Description:

The author of a PDF.

2) Subject

Type:

BSTR

Description:

The subject of a PDF.

3) CreatedDate

Type:

BSTR

Description:

The date when the PDF was created.

4) ModifiedDate

Type:

BSTR

Description:

The last time when the PDF was modified.

5) Keywords

Type:

BSTR

Description:

The keywords of a PDF.

6) Creator

Type:

BSTR

Description:

The creator of a PDF.

7) Producer

Type:

BSTR

Description:

The producer of a PDF.

8) Title

Type:

BSTR

Description:

The title of a PDF.

IFindResult

IFindResult helps to get the information about a text search result.

Method

1) GetFindRectsCount

Count the search result rectangles.

Prototype:

long GetFindRectsCount()

Return Value:

Return the total number of the search result rectangles if successful, otherwise 0.

2) GetFindRectByIndex

Get a specific search result rectangle.

Prototype:

long GetFindRectByIndex(long nIndex,float* left, float* bottom, float* right, float* top)

Parameters:

nIndex:	The index of a search result rectangle.
left:	[out]Left coordinate of the search result rectangle.
bottom:	[out]Bottom coordinate of the search result rectangle.
right:	[out]Right coordinate of the search result rectangle.
top:	[out]Top coordinate of the search result rectangle.

Return Value:

Return TRUE if successful, otherwise FALSE.

3) GetFindPageNum

Get the index of the page where the search result is.

Prototype:

long GetFindPageNum();

Parameters:

None

Return Value:

Return the index of the page where the search result is.

4) GetFindFileName

Get the name of the PDF where the search result is.

Prototype:

BSTR GetFindFileName();

Parameters:

None

Return Value:

Return the name of the PDF where the search result is.

5) GetFindString

Get the content of the search result.

Prototype:

BSTR GetFindString()

Parameters:

None

Return Value:

Return the content of the search result.

IPDFOutline

IPDFOutline helps get the information of an outline.

Method

1) GetOutlineDest

Get outline link destination.

Prototype:

ILink_Dest* GetOutlineDest()

Return Value:

Return a pointer to an ILink_Dest .

2) GetOutlineAction

Get outline associated action.

Prototype:

IPDF_Action* GetOutlineAction()

Return Value:

Return a pointer to an IPDF_Action.

3) GetOutlineColor

Get outline text color (RGB).

Prototype:

DWORD GetOutlineColor()

Return Value:

Return the text color of the outline.

4) NavigateOutline

Navigate to the destination specified by the outline.

Prototype:

void NavigateOutline()

Parameters:

None

Return Value:

None

5) GetOutlineTitle

Get the title of the outline.

Prototype:

BSTR GetOutlineTitle()

Parameters:

None

Return Value:

Return the title of the outline.

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

Count the child items of an outline.

Prototype:

Long GetOutlineExpandValue()

Return Value:

Return the total number of the child items.

ILink_Dest

ILink_Dest helps to get the information of a hyperlink.

Method

1) GetPageIndex

Prototype:

Long GetPageIndex()

Parameter:

None.

Return Value:

Return the index of the PDF page which the hyperlink connects to.

2) GetZoomMode

Get the zoom mode.

Prototype:

Long GetZoomMode();

Parameter:

None.

Return Value:

Return a value indicating the zoom mode. Possible values are:

2	-	Fit Page
3	-	Fit Width
4	-	Fit Height

3) GetZoomParamCount

Count Zoom parameters.

Prototype:

Long GetZoomParamCount()

Parameter:

None.

Return Value:

Return the total number of Zoom parameters.

4) GetZoomParam

Get the information of a specific Zoom parameter.

Prototype:

```
double GetZoomParam(long nIndex)
```

Parameter:

nIndex:	The index of a Zoom parameter.

Return Value:

Return the information of the specified Zoom parameter.

5) GetDestName

Get the name of a Name Destination object.

Prototype:

```
BSTR GetDestName()
```

Parameter:

None.

Return Value:

Return the name of a Name Destination object.

IPDFAction

IPDFAction helps to get the information of an action.

Method

1) GetURIPath

Get the path of URI related to the action.

Prototype:

BSTR GetURIPath();

Return Value:

Return the path of URI.

2) GetFilePath

Get the file path of the external files related to the action.

Prototype:

BSTR GetFilePath();

Return Value:

Return the file path of the external files related to the action.

3) GetType

Get the type of the action.

Prototype:

BSTR GetType()

Return Value:

Return a value indicating the action type. Possible values are:

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

4) GetDest

Get the destination of the action.

Prototype:

Ilink_Dest* GetDest();

Return Value:

Return a pointer to an Ilink_Dest object.

#IPDFForm

IPDFForm helps to manage and operate the form fields.

Method

1) #ImportFromFDF

Import form data from an FDF.

Prototype:

void ImportFromFDF(BSTR bstrFullPath)

Parameters:

bstrFullPath:	File path of a FDF from which the form data will be imported.

Return Value:

None

2) #ExportToFDF

Export form data to a FDF.

Prototype:

void ExportToFDF(BSTR bstrFullPath, boolean bEmptyFields, VARIANT arrFields)

Parameters:

bstrFullPath:		Full path of a FDF to which the form fields will be exported.
bEmptyFields:		A value specifying whether to only export the specified form fields.
TRUE	-	Only export the form fields specified by arrFields.
FALSE	-	Export all the other form fields but the ones specified by arrFields.
ArrFields:		The array of form fields to be exported.

Return Value:

None

3) #AddField

Add a new form field.

Prototype:

IPDFFormField* AddField(BSTR bstrFieldName, BSTR bstrFieldType, long pageIndex, float left, float top, float right, float bottom)

Parameters:

bstrFieldName:	The fully qualified name for the form field.

bstrFieldType:	The type of the form field to be created. It can be set to:
	text,
	button,
	combobox,
	listbox,
	checkbox,
	radio button
pageIndex:	The index of a PDF page.
Left:	The left coordinate of the form field rectangle.
top:	The top coordinate of the form field rectangle.
Right:	The right coordinate of the form field rectangle.
bottom:	The bottom coordinate of the form field rectangle.

Return Value:

Return an IPDFFormField object for the newly created form field.

Note:

The coordinate of the field rectangle is based on rotated page space. So, [0,0] is always at the left bottom corner regardless of page rotation.

4) #GetSelectedField

Get the selected form field.

Prototype:

IPDFFormField* GetSelectedField()

Parameters:

None

Return Value:

Return an IPDFFormField object for the selected form field.

5) #RemoveField

Remove a specific form field.

Prototype:

Void RemoveField(IPDFFormField pFormField)

Parameters:

pFormField:	A pointer to a form field object to be removed.

Return Value:

None

6) #RemoveFieldsByName

Delete a specific form field.

Prototype:

Void RemoveFieldsByName(LPCTSTR bstrFieldName)

Parameters:

bstrFieldName:	The fully qualified name of a form field to be deleted.

Return Value:

None

Note:

If the specified form field has child fields or multiple fields share a same name, all of the form fields will be deleted.

7) #GetFieldsCount

Count the form fields in PDF.

Prototype:

Long GetFieldsCount()

Parameters:

None

Return Value:

Return the total number of the form fields if successful, otherwise -1.

8) #GetFieldByIndex

Get a specific form field.

Prototype:

IPDFFormField* GetFieldByIndex(long index)

Parameters:

index:	The index of a form field.

Return Value:

Return an IPDFFormField object for the specified form field if successful, otherwise NULL.

9) #GetSelectedField

Select a form field.

Prototype:

IPDFFormField* GetSelectedField()

Parameters:

None

Return Value:

Return an IPDFFormField object

#IPDFFormField

IPDFFormField helps to access and operate a form field.

Properties

1) #Alignment

Type:

String

Description:

Horizontal alignment of the text in text field. Possible values are: left, center and right.

Note:

For Text Field only.

2) #BorderStyle

Type:

String

Description:

The border style of a form filed. Possible values are: solid, dashed, beveled, inset and underline.

Note:

For all form field types.

3) #BorderWidth

Type:

short

Description:

The border width of a form filed.

Note:

For all form field types.

4) #ButtonLayout

Type:

short

Description:

The layout of a Push Button. Possible values are:

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:

The index of a form field in CO array.

Note:

For all form types.

6) #CharLimit

Type:

short

Description:

The character number limit of a text field.

Note:

For Text Field Only

7) #DefaultValue

Type:

String

Description:

Default value of a form field.

Note:

For all form types.

8) #IsEditable

Type:

Boolean

Description:

A flag specifying whether a combo box is editable.

Note:

For Combo Box Only.

9) #Behavior

Type:

String

Description:

Possible values are: None, Invert, Outline and 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:

A flag specifying whether a form field is hidden.

Note:

For all form types.

11) #IsMultiline

Type:

Boolean

Description:

A flag specifying whether a text field is in multi-line or single-line.

Note:

For Text Field only.

12) #IsPassword

Type:

Boolean

Description:

A flag specifying whether to show the input as password.

Note:

For Text Field only.

13) #IsReadOnly

Type:

Boolean

Description:

A flag specifying whether to set a form field as read only.

Note:

For all form types.

14) #IsRequired

Type:

Boolean

Description:

A flag specifying whether a form field is required.

Note:

For Combo Box, Radio Button and Text Field.

15) #Name

Type:

String

Description:

The name of a form field.

Note:

Read only. For all form types.

16) #NoViewFlag

Type:

Boolean

Description:

A flag specifying whether to hide a form field.

0	-	Show.
1	-	Hide.

Note:

For all form types.

17) #PrintFlag

Type:

Boolean

Description:

A flag specifying whether to print the form field.

1	-	Print out.
0	-	Not to print out.

Note:

For all form types.

18) #Style

Type:

CString

Description:

A value indicating the shape of Check Box and Radio Button. Possible Values are:

check

cross

diamond

circle

star

square

Note:

For Check Box and Radio Button.

19) #TextFont

Type:

String

Description:

A font name to be used to display the text in a form field. Possible Values are:

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 Check Box and Radio Box.

20) #TextSize

Type:

Short

Description:

The font size of the text in a form field.

Note:

For all form types except for Check box and Radio Box.

21) #Type

Type:

String

Description:

The form type. Possible values are:

Text

Button

Combobox

Listbox

Checkbox

radiobutton

22) #Value

Type:

String

Description:

The value of a form field.

Note:

For Text Field, Combo Box, Radio Button, Check Box and List Box.

23) #Tooltip

Type:

String

Description:

A value indicating whether to display the tooltip.

Note:

For all form types.

24) #Orientation

Type:

Short

Description:

The text rotation of the form fields.

Note:

For all form types.

25) #DirtyFlag

Type:

boolean

Description:

A value indicating whether the form field has been changed.

26) #ID

Type:

BSTR

Description:

The ID of a form field.

Method

1) #PopulateListOrComboBox

Set values for List Box or Combo Box.

Prototype:

void PopulateListOrComboBox(VARIANT arrItems, VARIANT arrExportVal)

Parameter:

arrItem:	An array of strings, each of which represents an item name.
arrExportVal:	An array of strings, each of which represents an item name. The size of arrExportVal is the same as arrItem.

Return Value:

None

2) #SetBackgroundColor

Set the background color of a form field.

Prototype:

Void SetBackgroundColor(LPCTSTR bstrColorSpace, float redC, float greenM, float blueY, float AlphaK)

Parameter:

bstrColorSpace:	The color space. Possible values are: Transparent, Gray, RGB, CMYK.
Transparent:	redC parameter is required.
Gray:	redC parameter is required.
RGB:	redC, greenM and blueY are required.
CMYK:	redC, greenM, blueY and AlphaK are required.
redC:	A value which should be from 0 to 1.
greenM:	A value which should be from 0 to 1.
blueY:	A value which should be from 0 to 1.
AlphaK:	This parameter is required by CMYK.

Return Value:

None

3) #SetBorderColor

Set the border color of a form field.

Prototype:

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

Parameter:

bstrColorSpace:	The color space. Possible values are: Transparent, Gray, RGB, CMYK.
Transparent:	redC parameter is required.
Gray:	redC parameter is required.

RGB:	redC, greenM and blueY are required.
CMYK:	redC, greenM, blueY and AlphaK are required.
redC:	A value which should be from 0 to 1.
greenM:	A value which should be from 0 to 1.
blueY:	A value which should be from 0 to 1.
AlphaK:	This parameter is required by CMYK.

Return Value:

None

4) #SetForegroundColor

Set the foreground color.

Prototype:

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

Parameter:

bstrColorSpace:	The color space. Possible values are: Transparent, Gray, RGB, CMYK.
Transparent:	redC parameter is required.
Gray:	redC parameter is required.
RGB:	redC, greenM and blueY are required.
CMYK:	redC, greenM, blueY and AlphaK are required.
redC:	A value which should be from 0 to 1.
greenM:	A value which should be from 0 to 1.
blueY:	A value which should be from 0 to 1.
AlphaK:	This parameter is required by CMYK.

Return Value:

None

5) #SetButtonCaption

Set the text to be displayed.

Prototype:

Void SetButtonCaption (LPCTSTR bstrFace, LPCTSTR bstrCaption)

Parameter:

bstrFace: A string specifying the appearance . Possible values are:

N	-	Normal appearance
D	-	Down appearance

R - Appearance for rollover

bstrCaption: The text information of a button.

Return Value:

None

6) #SetButtonIcon

Set the icon for a button field.

Prototype:

Void SetButtonIcon (LPCTSTR bstrFace, LPCTSTR bstrFilePath)

Parameters:

bstrFace: A string specifying the caption of a button field. Possible values are:

N - Normal appearance

D - Down appearance

R - Appearance for rollover

BstrFilePath: File path of an image to be used as the icon of a button field.

Return Value:

None

7) #SetExportValues

Set the export values for Radio Button or Check Box according to the requirements (Selected, un-selected, checked, un-checked, etc)

Prototype:

Void SetExportValues (const VARIANT& arrExportVal)

Parameters:

arrExportVal: An array of export values.

Return Value:

None

8) #SetJavaScriptAction

Set JavaScript action.

Ptototype:

Void SetJavaScriptAction (LPCTSTR bstrTrigger, LPCTSTRbstrJavaScript)

Parameters:

BstrTrigger: A string specifying the trigger for the JavaScript. Possible values are: up, down, enter, exit, calculate, validate, format and keystroke.

bstrJavaScript: The JavaScript script to be executed.

Return Value:

None

9) #SetResetFormAction

Set reset-form action for a form field.

Prototype:

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

Parameters:

bstrTrigger : A string specifying the trigger for the reset-form action. Possible values are:

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 :	An array of form elements to be exported.

Return Value:

None

10) #SetSubmitFormAction

Set submit-form action for a form field.

Prototype:

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

Parameters:

bstrTrigger: A string specifying the trigger for the submit-form action. Possible values are:

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

Get the index of the page where the form field is.

Prototype:

LongGetPageIndex()

Parameters:

None

Return Value:

Return the index of a PDF page.

12) #GetRectTop

Get the top coordinate of a form field.

Prototype:

Float GetRectTop(l)

Parameters:

None

Return Value:

Return a top coordinate.

13) #GetRectLeft

Get the left coordinate of a form field.

Prototype:

Float GetRectLeft()

Parameters:

None

Return Value:

Return a left coordinate.

14) #GetRectRight

Get the right coordinate of a form field.

Prototype:

Float GetRectRight()

Parameters:

None

Return Value:

Return a right coordinate.

15) #GetRectBottom

Get the bottom coordinate of a form field.

Prototype:

Float GetRectBottom()

Parameters:

None

Return Value:

Return a bottom coordinate.

^IPDFPageAnnots

IPDFPageAnnots helps to access and operate the annotations in a PDF page.

Method

1) ^GetAnnot

Get a specific annotation in PDF page.

Prototype:

CPDFAnnot GetAnnot(long AnnotIndex)

Parameters:

AnnotIndex :	The index of a PDF annotation.

Return Value:

Return a PDF annotation for the specified annotation if successful, otherwise NULL.

2) ^GetLTAnnot

Get a specific annotation in PDF page.

Prototype:

IPDFAnnot*GetLTAnnot(long AnnotIndex)

Parameters:

AnnotIndex :	The index of a PDF annotation.

Return Value:

Return a PDF annotation for the specified annotation if successful, otherwise NULL.

Note:

The PDF annotation object return by GetLTAnnot should be released by ReleaseLTAnnot.

3) ^ReleaseLTAnnot

Release an annotation object returned by GetLTAnnot.

Prototype:

Void ReleaseLTAnnot(IPDFAnnot* Annot)

Parameters:

AnnotIndex :	The index of a PDF annotation.

Return Value:

Return a PDF annotation for the specified annotation if successful, otherwise NULL.

4) ^AddAnnot

Add a new PDF annotation.

Prototype:

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

Parameters:

AnnotToAddAfter :	Reserved. The value must be NULL.
SubType :	Annotation type. It can be set to:

"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 coordinate of the annotation rectangle.
Top :	The top coordinate of the annotation rectangle.
Right :	The right coordinate of the annotation rectangle.
Bottom :	The bottom coordinate of the annotation rectangle.

Return Value:

Return an IPDFAnnot object for the new annotation if successful, otherwise NULL.

Note:

All coordinates are based on the PDF coordinate space. To add a stamp annotation, SetStampParam must be called before using AddAnnot.

5) ^AddLTAnnot

Add a new PDF annotation.

Prototype:

IPDFAnnot*AddLTAnnot(IPDFAnnot*AnnotToAddAfter, BSTR SubType, float Left, float Top, float Right, float Bottom)

Parameters:

AnnotToAddAfter:	Reserved. The value must be NULL.
SubType:	Annotation type. It can be set to:

"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 coordinate of the annotation rectangle.
Top:	The top coordinate of the annotation rectangle.

Right:	The right coordinate of the annotation rectangle.
Bottom:	The bottom coordinate of the annotation rectangle.

Return Value:

Return an IPDFAnnot object for the new annotation if successful, otherwise NULL.

Note:

AddLTAnnot will return a long-term annotation object, which should be released by ReleaseLTAnnot. It helps developer better control an annotation object in some cases.

6) ^ SetStampParam

Set for a stamp object which will be added by AddAnnot.

Prototype:

Void SetStampParam(long source, long length, long nSourceType, LPCTSTR flag, short pageIndex)

Parameters:

source:	A pointer to a buffer specifying the source file for the stamp object.
length:	The buffer size.
nSourceType:	The type of the source file for the stamp object. 0 means using PDF source file and non-zero means using image source file.
LPCTSTR flag:	The name of an icon to be used in displaying the stamp object.
short pageIndex:	The index of the page in the source file.

Return Value:

None

Note:

SetStampParam must be called before calling AddAnnot to create stamp.

7) ^RemoveAnnot

Delete a specific PDF annotation.

Prototype:

Long RemoveAnnot(LPDISPATCH AnnotToRemove)

Parameters:

AnnotToRemove :	A pointer to a PDF annotation.

Return Value:

Return 0 if successful, otherwise -1.

8) ^GetAnnotIndex

Get the index of a specific PDF annotation.

Prototype:

Long GetAnnotIndex(LPDISPATCH Annot)

Parameters:

Annot :	A pointer to a PDF annotation.

Return Value:

Return the index of the specified PDF annotation if successful, otherwise -1.

9) ^GetAnnotsCount

Count the annotations.

Prototype:

long GetAnnotsCount()

Parameters:

None

Return Value:

Return the number of the annotations.

^IPDFAnnot

IPDFAnnot helps to access and operate a specific PDF annotation object.

Properties

1) ^Thickness

Type:

short. Read and Write.

Description:

The border width of an annotation.

Note:

Thickness is for drawing annotations, measure tools and the annotations which have a border, including Cloudy, Arrow, Line, Square, Rectangle, Circle, Ellipse, Polygon, Polyline, Pencil, Callout, Textbox and Link.

2) ^BorderStyle

Type:

Short. Read and Write

Description:

The border style of an annotation.

Note:

An annotation may optionally be surrounded by a border when displayed or printed. If present, the border is drawn completely inside the annotation rectangle.

Possible values are:

1	-	Solid
2	-	Dashed type 1
3	-	Dashed type 2
4	-	Dashed type 3
5	-	Dashed type 4
6	-	Dashed type 5
7	-	Dashed type 6
8	-	Cloudy type 1
9	-	Cloudy type 2

Solid type can be used by the following annotation types:

"Rectangle", "Cloudy" , "Ellipse", "Circle", "Arrow" , "Polygon", "Line", "Square" , "PolyLine", "Callout", "Textbox" ,"Image", "Movie", "Sound", "Rectangle Link", "Quadrilateral Link".

Dashed type can be used by the following annotation types:

"Rectangle", "Cloudy" , "Ellipse", "Circle", "Arrow" , "Polygon", "Line", "Square" , "PolyLine", "Callout", "Textbox".

Cloudy type can be used by the following annotation types:

"Rectangle", "Cloudy", "Ellipse", "Circle", "Polygon", "Square" , "Callout", "Textbox"

3) ^Color

Type:

OLE_COLOR. Read and Write.

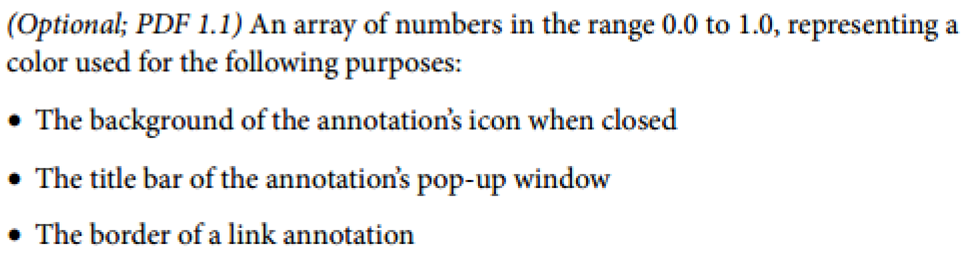
Description:

The background color of an annotation.

Note:

For all annotation types.

The following part is from PDF Reference document. For more details, please refer to the document.



(Optional; PDF 1.1) An array of numbers in the range 0.0 to 1.0, representing a color used for the following purposes:

- The background of the annotation's icon when closed
- The title bar of the annotation's pop-up window
- The border of a link annotation

4) ^LineStartingStyle

Type:

Short. Read and Write.

Description:

The starting style of line.

Note:

For 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:

For 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 a PDF annotation.

Note:

For the following annotation types: Cloudy, Arrow, Line, Square, Rectangle, Circle, Ellipse, Polygon and Polyline.

7) ^Opacity

Type:

Short. Read and Write.

Description:

The opacity of an annotation.

Note:

Can’t support the following annotation types: Link, Movie and Sound.

8) ^Author

Type:

BSTR. Read and Write.

Description:

The author of a PDF annotation.

Note:

Can’t support the following annotation types: Image, Movie, Sound and Link.

9) ^Subject

Type:

BSTR. Read and Write.

Description:

The subject of a PDF annotation.

Note:

Can’t support the following annotation types: Image, Movie, Sound and Link.

10) ^CreationDate

Type:

DATE. Read only.

Description

The date and time when the annotation was created.

11) ^ModificationDate

Type:

DATE. Read only.

Description:

The date and time when the annotation was most recently modified.

12) ^Locked

Type:

Boolean. Read and Write.

Description:

A flag indicating whether an annotation is locked.

Note:

For all the annotation types.

13) ^Print

Type:

Boolean. Read and Write.

Description:

A flag indicating whether to print the annotation when the page is printed.

Note:

For all the annotation types.

14) ^ReadOnly

Type:

Boolean. Read and Write.

Description:

A flag indicating whether an annotation is read only.

Note:

Read Only means that the annotation is not allowed to interact with the user. Then the annotation may be displayed or printed (depending on the settings of the NoView and Print flags) but should not respond to mouse clicks or change its appearance in response to mouse motions.

Note:

For all the annotation types.

15) ^Description

Type:

BSTR. Read and Write.

Description:

The description of a PDF annotation.

Note:

For FileAttachment only.

Methods

1) ^GetType

Get the type of a PDF annotation.

Prototype:

BSTR GetType()

Parameters:

None

Return Value:

Return the type of a PDF annotation.

2) ^GetSubType

Get the subtype of a PDF annotation.

Prototype:

BSTR GetSubType()

Parameters:

None

Return Value:

Return the Subtype of a PDF annotation.

3) ^GetContents

Get the content of a PDF annotation.

Prototype:

BSTR GetContents()

Parameters:

None

Return Value:

Return the contents of a PDF annotation.

4) ^SetContents

Set content for a PDF annotation.

Prototype:

long SetContents(LPCTSTR Contents)

Parameters:

Contents:	The content for a PDF annotation.

Return Value:

Return 0 if successful, otherwise -1

5) ^IsPopupOpen

Check if the popup box is open.

Prototype:

BOOL IsPopupOpened()

Parameters:

None

Return Value:

Return TRUE if the popup box is open, otherwise FALSE.

6) ^SetPopupOpen

Set to open the popup box.

Prototype:

LongSetPopupOpened(BOOL Open)

Parameters:

Open:	A flag specifying whether to open the popup box.

Return Value:

Return 0 if successful, otherwise -1.

7) ^HasPopup

Check if the annotation has a popup box.

Prototype:

BOOL HasPopup()

Parameters:

None

Return Value:

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

8) ^GetRect

Get the annotation rectangle, which defines the location of the annotation.

Prototype:

LongGetRect(float* pLeft, float* pTop, float* pRight, float* pBottom)

Parameters:

pLeft:	[out]The left coordinate of the annotation rectangle.

pTop:	[out]The top coordinate of the annotation rectangle.
pRight:	[out]The right coordinate of the annotation rectangle.
pBottom:	[out]The bottom coordinate of the annotation rectangle.

Return Value:

Return 0 if successful, otherwise -1.

9) ^SetRect

Set the rectangle for a PDF annotation.

Prototype:

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

Parameters:

Left:	The left coordinate of the annotation rectangle.
Top:	The top coordinate of the annotation rectangle.
Right:	The right coordinate of the annotation rectangle.
Bottom:	The bottom coordinate of the annotation rectangle.

Return Value:

Return 0 if successful, otherwise -1.

10) ^SetLinkGoToAction

Set a link annotation with GoTo action.

Prototype:

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

Parameters

nPageIndex:	The index of a PDF page.
left:	The left coordinate.
top:	The top coordinate.
zoom:	The zoom factor for the page viewing.

Return Value:

None

11) ^SetLinkURLAction

Set a link annotation with URL action.

Prototype:

Void SetLinkURLAction(LPCTSTR sURL)

Parameters:

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

Return Value:

None

12) ^DoAction

Execute the action of a link annotation.

Prototype:

long DoAction()

Parameters:

None

Return Value:

Return 0 if successful, otherwise -1.

13) ^HasAction

Check if a link annotation contains an action.

Prototype:

BOOL HasAction()

Parameters:

None

Return Value:

Return TRUE if the link annotation contains an action, otherwise FALSE.

14) ^GetMarkedState

Get the mark state of a PDF annotation.

Prototype:

Long GetMarkedState()

Parameters

None

Return Value

Return a value specifying the mark state.

0	-	unmarked
1	-	marked
-1	-	error.

15) ^SetMarkedState

Set mark state of the annotation.

Prototype:

long SetMarkedState(long state)

Parameters:

state: Set the mark state of the annotation. It can be set to:

0	-	unmarked
1	-	marked

Return Value:

Return 0 if successful, otherwise -1.

Note:

Can’t support the following annotation types: Image, Movie, Sound and Link.

16) ^GetReviewState

Get the review state of a PDF annotation.

Prototype:

long GetReviewState()

Parameters:

None

Return Value:

Return a value indicating the review state, otherwise -1 indicating an error. Possible values are:

0	-	NULL
1	-	Accepted
2	-	Rejected
3	-	Canceled
4	-	Completed

17) ^SetReviewState

Set review state for a PDF annotation.

Prototype:

long SetReviewState(long state)

Parameters:

State: A value specifying the review state of a PDF annotation. It can be set to:

0	-	NULL
1	-	Accepted
2	-	Rejected
3	-	Canceled
4	-	Completed

Return Value:

Return 0 if successful, otherwise -1.

Note:

Can’t support the following annotation types: Image, Movie, Sound, and Link.

18) ^GetMigrationState

Get the migration state of a PDF annotation.

Prototype:

long GetMigrationState()

Parameters:

None

Return Value:

Return a value indicating the migration state of a PDF annotation. Possible values are:

0	-	NULL
1	-	Not Confirmed
2	-	Confirmed
-1	-	Others.

19) ^SetMigrationState

Set the migration state for a PDF annotation.

Prototype:

long SetMigrationState(long state)

Parameters:

state: A value specifying the migration state of a PDF annotation. It can be set to:

0	-	NULL
1	-	Not Confirmed
2	-	Confirmed.

Return Value:

Return 0 if successful, otherwise -1.

Note:

Can’t support the following annotation types: Image, Movie, Sound and Link.

20) ^GetStartingPoint

Get the start point of the annotation.

Prototype:

long GetStartingPoint(float* PointX, float* PointY)

Parameters:

PointX:	[out]X coordinate of the start point.
PointY:	[out]Y coordinate of the start point.

Return Value:

Return 0 if successful, otherwise -1.

21) ^SetStartingPoint

Set the start point for a PDF annotation.

Prototype:

long SetStartingPoint(float PointX, float PointY)

Parameters:

PointX:	X coordinate of the start point.
PointY:	Y coordinate of the start point.

Return Value:

Return 0 if succeed, otherwise -1.

Note:

For Line and Arrow only.

22) ^GetEndingPoint

Get the end point for a PDF annotation.

Prototype:

Long GetEndingPoint(float* PointX, float* PointY)

Parameters:

PointX:	[out]X coordinate of the end point.
PointY:	[out]Y coordinate of the end point.

Return Value:

Return 0 if succeed, otherwise -1.

Note:

For Line and Arrow only.

23) ^SetEndingPoint

Set the end point for a PDF annotation.

Prototype:

long SetEndingPoint(float PointX, float PointY)

Parameters:

PointX:	X coordinate of the end point.

PointY:	Y coordinate of the end point.

Return Value:

Return 0 if successful, otherwise -1.

Note:

For Line and Arrow only.

24) ^SetMediaPoster

Set a poster (image) for annotations that support posters, including Movie and Sound.

Prototype:

long SetMediaPoster(LPCTSTR ImageFilePath)

Parameters:

ImageFilePath:	File path of an image.

Return Value:

Return 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:	File path of a media.
ContentType:	The MIME type of the media data.
Embed :	A value specifying whether to embed media to the PDF.
bShowCtrlBar:	A value specifying whether to show the control bar. .

Return Value:

Return 0 if successful, otherwise -1.

26) ^SetLinkQuadPoints

Set the appearance for Link annotation.

Prototype:

long SetLinkQuadPoints(long *PointsArray, long PointsCount)

Parameters:

PointsArray:	An array of points.
PointsCount:	The point number of the array. It must be 4.

Return Value:

Return 0 if successful, otherwise -1.

27) ^SetPolygonVertices

Set the appearance for Polygon annotation.

Prototype:

long SetPolygonVertices(long* PointsArray, long PointsCount)

Parameters:

PointsArray:	An array of points.
PointsCount:	The point number of the array.

Return Value:

Return 0 if successful, otherwise -1.

28) ^SetPencilVertices

Set the appearance for Pencil annotation.

Prototype:

long SetPencilVertices(long* PointsArray, long PointsCount)

Parameters:

PointsArray:	An array of points.
PointCount:	The point number of the array.

Return Value:

Return 0 if successful, otherwise -1.

29) ^AttachFile

Set attachment for FileAttachment annotation.

Prototype:

Long AttachFile(LPCTSTR FileName)

Parameters:

FileName:	File path of an attachment for FileAttachment annotation.

Return Value:

Return 0 if successful, otherwise -1.

30) ^ GetReplyList

Get the replies of a PDF annotation.

Prototype:

IPDFAnnotReplyList* GetReplyList()

Return Value:

Return a pointer to an annotation’s reply list if successful, otherwise NULL.

31) ^ UpdateAnnotReplies

Update annotation replies.

Prototype:

Void UpdateAnnotReplies(IPDFAnnotReplyList* pReplies)

Parameters:

pReplies:	The annotation's replies to be updated.

Return Value:

None

Note:

Update all the replies for the annotation.

Events

1) ^OnAnnotCreated

Triggered once a PDF annotation has been created. This is completed by a third-party.

Prototype:

void OnAnnotCreated(long pageIndex, long annotIndex)

Parameters:

pageIndex:	The index of a PDF page.
annotIndex:	The index of a PDF annotation.

2) ^OnAnnotDeleted

Triggered once a PDF annotation has been deleted. It is completed by a third-party.

Prototype:

void OnAnnotDeleted(long pageIndex, long annotIndex)

Parameters:

pageIndex:	The index of a PDF page.
annotIndex:	The index of a PDF annotation.

3) ^OnAnnotModified

Triggered once a PDF annotation has been modified. It is completed by a third-party.

Prototype:

void OnAnnotModified(long pageIndex, long annotIndex)

Parameters:

pageIndex:	The index of a PDF page.

annotIndex:	The index of a PDF annotation.

4) ^OnAnnotReplyCreated

Triggered once the reply of a PDF annotation has been created. This is completed by a third-party.

Prototype:

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

Parameters:

pageIndex:	The index of a PDF page.
annotIndex:	The index of a PDF annotation.
pReply :	An annotation reply to be created.

5) ^OnAnnotReplyDeleted

Triggered once the reply of a PDF annotation has been deleted. This is completed by a third-party.

Prototype:

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

Parameters:

pageIndex:	The index of a PDF page.
annotIndex:	The index of a PDF annotation.
pReply:	A pointer to an annotation reply to be deleted.

6) ^OnAnnotReplyModified

Triggered once the reply of a PDF annotation has been modified. This is completed by a third-party.

Prototype:

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

Parameters:

pageIndex:	The index of a PDF page.
annotIndex:	The index of a PDF annotation.
pReply :	A pointer to an annotation reply to be modified.

7) ^OnAnnotRButtonDown

Triggered when to right click a PDF annotation.

Prototype:

voidOnAnnotRButtonDown(IPDFAnnot*Annot, float x, float y, BOOL* bDefault)

Parameters:

Annot:	A pointer to a PDF annotation.
x:	Horizontal coordinate of the annotation (in PDF coordinate).

y:	Vertical coordinate of the annotation (in PDF coordinate).
bDefault:	A flag specifying whether to use the default action for right-clicking.

Return Value:

None

8) ^OnAnnotRButtonUp

Triggered when to release right-clicking a PDF annotation.

Prototype:

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

Parameters:

Annot:	A pointer to a PDF annotation.
x:	X coordinate of the annotation (in PDF coordinate).
y:	Y coordinate of the moved annotation (in PDF coordinate).
bDefault:	A flag specifying whether to use the default action for releasing right-clicking.

Return Value:

None

9) ^OnAnnotLButtonDbClick

Triggered when to double click a PDF annotation.

Prototype:

voidOnAnnotLButtonDbClick(IPDFAnnot* Annot, float x, float y, BOOL* bDefault)

Parameters:

Annot:	A pointer to a PDF annotation.
x:	X coordinate of the annotation (in PDF coordinate).
y:	y coordinate of the moved annotation (in PDF coordinate).
bDefault:	A flag specifying whether to use the default action for double clicking.

Return Value:

None

10) ^OnAnnotLButtonDown

Triggered when to left click a PDF annotation.

Prototype:

void OnAnnotLButtonDown(IPDFAnnot* Annot, float x, float y, boolean* bDefault)

Parameters:

Annot:	A pointer to a PDF annotation.
x:	Horizontal coordinate of the annotation (in PDF coordinate).

y:	Vertical coordinate of the annotation (in PDF coordinate).
bDefault:	A flag specifying whether to use the default action for left-clicking.

Return Value:

None

11) ^OnAnnotLButtonUp

Triggered when to release left-clicking a PDF annotation.

Prototype:

void OnAnnotLButtonUp(IPDFAnnot* Annot, float x, float y, boolean* bDefault)

Parameters:

Annot:	A pointer to a PDF annotation.
x:	X coordinate of the annotation (in PDF coordinate).
y:	Y coordinate of the moved annotation (in PDF coordinate).
bDefault:	A flag specifying whether to use the default action for releasing left-clicking.

Return Value:

None

12) ^OnAnnotPosChanged

Triggered once the annotation’s position has been changed.

Prototype:

void OnAnnotPosChanged(IPDFAnnot* Annot, float x, float y)

Parameters:

Annot:	A pointer to a PDF annotation.
x:	X coordinate of the annotation (in PDF coordinate).
y:	Y coordinate of the moved annotation (in PDF coordinate).

Return Value:

None

13) ^OnAnnotMoving

Triggered when to move a PDF annotation.

Prototype:

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

Parameters:

Annot:	A pointer to a PDF annotation.
x:	Horizontal coordinate of the annotation (in PDF coordinate).
y:	Vertical coordinate of the moved annotation (in PDF coordinate).

bDefault:	A flag specifying whether to use the default action for moving.

Return Value:

None

14) ^OnAnnotMouseEnter

Triggered when to move the mouse to a PDF annotation.

Prototype:

void OnAnnotMouseEnter(IPDFAnnot* Annot);

Parameters:

Annot:	A pointer to a PDF annotation.

Return Value:

None

15) ^OnAnnotMouseExit

Triggered when to move the mouse away from a PDF annotation.

Prototype:

Void OnAnnotMouseExit(IPDFAnnot* Annot);

Parameters:

Annot:	A pointer to a PDF annotation.

Return Value:

None

16) ^GetRectTop

Get the top coordinate of a PDF annotation object.

Prototype:

Float GetRectTop()

Parameters:

NULL

Return Value:

Return the top coordinate of a PDF annotation object.

17) ^GetRectLeft

Get the left coordinate of a PDF annotation object.

Prototype:

Float GetRectLeft()

Parameters:

[None]

Return Value:

Returns the left coordinate of a PDF annotation object.

18) ^GetRectRight

Get the right coordinate of a PDF annotation object.

Prototype:

Float GetRectRight()

Parameters:

NULL

Return Value:

Return the right coordinate of a PDF annotation object.

19) ^GetRectBottom

Get the bottom coordinate of a PDF annotation object.

Prototype:

Float GetRectBottom()

Parameters:

[None]

Return Value:

Returns the bottom coordinate of a PDF annotation object.

^ IPDFAnnotReplyList

IPDFAnnotReplyList helps to access and operate the reply list of a PDF annotation.

Methods

1) ^ GetCount

Count the replies of a PDF annotation.

Prototype:

long GetCount()

Return Value:

Return the total number of the annotation's replies.

2) ^ GetItem

Get a specific reply for a PDF annotation.

Prototype:

CPDFAnnotReply GetItem(long nIndex)

Parameters:

nIndex:	The index of a annotation reply.

Return Value:

Return a pointer to a specified annotation reply if successful, otherwise NULL.

3) ^ Remove

Remove a specific reply for a PDF annotation.

Prototype:

void Remove(long nIndex)

Parameters:

nIndex:	The index of a annotation reply.

Return Value:

None

4) ^ RemoveAll

Remove all the replies of a PDF annotation.

Prototype:

void RemoveAll()

Parameters:

None

Return Value:

None

5) ^ Add

Add a new reply for a PDF annotation.

Prototype:

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

Parameters:

Creator:	The creator of the new reply.
Content:	The contents of the new reply.
CreationDate:	The date and time when the new reply was created.
nIndex:	The index for the new reply. If it is -1, the new reply will be added at the end of reply list. The value must be between -1 and [GetCount-1].

Return Value:

None

^IPDFAnnotReply

IPDFAnnotReply helps to access and operate an annotation reply.

Methods

1) ^ GetCreator

Get the creator of an annotation reply.

Prototype:

BSTR GetCreator()

Parameters:

None

Return Value:

Return the creator of an annotation reply.

2) ^ SetCreator

Set the creator for an annotation reply.

Prototype:

Void SetCreator(LPCTSTR Creator)

Parameters:

Creator:	The creator for an annotation reply.

Return Value:

None

3) ^ GetContent

Get the contents of an annotation reply.

Prototype:

BSTR GetContent()

Parameters:

None

Return Value:

Return the contents of an annotation reply.

4) ^ SetContent

Set the content for the annotation reply.

Prototype:

Void SetContent(LPCTSTR Content)

Parameters:

Content:	The contents for an annotation reply.

Return Value:

None

5) ^ GetChildren

Get a child reply of a PDF annotation.

Prototype:

IPDFAnnotReplyList* GetChildren()

Parameters:

None

Return Value:

Return a pointer to the child reply of a PDF annotation if successful, otherwise NULL.

6) ^ GetParent

Get a parent reply of a PDF annotation.

Prototype:

CPDFAnnotReply GetParent()

Parameters:

None

Return Value:

Return a pointer to the parent reply of a PDF annotation if successful, otherwise NULL.

7) ^ GetCreationDate

Get the date and time when the reply was created.

Prototype:

DATE GetCreationDate()

Parameters:

None

Return Value:

Return the date and time when the reply was created.

8) ^ SetCreationDate

Set the date and time when the reply was created.

Prototype:

void SetCreationDate(DATE CreationDate)

Parameters:

CreationDate:	The date and time when the reply was created.

Return Value:

None

9) ^ SetReadonly

Set an annotation reply to Read Only.

Prototype:

void SetReadonly(BOOL bNewValue)

Parameters:

bNewValue:	A flag indicating whether to set it to Read Only.

TRUE	-	Set the annotation reply to Read Only.
FALSE	-	Set the annotation reply to Read&Write.

Return Value:

None

10) ^ GetReplyID

Get the ID of an annotation reply.

Prototype:

long GetReplyID()

Parameters:

None

Return Value:

Return an ID of the annotation reply.

^IPDFormatTool

IPDFFormatTool helps format the text annotations in PDF.

ActiveX supports three types of free text annotations, including Typewriter, Callout and TextBox. And each free text annotation maintains independent format data. Please note that the data of IPDFormatTool will be updated once the free text annotation has been changed.

Method

1) ^SetFontName

Set font for the free text annotation.

Prototype:

Void SetFontName(BSTR FontName)

Parameters:

FontName: A font name to be used to display the text of the free text annotation.

Return Value:

None

2) ^GetFontName

Get font name of the free text annotation.

Prototype:

BSTR GetFontName()

Parameters:

None

Return Value:

Return a font name.

3) ^SetFontSize

Set font size for the free text annotation.

Prototype:

Void SetFontSize(float FontSize)

Parameters:

FontSize: Font size of the text in the free text annotation.

Return Value:

None

4) ^GetFontSize

Get font size of the free text annotation.

Prototype:

float GetFontSize()

Parameters:

None

Return Value:

Return font size.

5) ^SetFontColor

Set text color for the free text annotation.

Prototype:

Void SetFontColor(OLE_COLOR FontColor)

Parameters:

FontColor:	The color to be used for the text of the free text annotation.

Return Value:

None

6) ^GetFontColor

Get the text color of the free text annotation.

Prototype:

OLE_COLOR GetFontColor()

Parameters:

None

Return Value:

Return the text color information.

7) ^SetBorderColor

Set the border color for the free text annotation.

Prototype:

Void SetBorderColor(OLE_COLOR color)

Parameters:

Color:	The color to be used for the border of the free text annotation.

Return Value:

None

8) ^GetBorderColor

Get the border color of the free text annotation.

Prototype:

OLE_COLOR GetBorderColor()

Parameters:

None

Return Value:

Return the border color information.

9) ^SetFillColor

Set fill color for the free text annotation.

Prototype:

Void SetFillColor(OLE_COLOR FillColor)

Parameters:

FillColor:	The color to be used for filling the free text annotation.

Return Value:

None

10) ^GetFillColor

Get fill color of the free text annotation.

Prototype:

OLE_COLOR GetFillColor()

Parameters:

None

Return Value:

Return the fill color of the free text annotation.

11) ^SetFontBold

Set the font to use the bold style.

Prototype:

Void SetFontBold(BOOL FontBold)

Parameters:

FontBold:	A value specifying whether to use bold style.

Return Value:

None

Note:

The bold style can be used for the three fonts, including Courier, Helvetica and Times Roman.

12) ^GetFontBold

Check if the font uses the bold style.

Prototype:

boolean GetFontBold()

Parameters:

None

Return Value:

Return TRUE if the bold style is used, otherwise FALSE.

13) ^GetFontBoldEnable

Check if the bold style can be used by the font of the free text annotation.

Prototype:

boolean GetFontBoldEnable()

Parameters:

None

Return Value:

Return TRUE if the bold style can be used, otherwise FALSE.

14) ^SetFontItalic

Set the font to use italic style.

Prototype:

void SetFontItalic(boolean FontItalic)

Parameters:

FontItalic: A value specifying whether to set the font to use italic style.

TRUE	-	Use italic style.
FALSE	-	Not to use italic style.

Return Value:

None

15) ^GetFontItalic

Check if the font uses italic style.

Prototype:

boolean GetFontItalic()

Parameters:

None

Return Value:

Return TRUE if the font uses italic style, otherwise FALSE.

16) ^GetFontItalicEnable

Check if the italic style can be used by the font of the free text annotation.

Prototype:

BOOL GetFontItalicEnable()

Parameters:

None

Return Value:

Return TRUE if the italic style can be used, otherwise FALSE.

17) ^SetAlign

Set text alignment for the free text annotation.

Prototype:

Void SetAlign(AlignStyle Style)

Parameters:

Style: The text alignment. It can be set to:

ASLEFT	-	0,
ASMIDDLE	-	1,
ASRIGHT	-	2

Return Value:

None

18) ^GetAlign

Get the text alignment.

Prototype:

AlignStyle GetAlign()

Parameters:

None

Return Value:

Return the text alignment of the free text annotation. Possible values are:

ASLEFT	=	0,
ASMIDDLE	=	1,
ASRIGHT	=	2

19) ^SetCharSpace

Set character spacing for the text of the free text annotation.

Prototype:

Void SetCharSpace(float CharSpace)

Parameters:

CharSpace:	Character spacing to be used for the text of the free text annotation

Return Value:

None

20) ^GetCharSpace

Get character space information of the free text annotation.

Prototype:

float GetCharSpace()

Parameters:

None

Return Value:

Return character space information of the free text annotation.

21) ^SetCharHorzScale

Set the horizontal scale of the characters of the free text annotation.

Prototype:

Void SetCharHorzScale(float CharHorzScale)

Parameters:

CharHorzScale:	The horizontal scale of the characters.

Return Value:

None

22) ^GetCharHorzScale

Get the horizontal scale of the characters of the free text annotation.

Prototype:

float GetCharHorzScale()

Parameters:

None

Return Value:

Return the horizontal scale of the characters of the free text annotation.

&IPDFSignatureMgr

IPDFSignatureMgr help to operate and manage the signature fields in PDF.

A digital signature can be used to authenticate the identity of a user and the document's contents.

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 a new signature field will be added.
left:	X coordinate of start position in a signature field rectangle.
top:	Y coordinate of start position in a signature field rectangle.
right:	X coordinate of end position in a signature field rectangle.
bottom:	Y coordinate of end position in a signature field rectangle.

Return Value:

Return an IPDFSignatureField object which was newly created.

Note:

The origin of coordinate of the PDF is the left bottom corner.

2) &SignDocument

Sign the unsigned signature fields. It will close and reopen the PDF once the PDF has been signed successfully.

Prototype:

boolean SignDocument(IPDFSignatureField* pSigField, BSTR signedFilePath, boolean bDefault)

Parameters:

pSigField:	An IPDFSignatureField object to be signed.
signedFilePath:	File path of a signed PDF which will be generated after signing.
bDefault:	A flag specifying whether to use the default signing method.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

To sign a PDF with the default signature handler, call SignDocument to complete signing PDF. Or please do as follows to sign a PDF with third-party handler.

1. Get content stream of unsigned PDF by GetSourceBuffer and GetSourceBufferLen, then sign it with a third-party handler. At last, call CreateSignedDoc to complete signing.
2. The IPDFSignatureField object becomes invalid once signing successfully with a standard signature handler. To continue to operate the object, you should reload it.

3) &Verify

Verify a signed signature field.

Prototype:

boolean Verify(IPDFSignatureField* pSigField, boolean bDefaultVerified)

Parameters:

pSigField: A signature field object to be verified.

bDefaultVerified: A flag specifying whether to verify the signature field with the default algorithm. It uses the default algorithm when setting this to TRUE.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

1. Call Verify to complete verifying when the signature uses the default handler.
2. For the signature which uses a third-party signature handler, please do as follows.
   Call GetSourceBuffer and GetSignedBuffer to get the signed PDF, and call SetVerifyResult to deliver the verify result to ActiveX. At last get verify result by GetState(State) of IPDFSignatureField.

4) &VerifyAll

Verify all signed signature fields.

Prototype:

boolean VerifyAll();

Parameters:

None

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

Generally, it asks to verify all the signature fields when to open a PDF. It will not pop up the status dialog by default then.

5) &GetCounts

Count the signature fields in PDF.

Prototype:

long GetCounts();

Parameters:

None

Return Value:

Return the total number of the signature fields, including the signed fields and unsigned fields.

6) &Get

Get a signature field by index.

Prototype:

IPDFSignatureField* Get(long index)

Parameters:

index:	The index of a signature field.

Return Value:

Return an IPDFSignatureField object to the specified signature field.

7) &Clear

Clear the information of a signature field to get an unsigned signature field.

Prototype:

boolean Clear(IPDFSignatureField* pSigField)

Parameters:

pSigField:	An IPDFSignatureField object to be cleared.

Return Value:

Return TRUE if successful, otherwise FALSE.

8) &Remove

Remove an unsigned signature field (object).

Prototype:

boolean Remove(IPDFSignatureField* pSigField)

Parameters:

pSigField:	An IPDFSignatureField object to be removed.

Return Value:

Return TRUE if successful, otherwise FALSE.

9) &InitStraddleValue

Add straddle sign.

Prototype:

BOOL InitStraddleValue(IPDFSignatureField* pSigField)

Parameters:

pSigField:	An IPDFSignatureField object.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

It requires the signing procedure, including a standard signature and a third-party signature, to complete the straddle sign.

10) &CreatePatternSigField

Create signature template, which can be used for adding signature objects.

Prototype:

VARIANT_BOOL CreatePatternSigField(BSTR imageFilePath, VARIANT_BOOL bSetMask, OLE_COLOR clrMask, FLOAT Height, FLOAT Width)

Parameters:

imageFilePath:	File path of an image for the signature template.
bSetMask:	A flag specifying whether to set mask.
clrMask:	The mask color.
Height:	The height.
Width:	The width.

Return Value:

Return an IPDFSignatureField object.

11) &SetCurPattenSigField

Set a signature template.

Prototype:

boolean SetCurPatternSigField(SHORT nIndex)

Parameters:

nIndex:	The index of a signature template.

Return Value:

Return TRUE if successful, otherwise FALSE.

12) &CountPatternSigFiel

Count the signature templates.

Prototype:

SHORT CountPatternSigField(void)

Parameters:

None.

Return Value:

Return the total number of the signature templates.

&IPDFSignatureField

IPDFSignatureField help access and operate a signature field.

Properties

1) &Reason

Type:

String

Description:

The reason for the signing.

Operation:

Read and Write

2) &Location

Type:

String

Description:

The location of a signature field.

Operation:

Read and Write

3) &Signer

Type:

String

Description:

The signer of the signature field.

Operation:

Read and Write

4) &Filter

Type:

String

Description:

The name of the preferred signature handler to use when validating this signature. The default is Adobe.PPKLite.

Operation:

Read and Write

5) &SubFilter

Type:

String

Description:

A name that describes the encoding of the signature value and key information in the signature dictionary. An application may use any handler that supports this format to validate the signature. The default is adbe.pkcs7.detached.

6) &State

Type:

Short

Description:

The state of a signature. If a signature field is signed by default, the state is decided by ActiveX. Or it is defined and will be sent to ActiveX by SetVerifyResult.

0	-	Unknown signature
1	-	Not signed
2	-	Pass verification
3	-	Did not pass verification

Operation:

Read only

Methods

1) &SetAPOptions

Customize signature appearance.

Prototype:

boolean SetAPOptions(long opts);

Parameters:

Opts: A value indicating the custom signature appearance. It can be set to:

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

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This should be done before calling SignDocument.

2) &SetAPText

Set text for the signature appearance.

Prototype:

boolean SetAPText(BSTR text);

Parameters:

text: The text to be displayed.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This should be done before calling SignDocument.

3) &SetAPImage

Set image for the signature appearance.

Prototype:

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

Parameters:

imageFilePath: File path of an image to be displayed.

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

clrMask: The mask color.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This should be done before calling SignDocument.

4) &SetAPImageData

Set image for the signature appearance.

Prototype:

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

Parameters:

imageDataBuffer:	Pointer to a buffer specifying the image data.
imageType:	The image type.
dataSize:	The size of the image data.
bSetMask:	A flag specifying whether to set mask.
clrMask:	The mask color.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

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

5) &IsSigned

Check if the signature field is signed.

Prototype:

boolean IsSigned();

Return Value:

Return TRUE if the signature field is signed, otherwise FALSE.

6) &SetSignerDN

Set a distinguished name for the certificate.

Prototype:

Boolean SetSignerDN(BSTR dn);

Parameters:

dn :	A distinguished name for the certificate.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This should be done before calling SignDocument.

7) &SetStatusImage

Set status image.

Prototype:

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

Parameters:

imagePath: File path of an image to be used as the state icon.

sState: The state of the signature. It decided by State.

sMode: A value specifying the image display mode. It can be set to:

- 0 - Displaying the image in top left corner. (Default)
- 1 - The image covers the whole signature field.

bRotate: A value specifying whether to rotate image when to the signature. It does not rotate the image by default.

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

clrMask: The mask color.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

The image can’t be saved into PDF. You can only set the image before the status of Signature is changed.

8) &SetStatusImageData

Prototype:

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

Parameters:

imageDataBuffer: Pointer to a buffer specifying the image data to be used as the state icon.

imageType: The state of the signature. It supports bmp, jpg, png and gif.

dataSize: The size of the image data.

sState: The state of a signature. It decided by State.

sMode : A value specifying the image display mode. It can be set to:

- 0 - Displaying the image in top left corner. (Default)
- 1 - The image covers the whole signature field.

bRotate: A value specifying whether to rotate image when to rotate the signature. It does not rotate the image by default.

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

clrMask: The mask color.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

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

9) &GetPageIndex

Get the index of the page where the signature field is.

Prototype:

Long GetPageIndex();

Parameters:

None

Parameters:

None

Return Value:

Return the index of the page.

10) &GetSourceBuffer

Get the source PDF to be signed or verified.

Prototype:

VARIANT GetSourceBuffer();

Parameters:

None

Return Value:

Return a buffer specifying the source content stream.

Note:

1. Call SignDocument or Verify to get TRUE first, and then call this interface.
2. The content stream is saved in parray field of VARIVANT. Please get the content stream pointer with parray->pvData in VC. And get content stream array with getArray in JS.

11) &GetSourceBufferLen

Get the length of the source PDF to be signed or verified.

Prototype:

long GetSourceBufferLen();

Return Value:

Return the length of the source PDF to be signed or verified.

Note:

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

12) &GetSignedBuffer

Get the signed PDF content.

Prototype:

VARIANT GetSignedBuffer();

Return Value:

Return a buffer specifying the signed PDF content stream.

Note:

1. Call SignDocument or Verify to get TRUE first, and then call this interface.
2. The content stream is saved in parray field of VARIVANT. Please get the content stream pointer with parray->pvData in VC. And get content stream array with getArray in JS.

13) &GetSignedBufferLen

Get the length of the signed PDF content.

Prototype:

long GetSignedBufferLen();

Return Value:

Return the length of the signed PDF content.

Note:

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

14) &CreateSignedDoc

Create a signed PDF which is signed by a third-party handler. The path is in SignDocument.

Prototype:

boolean CreateSignedDoc(VARIANT signedBuf, long length);

Parameters:

signedBuf:	Pointer to the buffer specifying the signed PDF data.
length:	The length of the signed PDF data.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

Call SignDocument to set file path for the signed PDF, without using the default signature handler.

And then call this function.

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 reloaded.

15) &SetVerifyResult

Customize the result for verifying a signature by a third-party signature handler.

Prototype:

boolean SetVerifyResult(short sResult)

Parameters:

sResult: A value indicating the verifying result.

0	-	Unknown signature
1	-	Not signed
2	-	Pass verification
3	-	Did not pass verification

Return Value:

Return TRUE if successful, otherwise FALSE.

16) &SetCertPath

Set certificate.

Prototype:

boolean SetCertPath(BSTR certPath, BSTR pfxPsw);

Parameters:

certPath:	File path of a certificate.
pfxPsw:	Password for accessing the certificate.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

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

17) &SetCertData

Set certification Data.

Prototype:

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

Parameters:

pCertData: The certificate data

length:	The length of the certificate data.
pfxPsw :	Password for accessing the certificate.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

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

18) &SetCertContext

Set certificate context.

Prototype:

boolean SetCertContext(long pCertContext)

Parameters:

pCertContext:	The type of certificate context.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

The certificate will take effect when to call any of the three functions, including &SetCertPath, &SetCertData and &SetCertContext.

19) &TurnGray

Turn the signature appearance to gray.

Prototype:

boolean TurnGray(boolean bGray,boolean bCanModify);

Parameters:

bGray:	A value specifying whether to turn the appearance to gray.
bCanModify:	A value specifying whether the signature can be modified

Return Value:

Return TRUE if successful, otherwise FALSE.

20) &TurnBlur

Blur the signature field.

Prototype:

boolean TurnBlur(boolean bBlur);

Parameters:

bBlur:	A value specifying whether to blur the signature field.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This only changes the signature displaying instead of the PDF content.

21) &SetVisible

Set signature to be visible.

Prototype:

boolean SetVisible(boolean bVisible)

Parameters:

bVisible:	A value specifying whether the signature is visible.

Return Value:

Return TRUE if successful, otherwise FALSE.

22) &GrayPrint

Turn the signature appearance to gray for PDF printing.

Prototype:

boolean GrayPrint(boolean bGrayPrint);

Parameters:

bGrayPrint:	A value specifying whether to turn the appearance to gray for PDF printing.

Return Value:

Return TRUE if successful, otherwise FALSE.

Note:

This only changes the signature displaying instead of the PDF content.

23) &SetStraddleType

Set the type of the straddle sign.

Prototype:

BOOL SetStraddleType(short nType)

Parameters:

nType: The type of the straddle sign. It can be set to:

0	-	Middle Straddle Sign
1	-	Left Straddle Sign
2	-	Right Straddle Sign
3	-	Top Straddle Sign
4	-	Bottom Straddle Sign

Return Value:

Return TRUE if successful, otherwise FALSE.

24) &SetStraddlePos

Set the position of straddle sign.

Prototype:

BOOL SetStraddlePos(float fPos)

Parameters:

fPos:	The position of the straddle sign.

For Top Straddle Sign and Bottom Straddle Sign, the value will be the horizontal middle point of the signature field.

For Right Straddle Sign and Left Straddle Sign, the value will be the vertical middle point of the signature field.

Return Value:

Return TRUE if successful, otherwise FALSE.

25) &SetStraddleBitmap

Set image for straddle sign appearance.

Prototype:

boolean SetStraddleBitmap(short nState, BSTR sFilePath)

Parameters:

nState: The state of straddle sign. It can be set to:

0	-	Unknown
1	-	Unsigned
2	-	Valid
3	-	Invalid

sFilePath: File path of an image for straddle sign appearance.

Return Value:

Return TRUE if successful, otherwise FALSE.

26) &SetStraddlePages

Set the page range where the straddle sign will be placed.

Prototype:

BOOL SetStraddlePages(BSTR sRange)

Parameters:

sRange: The page range where the straddle sign will be placed. The value must be in format as “0-10”, “0-2,3-10”.

Return Value:

Return TRUE if successful, otherwise FALSE.

27) &SetStraddleFirstPagePercent

Set the percent of the straddle sign to be divided and displayed in the first signed PDF page.

Prototype:

BOOL SetStraddleFirstPagePercent(float fPercent)

Parameters:

fPercent:	The percent of a straddle sign for the first signed page. It can be from 0 to 1.

Return Value:

Return TRUE if successful, otherwise FALSE.

28) &SetSigFieldAlpha

Set the transparency of the signature image.

Prototype:

SetSigFieldAlpha(short alpha)

Parameters:

alpha:	A value specifying the image transparency. It can be from 0 to 255.

Return Value:

Return TRUE if successful, otherwise FALSE.

29) &Refresh

Refresh the signature field.

Prototype:

void Refresh()

Parameters:

NULL

Return Value:

NULL

Note:

This could be used only when a signature object is uncompleted.

30) &SetDefaultContentsLength

Set length of the content to be encrypted when using a third-party encryption algorithm.

Prototype:

Bool SetDefaultContentsLength(long length);

Parameters:

length: The length of the content to be encrypted. The value can be half of the content length at most.

Return Value:

Return TRUE if successful, otherwise FALSE.

Events

1) &OnSetSignatureInfo

Triggered when to click or right click an unsigned 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 a signature field is unsigned. The signature field will be signed by the default or a third-party method.

Prototype:

void OnSigning(IPDFSignatureField* pSignature)

Parameters:

pSignature: A signature field to be signed.

3) &OnVerifying

Triggered when to click a signed signature field or right click to select Verify option. It will be verified by default or by third-party handler.

Prototype:

void OnVerifying(IPDFSignatureField* pSignature);

Parameters:

pSignature: A signature field to be verified.

4) &OnShowSignaturePropertyDialog

Triggered when to right click a signature field and then choose Properties option to check the properties information.

Prototype:

Void OnShowSignaturePropertyDialog (IPDFSignatureField*pSignature, boolean* bShowProperty)

Parameters:

pSignature:	The signature field to be right-clicked.
bShowProperty:	A value specifying whether to pop up Properties dialog for the signature field.

*IPDFTextDoc

Methods

1) *ReleasePDFTextDoc

Release all resources allocated for a PDF text document.

Prototype:

void ReleasePDFTextDoc()

Return Value:

None

2) *LoadPDFTextPage

Prepare information about all the characters in a PDF page.

Prototype:

IPDFTextPage*LoadPDFTextPage(long pageIndex)

Parameters:

pageIndex:	A zero-based index of a PDF page.

Return Value:

Return an IPDFTextPage object.

*IPDFTextPage

Methods

1) *ReleasePDFTextPage

Release all resources allocated for a PDF text page.

Prototype:

void ReleasePDFTextPage()

Return Value:

None

2) *CountChars

Count the characters in a page.

Prototype:

long CountChars()

Return Value:

Return the total number of the characters in the page.

3) *GetChars

Get text content from a PDF page by specifying the character range.

Prototype:

BSTR GetChars(long start, long count);

Parameters:

start: The index of the first character in text content to be got. The value is from 0 to (CountChars - 1).

count: The total number of characters to be got. If -1, it will get all the characters in the PDF page. And if count > (CountChars - start), all the rest characters will be got.

Return Value:

Return a text string.

4) *GetCharInfo

Get information of a specified character.

Prototype:

IPDFCharInfo* GetCharInfo(long charIndex);

Parameters:

charIndex: The index of a character. The value is from 0 to (CountChars – 1).

Return Value:

Return an IPDFCharInfo object.

5) *GetCharIndexAtPos

Get index of a character at or nearby a specified position on the page.

Prototype:

long GetCharIndexAtPos (float x, float y, float tolerance);

Parameters:

x:	X coordinate in device coordinate.
y:	Y coordinate in device coordinate.
tolerance:	Tolerance value for character hit detection and it is measured in Point. This should not be a negative.

Return Value:

Return the index of a character.

6) *GetNextCharIndexByDirection

Get the index of next character in specified direction.

Prototype:

long GetNextCharIndexByDirection(long curIndex, short direction);

Parameters:

curIndex: The index of current character.

direction: A value indicates the direction to get next character. It can be se to: -1, 1, -2 and 2.

-1	-	Text direction: left.
1	-	Text direction: right.
-2	-	Text direction: up.
2	-	Text direction: down.

Return Value:

Return an index of the next character. The three special values, including -1, -2 and -3, point out the errors.

-1	-	Reach the beginning of the page.
-2	-	Reach the end of the page.
-3	-	Other errors.

7) *SelectByRange

Get a text selection object by specifying character range.

Prototype:

IPDFTextSelection* SelectByRange (long start, long count);

Parameters:

start: The index of the first character to be selected.

count: The count of characters to be selected. If -1, it will select all the characters in the PDF page.

Return Value:

Return an IPDFTextSelection object.

8) *SelectByRectangle

Get a text selection object in a specified rectangle.

Prototype:

IPDFTextSelection* SelectByRectangle (long left, long top, long right, long bottom);

Parameters:

left: The left coordinate of a rectangle.

top: The top coordinate of a rectangle.

right: The right coordinate of a rectangle.

bottom : The bottom coordinate of a rectangle.

Return Value:

Return an IPDFTextSelection object.

9) *StartSearch

Start a PDF text search.

Prototype:

IPDFTextSearch* StartSearch(BSTR searchPattern, long flags, long startIndex);

Parameters:

searchPattern: A keyword for the text search.

flags: The settings for the search. It can be set to 0, one or combination of 1, 2 and 4.

0	-	Not any settings.
1	-	Match case
2	-	Match whole word
4	-	Consecutive

startIndex: The index of the first character to be searched and the value is from 0. If -1, the search will be from the end of the page.

Return Value:

Return an IPDFTextSearch object if successful.

10) *ExtractLinks

Get a URL formatted link object from the PDF text page.

Prototype:

IPDFTextLink* ExtractLinks ();

Return Value:

Return an IPDFTextLink object

11) *ExtractPageText

Get the text content from a PDF text page.

Prototype:

BSTR ExtractPageText ();

Return Value:

Return the text content of a PDF text page.

*IPDFCharInfo

Properties

1) *state

Type:

short. Read only.

Description:

State of the character. Possible values are:

1	-	Normal character.
2	-	Character is generated by Foxit, such as space character.
3	-	Character doesn't have its own unicode value.

2) *fontSize

Type:

float. Read only.

Description:

The original font size of the character and it is measured in Point (1/72 Inch).

3) *originX

Type:

float. Read only.

Description:

X position of the character and it is based on PDF Page Coordinate. -1 means error.

4) *originY

Type:

float. Read only.

Description:

Y position of the character and and it is based on PDF Page Coordinate.

5) *fontName

Type:

BSTR. Read only.

Description:

The font name.

6) *fontAscent

Type:

long. Read only.

Description:

An ascent value of the font.

7) *fontDescent

Type:

long. Read only.

Description:

A descent value of the font.

8) *displayFontSize

Type:

float

Description:

Fontsize for text displaying in viewer.

Methods

1) *GetBBox

Get 4 positions (left, bottom, right, top) of character's bounding box.

Prototype:

VARIANT GetBBox();

Parameters:

None

Return Value:

Return an array including 4 float members, which represent a rectangle.

0	-	left
1	-	top
2	-	right
3	-	bottom

2) *GetMatrix

Get the matrix of a character.

Prototype:

```
VARIANT GetMatrix();
```

Parameters:

None

Return Value:

Return an array including 6 float members, which represent a matrix.

0	-	a
1	-	b
2	-	c
3	-	d
4	-	e
5	-	f

*IPDFTextSearch

Methods

1) *ReleaseTextSearch

Release all resources allocated for a PDF text search module.

Prototype:

void ReleaseTextSearch()

Parameters:

None

Return Value:

None

2) *FindNext

Search the text from start to end of the page.

Prototype:

BOOL FindNext ()

Parameters:

None

Return Value:

Return a value indicating whether a match is found.

3) *FindPrev

Search the text from end to start of the page.

Prototype:

BOOL FindPrev()

Parameters:

None

Return Value:

Return a value indicating whether a match is found.

4) *GetSelection

Get a text selection from a text search when a match is found.

Prototype:

IPDFTextSelection* GetSelection();

Parameters:

None

Return Value:

Return a text selection object.

*IPDFTextSelection

Methods

1) *ReleaseTextSelection

Release all resources allocated for a PDF text selection object.

Prototype:

void ReleaseTextSelection()

Parameters:

None

Return Value:

None

2) *GetBBox

Get bounding box (a rectangle area) of a PDF text selection.

Prototype:

VARIANT GetBBox()

Parameters:

None

Return Value:

An array contains four float members, which represent a rectangle.

0	-	left
1	-	top
2	-	right
3	-	bottom

3) *GetBBoxEx

Get bounding box (a rectangle area) of a PDF text selection. This can be called by Javascript.

Prototype:

VARIANT GetBBoxEx();

Parameters:

None

Return Value:

Return an array including 4 float members, which represent a rectangle.

0	-	left

1	-	top
2	-	right
3	-	bottom

4) *GetChars

Extract all the text from a PDF text selection.

Prototype:

BSTR GetChars()

Return Value:

Return the text string.

5) *CountPieces

Count the rectangular areas of segments in a PDF text selected area.

Prototype:

long CountPieces ()

Return Value:

Return the total number of segments in a PDF text selected area.

6) *GetPieceRect

Get rectangular area of a specific segment, based on the result returned by function CountPieces.

Prototype:

VARIANT GetPieceRect(long pieceIndex);

Parameters:

pieceIndex	-	The index of a segment to be got.

Return Value:

Return an array including 4 float members, which represent a rectangle.

0	-	left
1	-	top
2	-	right
3	-	bottom

7) *GetPieceCharStart

Get the index of the first character in a specified segment.

Prototype:

long GetPieceCharStart(long pieceIndex);

Parameters:

pieceIndex - The index of a segment to be got.

Return Value:

Return the index of the first character in a segment. -1 means error.

8) *GetPieceCharCount

Count the characters of a specified segment.

Prototype:

```
long GetPieceCharCount(long pieceIndex);
```

Parameters:

pieceIndex - The index of a segment to be got.

Return Value:

Return the total number of characters in the specified segment. -1 means error.

*IPDFTextLink

Methods

1) *CountLinks

Get count of the URL formatted texts in a PDF page.

Prototype:

long CountLinks()

Parameters:

None

Return Value:

Return the total number of the URL formatted texts. -1 means error.

2) *GetLink

Get the URL associated with a specified hyperlink.

Prototype:

BSTR GetLink (long linkIndex);

Parameters:

linkIndex:	The index of a hyperlink.

Return Value:

Return an URL.

3) *GetSelection

Get a PDF text selection object from a specified hyperlink.

Prototype:

IPDFTextSelection* GetSelection (long linkIndex);

Parameters:

linkIndex:	The index of a hyperlink.

Return Value:

Return a PDF text selection object.

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 Software Incorporated

  41841 Albrae Street

  Fremont CA 94538

  USA

- ***Sales:***

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

- ***Support:***

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

- ***Fax:***

  1-510-405-9288

- ***Website:***

  www.foxitsoftware.com

- ***E-mail:***

  Sales and Information - sales@foxitsoftware.com

  Technical Support - Input a trouble ticket online

  Marketing Service - marketing@foxitsoftware.com