Interapplication Communication with the Acrobat SDK
## Contents

**List of Examples** ......................................................................................................................... 12

1 **Developing Applications Using IAC** .......................................................................................... 13
   - About the API object layers .................................................................................................. 13
   - Object reference syntax ........................................................................................................ 13
   - Objects in the Acrobat DC application layer ........................................................................ 14
   - Objects in the portable document layer ............................................................................... 15
   - Plugins for extending the IAC interfaces ............................................................................ 16
   - Developing for Acrobat Reader .............................................................................................. 16
   - DDE messages ........................................................................................................................ 19
   - Apple events .......................................................................................................................... 19

2 **Using OLE** .................................................................................................................................. 20
   - OLE capabilities in Acrobat DC ............................................................................................ 21
   - On-screen rendering .............................................................................................................. 21
   - Remote control of Acrobat DC ............................................................................................. 21
   - PDF browser controls ........................................................................................................... 21
   - Development environment considerations ........................................................................... 22
   - Environment configuration ................................................................................................... 23
   - Necessary C knowledge ......................................................................................................... 24
   - Using the Acrobat DC OLE interfaces .................................................................................... 25
     - About the CAcro classes .................................................................................................. 25
     - About the COleDispatchDriver class ............................................................................. 25
     - Using COleDispatchDriver objects and methods ............................................................. 25
   - Using the JSObject interface .................................................................................................. 29
     - Adding a reference to the Acrobat DC type library .......................................................... 29
     - Creating a simple application ......................................................................................... 30
     - Working with annotations ................................................................................................. 31
     - Spell-checking a document ............................................................................................... 34
   - Tips for translating JavaScript to JSObject ............................................................................ 36
   - Other development topics ..................................................................................................... 37
     - Synchronous messaging .................................................................................................... 37
     - MDI applications ............................................................................................................... 37
     - Event handling in child windows ....................................................................................... 37
     - Determining if an Acrobat DC application is running ....................................................... 38
     - Exiting from an application ............................................................................................... 38
     - Summary of OLE objects and methods ............................................................................ 39

3 **Using DDE** ............................................................................................................................... 40

4 **Using Apple Events** ............................................................................................................... 41

5 **OLE Automation** .................................................................................................................... 42
   - AcroExch.App ....................................................................................................................... 42
   - CloseAllDocs ....................................................................................................................... 44
   - Exit ...................................................................................................................................... 44
   - GetActiveDoc ....................................................................................................................... 45
   - GetActiveTool ...................................................................................................................... 45
GetAVDoc ................................................................. 46
GetFrame ................................................................. 46
GetInterface ............................................................. 46
GetLanguage ............................................................. 47
GetNumAVDocs .......................................................... 48
GetPreference ............................................................ 48
GetPreferenceEx ......................................................... 49
Hide ............................................................................. 49
Lock ............................................................................ 49
Minimize ................................................................. 50
Maximize ................................................................. 51
MenuItemExecute ........................................................ 51
MenuItemIsEnabled ..................................................... 52
MenuItemIsMarked ..................................................... 52
MenuItemRemove ....................................................... 53
Restore ........................................................................ 53
SetActiveTool ............................................................ 54
SetFrame ................................................................. 54
SetPreference ............................................................ 55
SetPreferenceEx ........................................................ 55
Show ......................................................................... 56
ToolButtonIsEnabled .................................................. 56
ToolButtonRemove ..................................................... 57
Unlock ................................................................. 57
UnlockEx ................................................................. 58

AcroExch.AVDoc .......................................................... 58
BringToFront ............................................................ 59
ClearSelection ........................................................... 60
Close ......................................................................... 60
FindText ................................................................. 61
GetAVPageView ......................................................... 62
GetFrame ................................................................. 62
GetPDDoc ................................................................. 62
GetTitle ...................................................................... 63
GetViewMode ............................................................ 63
IsValid ................................................................. 64
Maximize ................................................................. 64
Open ......................................................................... 64
OpenInWindow .......................................................... 65
OpenInWindowEx ....................................................... 66
PrintPages ................................................................. 68
PrintPagesEx ............................................................. 69
PrintPagesSilent ........................................................ 70
PrintPagesSilentEx .................................................... 70
SetFrame ................................................................. 71
SetTextSelection ....................................................... 72
SetTitle ................................................................. 73
SetViewMode ............................................................ 73
ShowTextSelect ......................................................... 74

AcroExch.AVPageView ............................................... 76
DevicePointToPage ..................................................... 76
DoGoBack .................................................................................................................................................................................77
DoGoForward .............................................................................................................................................................................77
GetAperture ............................................................................................................................................................................78
GetAVDoc .............................................................................................................................................................................78
GetDoc ..................................................................................................................................................................................78
GetPage ..................................................................................................................................................................................79
GetPageNum ..........................................................................................................................................................................79
GetZoom ..............................................................................................................................................................................80
GetZoomType .......................................................................................................................................................................80
Goto ......................................................................................................................................................................................81
PointToDevice .......................................................................................................................................................................82
ReadPageDown ....................................................................................................................................................................82
ReadPageUp ..........................................................................................................................................................................83
ScrollTo ...............................................................................................................................................................................83
ZoomTo ...............................................................................................................................................................................84
AcroExch.HiliteList ...............................................................................................................................................................85
Add ......................................................................................................................................................................................85
AcroExch.PDAnnot ...............................................................................................................................................................85
GetColor ...............................................................................................................................................................................86
GetContents .........................................................................................................................................................................87
GetDate ...............................................................................................................................................................................87
GetRect ...............................................................................................................................................................................87
GetSubtype .........................................................................................................................................................................88
GetTitle ...............................................................................................................................................................................88
IsEqual ...............................................................................................................................................................................89
IsOpen ...............................................................................................................................................................................89
IsValid ..............................................................................................................................................................................90
Perform ...............................................................................................................................................................................91
SetColor .............................................................................................................................................................................91
SetContents .......................................................................................................................................................................92
SetDate ..............................................................................................................................................................................92
SetOpen ..............................................................................................................................................................................93
SetRect ..............................................................................................................................................................................93
SetTitle ............................................................................................................................................................................94
AcroExch.PDBookmark ........................................................................................................................................................95
Destroy .................................................................................................................................................................................95
GetByTitle .........................................................................................................................................................................96
GetTitle .............................................................................................................................................................................96
IsValid ............................................................................................................................................................................97
Perform ............................................................................................................................................................................97
SetTitle ............................................................................................................................................................................98
AcroExch.PDDoc .................................................................................................................................................................98
AcquirePage ......................................................................................................................................................................100
ClearFlags .........................................................................................................................................................................100
Close ................................................................................................................................................................................101
Create ...............................................................................................................................................................................101
CreateTextSelect ..............................................................................................................................................................102
CreateThumbs ...................................................................................................................................................................103
CropPages .........................................................................................................................................................................103
DeletePages ......................................................................................................................................................................104
DeleteThumbs .................................................................................................................................................................104
AcroExch праздник

AcroExch.PDPage

AddAnnot

AddNewAnnot

CopyToClipboard

CreatePageHilite

CreateWordHilite

CropPage

Draw

DrawEx

GetAnnot

GetAnnotIndex

GetDoc

GetNumAnnots

GetNumber

GetRotate

GetSize

RemoveAnnot

SetRotate

AcroExch.PDTextSelect

Destroy

GetBoundingRect

GetNumText

GetPage

GetText

AcroExch.Point

X

Y

AcroExch.Rect

Bottom

Left

Right

Top

AcroExch.Time

Date
6 DDE Messages .......................................................................................................................... 150

AppExit .................................................................................................................................................. 151
AppHide ................................................................................................................................................... 152
AppShow ................................................................................................................................................ 152
CloseAllDocs ........................................................................................................................................ 152
DocClose ............................................................................................................................................... 153
DocDeletePages .................................................................................................................................. 153
DocFind ............................................................................................................................................... 154
DocGoTo .............................................................................................................................................. 154
DocGoToNameDest ............................................................................................................................... 155
DocInsertPages ................................................................................................................................... 155
DocOpen ............................................................................................................................................... 155
DocPageDown ...................................................................................................................................... 156
DocPageLeft ...................................................................................................................................... 157
DocPageRight .................................................................................................................................... 158
DocPageUp .......................................................................................................................................... 158

Hour ................................................................................................................................................... 135
Minute .................................................................................................................................................. 135
Month ................................................................................................................................................... 135
Second .................................................................................................................................................. 136
Year ...................................................................................................................................................... 136

AxAcroPDFLib.AxAcroPDF .................................................................................................................. 136
GetVersions ......................................................................................................................................... 138
GoBackwardStack ............................................................................................................................... 138
GoForwardStack ............................................................................................................................... 138
GotoFirstPage ..................................................................................................................................... 138
GotoLastPage ...................................................................................................................................... 139
GotoNextPage ...................................................................................................................................... 139
GotoPreviousPage .............................................................................................................................. 139
LoadFile ............................................................................................................................................. 140
Print ..................................................................................................................................................... 140
PrintAll ............................................................................................................................................... 141
PrintAllFit .......................................................................................................................................... 141
PrintPages ......................................................................................................................................... 142
PrintPagesFit ....................................................................................................................................... 142
PrintWithDialog ................................................................................................................................. 143
SetCurrentHighlight .......................................................................................................................... 143
SetCurrentPage .................................................................................................................................... 144
SetLayoutMode .................................................................................................................................... 144
SetNamedDest ..................................................................................................................................... 145
SetPageMode ...................................................................................................................................... 145
SetShowScrollbars ............................................................................................................................ 146
SetShowToolbar ................................................................................................................................. 146
SetView ............................................................................................................................................... 146
SetViewRect ........................................................................................................................................ 147
SetViewScroll ..................................................................................................................................... 148
SetZoom .............................................................................................................................................. 148
SetZoomScroll .................................................................................................................................... 149
Src ....................................................................................................................................................... 149

Year ...................................................................................................................................................... 136
Second ................................................................................................................................................... 136
Month ................................................................................................................................................... 135
Second .................................................................................................................................................. 136
Year ...................................................................................................................................................... 136

AxAcroPDFLib.AxAcroPDF .................................................................................................................. 136
GetVersions ......................................................................................................................................... 138
GoBackwardStack ............................................................................................................................... 138
GoForwardStack ............................................................................................................................... 138
GotoFirstPage ..................................................................................................................................... 138
GotoLastPage ...................................................................................................................................... 139
GotoNextPage ...................................................................................................................................... 139
GotoPreviousPage .............................................................................................................................. 139
LoadFile ............................................................................................................................................. 140
Print ..................................................................................................................................................... 140
PrintAll ............................................................................................................................................... 141
PrintAllFit .......................................................................................................................................... 141
PrintPages ......................................................................................................................................... 142
PrintPagesFit ....................................................................................................................................... 142
PrintWithDialog ................................................................................................................................. 143
SetCurrentHighlight .......................................................................................................................... 143
SetCurrentPage .................................................................................................................................... 144
SetLayoutMode .................................................................................................................................... 144
SetNamedDest ..................................................................................................................................... 145
SetPageMode ...................................................................................................................................... 145
SetShowScrollbars ............................................................................................................................ 146
SetShowToolbar ................................................................................................................................. 146
SetView ............................................................................................................................................... 146
SetViewRect ........................................................................................................................................ 147
SetViewScroll ..................................................................................................................................... 148
SetZoom .............................................................................................................................................. 148
SetZoomScroll .................................................................................................................................... 149
Src ....................................................................................................................................................... 149

6 DDE Messages .......................................................................................................................... 150
7 Apple Event Objects and Apple Events

<table>
<thead>
<tr>
<th>Object</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>171</td>
</tr>
<tr>
<td>application</td>
<td>172</td>
</tr>
<tr>
<td>AVPageView</td>
<td>175</td>
</tr>
<tr>
<td>bookmark</td>
<td>175</td>
</tr>
<tr>
<td>conversion</td>
<td>177</td>
</tr>
<tr>
<td>document</td>
<td>177</td>
</tr>
<tr>
<td>EPS Conversion</td>
<td>179</td>
</tr>
<tr>
<td>Link Annotation</td>
<td>179</td>
</tr>
<tr>
<td>menu</td>
<td>179</td>
</tr>
<tr>
<td>menu item</td>
<td>180</td>
</tr>
<tr>
<td>page</td>
<td>181</td>
</tr>
<tr>
<td>PDAnnot</td>
<td>182</td>
</tr>
<tr>
<td>PDBookMark</td>
<td>182</td>
</tr>
<tr>
<td>PDLinkAnnot</td>
<td>182</td>
</tr>
<tr>
<td>PDPage</td>
<td>182</td>
</tr>
<tr>
<td>PDPaperAnnot</td>
<td>182</td>
</tr>
<tr>
<td>PDF Window</td>
<td>182</td>
</tr>
<tr>
<td>PostScript Conversion</td>
<td>183</td>
</tr>
<tr>
<td>Text Annotation</td>
<td>184</td>
</tr>
<tr>
<td>Required suite events</td>
<td>184</td>
</tr>
<tr>
<td>open</td>
<td>185</td>
</tr>
<tr>
<td>print</td>
<td>185</td>
</tr>
<tr>
<td>quit</td>
<td>185</td>
</tr>
<tr>
<td>run</td>
<td>185</td>
</tr>
<tr>
<td>Core suite events</td>
<td>186</td>
</tr>
<tr>
<td>close</td>
<td>186</td>
</tr>
<tr>
<td>count</td>
<td>186</td>
</tr>
<tr>
<td>delete</td>
<td>187</td>
</tr>
</tbody>
</table>
exists................................................................................................................................. 187
get........................................................................................................................................ 188
make .................................................................................................................................. 188
move.................................................................................................................................... 189
open..................................................................................................................................... 190
quit...................................................................................................................................... 190
save ..................................................................................................................................... 190
set ....................................................................................................................................... 191

Acrobat DC application events............................................................................................ 191
bring to front ...................................................................................................................... 192
clear selection ................................................................................................................... 193
close all docs .................................................................................................................... 193
create thumbs ................................................................................................................... 194
delete pages .................................................................................................................... 195
delete thumbs .............................................................................................................. 195
execute .............................................................................................................................. 196
find next note ................................................................................................................... 196
find text ............................................................................................................................. 197
get info .............................................................................................................................. 198
go backward ..................................................................................................................... 198
go forward ........................................................................................................................ 199
goto .................................................................................................................................... 199
goto next .......................................................................................................................... 200
goto previous ................................................................................................................... 201
insert pages ...................................................................................................................... 201
is toolbutton enabled ....................................................................................................... 202
maximize ........................................................................................................................... 203
perform ............................................................................................................................... 203
print pages ....................................................................................................................... 203
read page down ............................................................................................................... 204
read page up .................................................................................................................... 205
remove toolbutton .......................................................................................................... 206
replace pages .................................................................................................................. 206
scroll .................................................................................................................................. 207
select text .......................................................................................................................... 208
set info .............................................................................................................................. 209
zoom .................................................................................................................................. 209

Miscellaneous events........................................................................................................ 210
do script ............................................................................................................................. 210

8 Acrobat DC Catalog Plug-In ............................................................................................ 211
Catalog Windows messages ............................................................................................. 211
Catalog DDE methods ..................................................................................................... 211
AppExit ............................................................................................................................ 211
AppFront .......................................................................................................................... 211
FileBuild .......................................................................................................................... 212
FileOpen ........................................................................................................................... 212
FilePurge .......................................................................................................................... 212

9 Acrobat DC Forms Plug-In ............................................................................................ 213
Forms plug-in OLE automation ......................................................................................... 213
Exceptions ....................................................................................................................... 213
Developing Applications Using Interapplication Communication

Adobe Acrobat SDK

NewEnum
Remove
ImportAnFDF
ExportAsHtml
ExportAsFDF
ExecuteThisJavascript
AddDocJavascript
SetJavaScriptAction
SetExportValues
SetButtonIcon
SetButtonCaption
SetForeColor
SetBackgroundColor
SetBorder
SetBorderWidth
SetBorderColor
ButtonLayout
CalcOrderIndex
CharLimit
DefaultValue
Editable
Highlight
IsHidden
IsMultiline
IsPassword
IsReadOnly
IsRequired
IsPassword
IsTerminal
Name
NoViewFlag
PrintFlag
Style
TextFont
TextSize
Type
Value
Fields
Methods
Add
AddDocJavascript
ExecuteThisJavascript
ExportAsFDF
ExportAsHtml
ImportAnFDF
Remove
Count
Item
NewEnum
10 Acrobat DC Search Plug-in ................................................................. 239
   Search plug-in using DDE ................................................................. 239
      Simple query item ........................................................................ 239
      Query item .................................................................................. 239
         Query options ........................................................................... 240
         Query language type constants .................................................. 240
         Word option bit-flag constants .................................................. 241
      Manipulating indexes through DDE ................................................ 241
         Options .................................................................................... 242
            Index operation selectors ....................................................... 242
   Search plug-in using Apple events ....................................................... 242
      SearchAddIndex ............................................................................ 242
      SearchCountIndexList ................................................................... 243
      SearchDoQuery ............................................................................. 243
      SearchGetIndexByPath .................................................................. 245
      SearchGetIndexFlags ..................................................................... 245
      SearchGetIndexList ....................................................................... 246
      SearchGetIndexPath ...................................................................... 246
      SearchGetIndexTitle ..................................................................... 247
      SearchGetNthIndex ....................................................................... 247
      SearchRemoveIndex ....................................................................... 247
      SearchSetIndexFlags ..................................................................... 248
   Search lists ...................................................................................... 248
      Menu names .................................................................................. 248
      Menu item names ........................................................................... 249
      Toolbar button names ................................................................... 249
11 Coordinate Systems ......................................................................... 250
   User space ........................................................................................ 250
   Device space .................................................................................... 250
12 Index .................................................................................................. 252
List of Examples

Example: Viewing a page with Visual Basic ................................................................. 22
Example: Viewing a page with Visual C++ ............................................................... 22
Example: Using AcroPDF browser controls with Visual Basic ............................... 22
Example: CAcroHiliteList class declaration ............................................................ 25
Example: Using the COleDispatchDriver class ...................................................... 26
Example: Displaying “Hello, Acrobat!” in the JavaScript console ......................... 30
Example: Adding an annotation .............................................................................. 32
Example: Spell-checking a document ................................................................. 35
Example: Handling resize events .......................................................................... 37
Example: Setting up a DDE message .................................................................. 40
1 Developing Applications Using IAC

With IAC, an external application can control Acrobat DC or Acrobat DC Reader. For example, you can write an application that launches Acrobat DC, opens a specific file, and sets the page location and zoom factor. You can also manipulate PDF files by, for example, deleting pages or adding annotations and bookmarks. Communication between your application and the Acrobat DC or Acrobat DC Reader application occurs through objects and events.

About the API object layers

You can think of the Acrobat DC API as having two distinct layers that use IAC objects:

- The Acrobat DC application (AV) layer. The AV layer enables you to control how the document is viewed. For example, the view of a document object resides in the layer associated with Acrobat DC.
- The portable document (PD) layer. The PD layer provides access to the information within a document, such as a page. From the PD layer you can perform basic manipulations of PDF documents, such as deleting, moving, or replacing pages, as well as changing annotation attributes. You can also print PDF pages, select text, access manipulated text, and create or delete thumbnails.

You can control the application’s user interface and the appearance of its window by either using its PD layer object, PDPage, or by using its AV layer object, AVDoc. The PDPage object has a method called Draw that exposes the rendering capabilities of Acrobat DC. If you need finer control, you can create your application with the AVDoc object, which has a function called OpenInWindow that can display text annotations and active links in your application’s window.

You can also treat a PDF document as an ActiveX® document and implement convenient PDF browser controls through the AcroPDF object. This object provides you with the ability to load a file, move to various pages within a file, and specify various display and print options. A detailed description of its usage is provided in “Summary of OLE objects and methods” on page 39.

Object reference syntax

The Acrobat DC core API exposes most of its architecture in C, although it is written to simulate an object-oriented system with nearly fifty objects. The IAC interface for OLE automation and Apple events exposes a smaller number of objects. These objects closely map to those in the Acrobat DC API and can be accessed through various programming languages.

DDE does not organize IAC capabilities around objects, but instead uses DDE messages to Acrobat DC.

OLE automation, Apple events, and AppleScript each refer to the objects with a different syntax.

- In OLE, you use the object name in either a Visual Basic or Visual C# CreateObject statement or in an MFC CreateDispatch statement.
- In Apple events, you use the name of the object in a CreateObjSpecifier statement.
- In AppleScript, you use the object name in a set ... to statement.
## Objects in the Acrobat DC application layer

This table describes the IAC objects in the Acrobat DC application (AV) layer. The first three objects are the primary source for controlling the user interface.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>OLE automation class name</th>
<th>Apple event class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVApp</td>
<td>Controls the appearance of Acrobat DC. This is the top-level object, representing Acrobat DC. You can control the appearance of Acrobat DC, determine whether an Acrobat DC window appears, and set the size of the application window. Your application has access to the menu bar and the toolbar through this object.</td>
<td>AcroExch. App</td>
<td>Application</td>
</tr>
<tr>
<td>AVDoc</td>
<td>Represents a window containing an open PDF file. Your application can use this object to cause Acrobat DC to render into a window so that it closely resembles the Acrobat DC window. You can also use this object to select text, find text, or print pages. This object has several bridge methods to access other objects. For more information on bridge methods, see “Summary of OLE objects and methods” on page 39.</td>
<td>AcroExch. AVDoc</td>
<td>Document</td>
</tr>
<tr>
<td>AVPageView</td>
<td>Controls the contents of the AVDoc window. Your application can scroll, magnify, or go to the next, previous, or any arbitrary page. This object also holds the history stack.</td>
<td>AcroExch. AVPageView</td>
<td>PDF Window</td>
</tr>
<tr>
<td>AVMenu</td>
<td>Represents a menu in Acrobat DC. You can count or remove menus. Each menu has a language-independent name used to access it.</td>
<td>None</td>
<td>Menu</td>
</tr>
<tr>
<td>AVMenuItem</td>
<td>Represents a single item in a menu. You can execute or remove menu items. Every menu item has a language-independent name used to access it.</td>
<td>None</td>
<td>Menu item</td>
</tr>
<tr>
<td>AVConversion</td>
<td>Represents the format in which to save the document.</td>
<td>None</td>
<td>conversion</td>
</tr>
</tbody>
</table>
### Objects in the portable document layer

This table describes the IAC objects in the portable document (PD) layer.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>OLE automation class name</th>
<th>Apple event class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDoc</td>
<td>Represents the underlying PDF document. Using this object, your application can perform operations such as deleting and replacing pages. You can also create and delete thumbnails, and set and retrieve document information fields. For OLE automation, the first page of a document is page 0. For Apple events, the first page is page 1.</td>
<td>AcroExch. PDDoc</td>
<td>Document</td>
</tr>
<tr>
<td>PDPage</td>
<td>Represents one page of a PDDoc object. You can use this object to render Acrobat DC to your application’s window. You can also access page size and rotation, set up text regions, and create and access annotations. For OLE automation, the first page of a document is page 0. For Apple events, the first page is page 1.</td>
<td>AcroExch. PDPage</td>
<td>page</td>
</tr>
<tr>
<td>PDAnnot</td>
<td>Manipulates link and text annotations. You can set and query the physical attributes of an annotation and you can perform a link annotation with this object. Apple events have two additional, related objects: PDTextAnnot, a text annotation, and PDLinkAnnot, a link annotation.</td>
<td>AcroExch. PDAnnot</td>
<td>annotation</td>
</tr>
<tr>
<td>PDBookmark</td>
<td>Represents bookmarks in the PDF document. You cannot directly create a bookmark, but if you know a bookmark's title, you can change its title or delete it.</td>
<td>AcroExch. PDBookmark</td>
<td>bookmark</td>
</tr>
<tr>
<td>PDTTextSelect</td>
<td>Causes text to appear selected. If selected text exists within an AVDoc object, your application can also access the words in that region through this object.</td>
<td>AcroExch. PDTTextSelect</td>
<td>None</td>
</tr>
</tbody>
</table>
Plugins for extending the IAC interfaces

You can extend the functionality of the IAC interfaces by writing Plugins that use core API objects that are not already part of the IAC support system. The following graphic shows the software architecture needed to establish a connection. The plug-in calls methods through host function tables (HFTs).

**Using Plugins for interapplication communication**

Similarly, the `JSObject` interface provides you with convenient access to the Acrobat DC features made available through JavaScript. Take advantage of this interface wherever possible. Its usage is explained in “Using the JSObject interface” on page 29.

**Tip:** Your plugins should make use of a broker to work correctly when protected mode is enabled.

Developing for Acrobat Reader

On Windows, the only OLE automation supported for Reader is the *PDF browser controls* interface, which enables you to treat a PDF document as an ActiveX document within an external application. This makes it possible to load a file, move to various pages within the file, highlight a text selection, and specify various print and display options, as shown below.
PDF browser controls are available through the AxAcroPDFLib.AxAcroPDF interface, which provides the following methods used to programmatically control the PDF document window:

- GoBackwardStack
- GoForwardStack
- GotoFirstPage
- GotoLastPage
- GotoNextPage
- GotoPreviousPage
- LoadFile
- Print
- PrintAll
- PrintAllFit
- PrintPages
- PrintPagesFit
- PrintWithDialog
- SetCurrentHighlight
- SetCurrentPage
- SetLayoutMode
- SetNamedDest
- SetPageMode
- SetShowScrollbars
- SetShowToolbar
- SetView
- SetViewRect
- SetViewScroll
- SetZoom
- SetZoomScroll
DDE messages

Adobe Reader supports the following DDE messages:

- AppExit
- CloseAllDocs
- DocClose
- DocGoTo
- DocGoToNameDest
- DocOpen
- FileOpen
- FileOpenEx
- FilePrint
- FilePrintEx
- FilePrintSilent
- FilePrintSilentEx
- FilePrintTo
- FilePrintToEx

Apple events

On Mac OS, you may use Apple events and AppleScript. Adobe Reader supports only the following required Apple events:

- open
- print
- quit
- run
This chapter describes how you can use OLE 2.0 support in Adobe Acrobat DC for Microsoft Windows. Acrobat DC applications are OLE servers and also respond to a variety of OLE automation messages.

Since Acrobat DC provides the appropriate interfaces to be an OLE server, you can embed PDF documents into documents created by an application that is an OLE client, or link them to OLE containers. However, Acrobat DC does not perform in-place activation.

Acrobat DC supports the OLE automation methods that are summarized in this chapter and described fully in the IAC API Reference. Acrobat DC Reader does not support OLE automation, except for the PDF browser controls provided in the AcroPDF object.

The best practical resources for Visual Basic or Visual C# programmers, besides the object browser, are the sample projects. The samples demonstrate use of the Acrobat DC OLE objects and contain comments describing the parameters for the more complicated methods. For more information, see the Acrobat SDK Samples Guide.

OLE capabilities in Acrobat DC

For OLE automation, Acrobat DC provides three capabilities: rendering PDF documents, remotely controlling the application, and implementing PDF browser controls.

On-screen rendering

You can render PDF documents on the screen in two ways:

- Use an interface similar to the Acrobat DC user interface.
  
  In this approach, use the `AVDoc` object’s `OpenInWindowEx` method to open a PDF file in your application’s window. The window has vertical and horizontal scroll bars, and has buttons on the window’s perimeter for setting the zoom factor. Users interacting with this type of window find its operation similar to that of working in Acrobat DC. For example, links are active and the window can display any text annotation on a page.
  
  The `ActiveView` sample in the Acrobat SDK Samples Guide demonstrates this approach.

- Use the `PDPage` object’s `DrawEx` method.
  
  In this approach, you provide a window and a device context, as well as a zoom factor. Acrobat DC renders the current page into your window. The application must manage the scroll bars and other items in the user interface.
  
  The `StaticView` sample in the Acrobat SDK Samples Guide demonstrates this approach.

Remote control of Acrobat DC

You can control Acrobat DC remotely in two ways:

- Given the exported interfaces, you can write an application that manipulates various aspects of PDF documents, such as pages, annotations, and bookmarks. Your application might use `AVDoc`, `PDDoc`, `PDPage`, and `annotation` methods, and might not provide any visual feedback that requires rendering into its application window.

- You can launch Acrobat DC from your own application, which has set up the environment for the user. Your application can cause Acrobat DC to open a file, set the page location and zoom factor, and possibly even select some text. For example, this could be useful as part of a help system.

PDF browser controls

You can use the `AcroPDF` library to display a PDF document in applications using simplified browser controls. In this case, the PDF document is treated as an ActiveX document, and the interface is available in Acrobat DC Reader.

Load the document with the `AcroPDF` object’s `LoadFile` method. You can then implement browser controls for the following functionality:

- To determine which page to display
- To choose the display, view, and zoom modes
- To display bookmarks, thumbs, scrollbars, and toolbars
- To print pages using various options
- To highlight a text selection
Development environment considerations

You have a choice of environments in which to integrate with Acrobat DC: Visual Basic, Visual C#, and Visual C++.

If possible, use Visual Basic or Visual C#. The run-time type checking offered by the `CreateObject` call in Visual Basic allows quick prototyping of an application, and in both of these languages the implementation details are simplified.

For comparison, consider the following examples, in which you can see strings with "AcroExch.App" and strings with "Acrobat.CAcroApp". The first is the form for the external string used by OLE clients to create an object of that type. The second is the form that is included in developer type libraries.

This example shows a Visual Basic subroutine to view a given page of an open document:

**Example: Viewing a page with Visual Basic**

Private Sub myGoto(ByVal where As Integer)
    Dim app as Object, avdoc as Object, pageview as Object

    Set app = CreateObject("AcroExch.App")
    Set avdoc = app.GetActiveDoc
    Set pageview = avdoc.GetAVPageView
    pageview.Goto(where)
End Sub

The following example does the same, but in Visual C++:

**Example: Viewing a page with Visual C++**

```c++
void goto(int where)
{
    CAcroApp app;
    CAcroAVDoc *avdoc = new CAcroAVDoc;
    CAcroAVPageView pageview;
    COleException e;
    app.CreateDispatch("AcroExch.App");
    avdoc->AttachDispatch(app.GetActiveDoc, TRUE);
    pageview->AttachDispatch(avdoc->GetAVPageView, TRUE);
    pageview->Goto(where);
}
```

The next example shows how to use PDF browser controls to view a page in Visual Basic:

**Example: Using AcroPDF browser controls with Visual Basic**

```vbnet
Friend WithEvents AxAcroPDF1 As AxAcroPDFLib.AxAcroPDF
Me.AxAcroPDF1 = New AxAcroPDFLib.AxAcroPDF
    'AxAcroPDF1
Me.AxAcroPDF1.Enabled = True
Me.AxAcroPDF1.Location = New System.Drawing.Point(24, 40)
Me.AxAcroPDF1.Name = "AxAcroPDF1"
Me.AxAcroPDF1.OcxState = CType(resources.GetObject("AxAcroPDF1.OcxState"),
```
Me.AxAcroPDF1.Size = New System.Drawing.Size(584, 600)
Me.AxAcroPDF1.TabIndex = 0
AxAcroPDF1.LoadFile("http://www.example.com/example.pdf")
AxAcroPDF1.setCurrentPage(TextBox2.Text)

The Visual Basic examples are simpler to read, write, and support, and the implementation details are similar to Visual C#.

In Visual C++, the CAcro classes hide much of the type checking that must be done. Using OLE automation objects in Visual C++ requires an understanding of the AttachDispatch and CreateDispatch methods of the COleDispatchDriver class. For more information, see “Using the Acrobat DC OLE interfaces” on page 25.

**Note:** The header files containing the values of constants that are required by C and C++ programmers to use OLE automation are located in the Acrobat DC SDK IAC directory. Visual Basic and Visual C# users do not need these header files, though it may be useful to refer to them in order to verify the constant definitions.

---

**Environment configuration**

The only requirement for using the OLE objects made available by Acrobat DC is to have the product installed on your system and the appropriate type library file included in the project references for your project. The Acrobat DC type library file is named Acrobat.tlb. This file is included in the InterAppCommunicationSupport\Headers folder in the SDK. Once you have the type library file included in your project, you can use the object browser to browse the OLE objects.

It is not sufficient to install just an ActiveX control or DLL to enable OLE automation. You must have the full Acrobat DC product installed.

If you are a Visual Basic programmer, it is helpful to include the iac.bas module in your project (included in the headers folder). This module defines the constant variables.
Necessary C knowledge

These documents, as well as the API, were designed with C programming in mind and programming with the API requires some familiarity with C concepts.

Although you do not need the header files provided in the SDK, you can use them to find the values of various constants, such as `AV_DOC_VIEW`, that are referenced in the documentation. The file `iac.h` contains most of these values.

Some of the methods, such as `OpenInWindowEx`, can be initially confusing when used in Visual Basic. `OpenInWindowEx` takes a `long` for the `openflags` parameter. The options for this parameter, as provided in the `IAC API Reference`, are:

- `AV_EXTERNAL_VIEW` — Open the document with the toolbar visible.
- `AV_DOC_VIEW` — Draw the page pane and scrollbars.
- `AV_PAGE_VIEW` — Draw only the page pane.

If you were developing in C, these strings would be replaced by a numeric value prior to compilation; passing these strings to the method would not raise an error. When programming in Visual Basic, these strings correspond to constant variables defined in `iac.bas`.

In some situations, you need to apply a bitwise `OR` to multiple values and pass the resultant value to a method. For example, in `iac.h` the `ntype` parameter of the `PDDocSave` method is a bitwise `OR` of the following flags:

```c
/* PDSaveFlags — used for PD-level Save
** All undefined flags should be set to zero.
** If either PDSaveCollectGarbage or PDSaveCopy are used, PDSaveFull must be used. */
typedef enum {
    PDSaveIncremental = 0x0000, /* write changes only */
    PDSaveFull = 0x0001, /* write entire file */
    PDSaveCopy = 0x0002, /* write copy w/o affecting current state */
    PDSaveLinearized = 0x0004, /* write the file linearized for
     ** page-served remote (net) access. */
    PDSaveBinaryOK = 0x0010, /* OK to store binary in file */
    PDSaveCollectGarbage = 0x0020 /* perform garbage collection on
     ** unreferenced objects */
} PDSaveFlags;
```

For example, if you would like to fully save the PDF file and optimize it for the Web (linearize it) within a Visual Basic application, pass `PDSaveFull + PDSaveLinearized` (both defined in `iac.bas`) into the `ntype` parameter; this is the equivalent of a binary `OR` of the `PDSaveFull` and `PDSaveLinearized` parameters.

In many instances, the numeric values are spelled out in comments in the Visual Basic sample code. However, knowledge of why the methods are structured in this way and how they are used in C can be useful to Visual Basic and Visual C# programmers.
Using the Acrobat DC OLE interfaces

This section describes using the CAcro classes and the COleDispatchDriver class. The CAcro classes are subclasses of COleDispatchDriver.

About the CAcro classes

OLE 2.0 support in Acrobat DC includes several classes whose names begin with "CAcro", such as CAcroApp and CAcroPDDoc. Several files in the SDK encapsulate the definitions of these classes.

The CAcro classes are defined in the Acrobat DC type library acrobat.tlb. The OLEView tool in Visual Studio allows you to browse registered type libraries. Use acrobat.tlb when defining OLE automation for a project in Microsoft Visual C++. The files acrobat.h and acrobat.cpp are included in the Acrobat DC SDK, and implement a type-safe wrapper to the Acrobat DC automation server.

Note: Do not modify the acrobat.tlb, acrobat.h, and acrobat.cpp files in the SDK; these define Acrobat DC's OLE automation interface.

The CAcro classes inherit from the MFC COleDispatchDriver class. Understanding this class makes it easier to write applications that use the CAcro classes and their methods.

See the Interapplication Communication API Reference for details on the CAcro classes and their methods.

About the COleDispatchDriver class

The COleDispatchDriver class implements the client side of OLE automation, providing most of the code needed to access automation objects. It provides the wrapper functions AttachDispatch, DetachDispatch, and ReleaseDispatch, as well as the convenience functions InvokeHelper, SetProperty, and GetProperty. You employ some of these methods when you use the Acrobat DC-provided automation objects. Other methods are used in the Acrobat DC implementation of these objects.

COleDispatchDriver is essentially a “class wrapper” for IDispatch, which is the OLE interface by which applications expose methods and properties so that other applications written in Visual Basic and Visual C# can use the application’s features. This provides OLE support for Acrobat DC applications.

Using COleDispatchDriver objects and methods

This section discusses how to use the classes exported by acrobat.cpp, and shows when to call the CreateDispatch and AttachDispatch methods.

The following is a section of code from acrobat.h that declares the CAcroHiliteList class. CAcroHiliteList is a subclass of the COleDispatchDriver class, which means that it shares all the instance variables of COleDispatchDriver.

One of these variables is m_lpDispatch, which holds an LPDISPATCH for that object. An LPDISPATCH is a long pointer to an IDispatch, which can be considered an opaque data type representing a dispatch connection. m_lpDispatch can be used in functions that require an LPDISPATCH argument.

Example: CAcroHiliteList class declaration

class CAcroHiliteList : public COleDispatchDriver
{

### public:

```cpp
CAcroHiliteList() {}  // Calls COleDispatchDriver default constructor
CAcroHiliteList(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}  
CAcroHiliteList(const CAcroHiliteList& dispatchSrc) :  
    COleDispatchDriver(dispatchSrc) {} 
```

// Attributes

// Operations

```cpp
public:

// Attributes

public:

// Operations

bool Add(short nOffset, short nLength);
```

---

The following is the related implementation section of the `Add` method from `acrobat.cpp`:

```cpp
bool CAcroHiliteList::Add(short nOffset, short nLength)
{
    bool result;
    static BYTE parms[] =
        VTS_I2 VTS_I2;
    InvokeHelper(0x1, DISPATCH_METHOD, VT_I4, (void*)&result, parms,  
        nOffset, nLength);
    return result;
}
```

When the `Add` method is called, such as with this code from the following example using the `COleDispatchDriver class`

```cpp
hilite->Add(0, 10);
```

the `InvokeHelper` function is called. This `COleDispatchDriver` method takes a variable number of arguments. It eventually calls the Acrobat DC implementation for `CAcroHiliteList` object’s `Add` method. This happens across the virtual OLE “wires” and takes care of all the OLE details. The end result is that a page range is added to the `CAcroHiliteList` object.

The following is an implementation of a method adapted from the `ActiveView` sample:

#### Example: Using the `COleDispatchDriver class`

```cpp
// This code demonstrates how to highlight words with either a word or page highlight list
void CActiveViewDoc::OnToolsHilitewords()
{
    CAcroAVPageView pageView;
    CAcroPDPage page;
    CAcroPDTextSelect* textSelect = new CAcroPDTextSelect;
    CAcroHiliteList* hilite = new CAcroHiliteList;
    char buf[255];
    long selectionSize;

    if (((BOOL) GetCurrentPageNum() > PDBeforeFirstPage) {  
        // Obtain the AVPageView
        pageView.AttachDispatch(m_pAcroAVDoc->GetAVPageView(),TRUE);
        // Create the Hilite list object
```
hilite->CreateDispatch("AcroExch.HiliteList");
if (hilite) {

    // Add the first 10 words or characters of that page to the highlight list
    hilite->Add(0,10);
    page.AttachDispatch(pageView.GetPage(), TRUE);

    // Create text selection for either page or word highlight list
    textSelect->AttachDispatch(page.CreateWordHilite(hilite->m_lpDispatch));
    m_pAcroAVDoc->SetTextSelection(textSelect->m_lpDispatch);
    m_pAcroAVDoc->ShowTextSelect();

    // Extract the number of words and the first word of text selection
    selectionSize = textSelect->GetNumText();
    if (selectionSize)
        sprintf (buf, "# of words in text selection: %ld
1st word in text
    selection = '%s'", selectionSize, textSelect->GetText(0));
    else
        sprintf (buf, "Failed to create text selection.");

    AfxMessageBox(buf);
}
}

delete textSelect;
delete hilite;
}

In the preceding example, the objects with the prefix CAcro are all CAcro class objects—and they are also COleDispatchDriver objects—because all the Acrobat DC CAcro classes are subclasses of COleDispatchDriver.

Instantiating a class is not sufficient to use it. Before you use an object, you must attach your object to the appropriate Acrobat DC object by using one of the Dispatch methods of the COleDispatchDriver class. These functions also initialize the m_lpDispatch instance variable for the object.

This code from the previous example shows how to attach an IDispatch that already exists:
CAcroAVPageView pageView;
// Obtain the AVPageView
pageView.AttachDispatch(m_pAcroAVDoc->GetAVPageView(), TRUE);

The GetAVPageView method of the CAcroAVDoc class returns an LPDISPATCH, which is what the AttachDispatch method is expecting for its first argument. The BOOL passed as the second argument indicates whether or not the IDispatch should be released when the object goes out of scope, and is typically TRUE. In general, when an LPDISPATCH is returned from a method such as GetAVPageView, you use AttachDispatch to attach it to an object.

The following code from the previous example uses the CreateDispatch method:
CAcroHiliteList *hilite = new CAcroHiliteList;
hilite->CreateDispatch("AcroExch.HiliteList");
hilite->Add(0, 10);
In this case, the `CreateDispatch` method both creates the `IDispatch` object and attaches it to the object. This code works fine; however, the following code would fail:

```c++
CAcroHiliteList *hilite = new CAcroHiliteList;
hilite->Add(0, 10);
```

This error is analogous to using an uninitialized variable. Until the `IDispatch` object is attached to the `COleDispatchDriver` object, it is not valid.

`CreateDispatch` takes a string parameter, such as "AcroExch.HiliteList", which represents a class. The following code is incorrect:

```c++
CAcroPDDoc doc = new CAcroPDDoc;
doc.CreateDispatch("AcroExch.Create");
```

This fails because Acrobat DC won’t respond to such a parameter. The parameter should be "AcroExch.PDDoc" instead.

The valid strings for `CreateDispatch` are as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAcroPoint</td>
<td>&quot;AcroExch.Point&quot;</td>
</tr>
<tr>
<td>CAcroRect</td>
<td>&quot;AcroExch.Rect&quot;</td>
</tr>
<tr>
<td>CAcroTime</td>
<td>&quot;AcroExch.Time&quot;</td>
</tr>
<tr>
<td>CAcroApp</td>
<td>&quot;AcroExch.App&quot;</td>
</tr>
<tr>
<td>CAcroPDDoc</td>
<td>&quot;AcroExch.PDDoc&quot;</td>
</tr>
<tr>
<td>CAcroAVDoc</td>
<td>&quot;AcroExch.AVDoc&quot;</td>
</tr>
<tr>
<td>CAcroHiliteList</td>
<td>&quot;AcroExch.HiliteList&quot;</td>
</tr>
<tr>
<td>CAcroPDBBookmark</td>
<td>&quot;AcroExch.PDBBookmark&quot;</td>
</tr>
<tr>
<td>CAcroMatrix</td>
<td>&quot;AcroExch.Matrix&quot;</td>
</tr>
<tr>
<td>AcroPDF</td>
<td>&quot;AxAcroPDFLib.AxAcroPDF&quot;</td>
</tr>
</tbody>
</table>

Refer again to this code from the previous example:

```c++
CAcroPDPage page;
page.AttachDispatch(pageView.GetPage(), TRUE);
```

A `PDPage` object is required because the purpose of this code is to highlight words on the current page. Since it is a `CAcro` variable, it is necessary to attach to the OLE object before using its methods.

`CreateDispatch` cannot be used to create a `PDPage` object because "AcroExch.PDPage" is not a valid string for `CreateDispatch`. However, the `AVPageView` method `GetPage` returns an `LPDISPATCH` pointer for a `PDPage` object. This is passed as the first argument to the `AttachDispatch` method of the page object. The `TRUE` argument indicates that the object is to be released automatically when it goes out of scope.

```c++
CAcroPDTextSelect* textSelect = new CAcroPDTextSelect;
textSelect->AttachDispatch(page.CreateWordHilite(hilite->m_lpDispatch));
```
m_pAcroAVDoc->SetTextSelection (textSelect->m_lpDispatch);
m_pAcroAVDoc->ShowTextSelect();

This code performs the following steps:

1. Declares a text selection object textSelect.

2. Calls the CAcroPDPage method CreateWordHilite, which returns an LPDISPATCH for a PDTextSelect. CreateWordHilite takes an LPDISPATCH argument representing a CAcroHilite list. The hilite variable already contains a CAcroHiliteList object, and its instance variable m_lpDispatch contains the LPDISPATCH pointer for the object.

3. Calls the CAcroAVDoc object's SetTextSelection method to select the first ten words on the current page.

4. Calls the AcroAVDoc's ShowTextSelect method to cause the visual update on the screen.

Using the JSObject interface

Acrobat DC provides a rich set of JavaScript programming interfaces that can be used from within the Acrobat DC environment. It also provides the JSObject interface, which allows external clients to access the same functionality from environments such as Visual Basic.

In precise terms, JSObject is an interpretation layer between an OLE automation client, such as a Visual Basic application, and the JavaScript functionality provided by Acrobat DC. From a developer's point of view, programming JSObject in a Visual Basic environment is similar to programming in JavaScript using the Acrobat DC console.

This section explains how to extend Acrobat DC using JavaScript in a Visual Basic programming environment. It provides a set of examples to illustrate the key concepts.

Whenever possible, you should take advantage of these capabilities by using the JSObject interface available within the AcroExch.PDDoc object. To obtain the interface, invoke the object's GetJSObject method.

Adding a reference to the Acrobat DC type library

This procedure adds a reference to the Acrobat DC type library so that you can access the Acrobat DC automation APIs, including JSObject, in Visual Basic. Do this before using the JSObject interface, as in the examples that follow.

To add a reference to the Acrobat DC type library:

1. Install Acrobat DC and Visual Basic.

2. Create a new Visual Basic project from the Windows Application template. This provides a blank form and project workspace.

3. Select Project > Add Reference and click the COM tab.

4. From the list of available references, select Adobe Acrobat <version> Type Library and click OK.
Creating a simple application

This example provides the minimum code to display “Hello, Acrobat!” in the Acrobat DC JavaScript console.

To set up and run the “Hello, Acrobat!” example:

1. Open the source code window for the default form by clicking View > Code.

2. Select (Form1 Events) from the selection box in the upper left corner of that window.
   The selection box in the upper right corner now shows all the functions available to the Form1 object.

3. Select Load from the functions selection box. This creates an empty function stub. The Form1 Load function is called when Form1 is first displayed, so this is a good place to add the initialization code.

4. Add the following code to define some global variables before the subroutine.
   
   ```vbc
   Dim gApp As Acrobat.CAcroApp
   Dim gPDDoc As Acrobat.CAcroPDDoc
   Dim jso As Object
   ```

5. Add the following code to the private Form1_Load subroutine.
   
   ```vbc
   gApp = CreateObject("AcroExch.App")
   gPDDoc = CreateObject("AcroExch.PDDoc")
   If gPDDoc.Open("c:\example.pdf") Then
       jso = gPDDoc.GetJSObject
       jso.console.Show
       jso.console.Clear
       jso.console.println ("Hello, Acrobat!")
       gApp.Show
   End If
   ```

6. Create a file called example.pdf at the root level of the C: drive.

7. Save and run the project.

When you run the application, Acrobat DC is launched, Form1 is displayed, and the JavaScript Debugger window is opened, displaying “Hello, Acrobat!”.

**Example: Displaying “Hello, Acrobat!” in the JavaScript console**

```vbc
Dim gApp As Acrobat.CAcroApp
Dim gPDDoc As Acrobat.CAcroPDDoc
Dim jso As Object

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Me.Load
    gApp = CreateObject("AcroExch.App")
    gPDDoc = CreateObject("AcroExch.PDDoc")
    If gPDDoc.Open("c:\example.pdf") Then
        jso = gPDDoc.GetJSObject
        jso.console.Show
        jso.console.Clear
        jso.console.println ("Hello, Acrobat!")
        gApp.Show
    End If
```
End Sub

The Visual Basic program attaches to the Acrobat DC automation interface using the `CreateObject` call, and then shows the main window using the `App` object’s `Show` command.

You may have a few questions after studying the code. For example, why is `jso` declared as an Object, while `gApp` and `gPDDoc` are declared as types found in the Acrobat DC type library? Is there a real type for `JSObject`?

The answer is no, `JSObject` does not appear in the type library, except in the context of the `CAcroPDDoc.GetJSObject` call. The COM interface used to export JavaScript functionality through `JSObject` is known as an IDispatch interface, which in Visual Basic is more commonly known simply as an “Object” type. This means that the methods available to the programmer are not particularly well-defined. For example, if you replace the call to

```vbnet
jso.console.clear
```

with

```vbnet
jso.ThisCantPossiblyCompileCanIt("Yes it can!")
```

the compiler compiles the code, but fails at run time. Visual Basic has no type information for `JSObject`, so Visual Basic does not know if a particular call is syntactically valid until run-time, and will compile any function call to a `JSObject`. For that reason, you must rely on the documentation to know what functionality is available through the `JSObject` interface. For details, see the JavaScript for Acrobat API Reference.

You may also wonder why it is necessary to open a `PDDoc` before creating a `JSObject`. Running the program shows that no document appears onscreen, and suggests that using the JavaScript console should be possible without a `PDDoc`. However, `JSObject` is designed to work closely with a particular document, as most of the available features operate at the document level. There are some application-level features in JavaScript (and therefore in `JSObject`), but they are of secondary interest. In practice, a `JSObject` is always associated with a particular document.

When working with a large number of documents, you must structure your code so that a new `JSObject` is acquired for each document, rather than creating a single `JSObject` to work on every document.

### Working with annotations

This example uses the `JSObject` interface to open a PDF file, add a predefined annotation to it, and save the file back to disk.

**To set up and run the annotations example:**

1. Create a new Visual Basic project and add the Adobe Acrobat DC type library to the project.
2. From the Toolbox, drag the `OpenFileDialog` control to the form.
3. Drag a **Button** to your form.

![Form and Toolbox](image)

4. Select **View > Code** and set up the following source code:

**Example: Adding an annotation**

```vbnet
Dim gApp As Acrobat.CAcroApp

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
gApp = CreateObject("AcroExch.App")
End Sub

Private Sub Form1_Closed(Cancel As Integer)
If Not gApp Is Nothing Then
gApp.Exit
End If
gApp = Nothing
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim pdDoc As Acrobat.CAcroPDDoc
Dim page As Acrobat.CAcroPDPage
Dim jso As Object
Dim path As String
Dim point(1) As Integer
Dim popupRect(3) As Integer
Dim pageRect As Object
Dim annot As Object
Dim props As Object

OpenFileDialog1.ShowDialog()
```

path = OpenFileDialog1.FileName

pdDoc = CreateObject("AcroExch.PDDoc")
If pdDoc.Open(path) Then
    jso = pdDoc.GetJSObject
    If Not jso Is Nothing Then

        ' Get size for page 0 and set up arrays
        page = pdDoc.AcquirePage(0)
        pageRect = page.GetSize
        point(0) = 0
        point(1) = pageRect.y
        popupRect(0) = 0
        popupRect(1) = pageRect.y - 100
        popupRect(2) = 200
        popupRect(3) = pageRect.y

        ' Create a new text annot
        annot = jso.AddAnnot
        props = annot.getProps
        props.Type = "Text"
        annot.setProps props

        ' Fill in a few fields
        props = annot.getProps
        props.page = 0
        props.point = point
        props.popupRect = popupRect
        props.author = "John Doe"
        props.noteIcon = "Comment"
        props.strokeColor = jso.Color.red
        props.Contents = "I added this comment from Visual Basic!"
        annot.setProps props
    End If
    pdDoc.Close
    MsgBox "Annotation added to " & path
Else
    MsgBox "Failed to open " & path
End If

pdDoc = Nothing
End Sub

5. Save and run the application.

The code in the Form_Load and Form_Closed routines initializes and shuts down the Acrobat DC automation interface. More interesting work happens in the Command button's click routine. The first lines declare local variables and show the Windows Open dialog box, which allows the user to select a file to be annotated. The code then opens the PDF file's PDDoc object and obtains a JSObject interface to that document.

Some standard Acrobat DC automation methods are used to determine the size of the first page in the document. These numbers are critical to achieving the correct layout, because the PDF coordinate system is based in the lower-left corner of the page, but the annotation will be anchored at the upper left corner of the page.
The lines following the "Create a new text annot" comment do exactly that, but this block of code bears additional explanation.

First, `addAnnot` looks as if it is a method of `JSObject`, but the JavaScript reference shows that the method is associated with the `doc` object. You might expect the syntax to be `jso.doc.addAnnot`. However, `jso` is the `Doc` object, so `jso.addAnnot` is correct. All of the properties and methods in the `Doc` object are used in this manner.

Second, observe the use of `annot.getProps` and `annot.setProps`. The `Annot` object is implemented with a separate properties object, meaning that you cannot set the properties directly. For example, you cannot do the following:

```vba
annot = jso.AddAnnot
anot.Type = "Text"
anot.page = 0
...
```

Instead, you must obtain the properties object of `Annot` using `annot.getProps`, and use that object for read or write access. To save changes back to the original `Annot`, call `annot.setProps` with the modified properties object.

Third, note the use of `JSObject`'s `color` property. This object defines several simple colors such as red, green, and blue. In working with colors, you may need a greater range of colors than is available through this object. Also, there is a performance hit associated with every call to `JSObject`. To set colors more efficiently, you can use code such as the following, which sets the annot's `strokeColor` to red directly, bypassing the color object.

```vba
dim color(0 to 3) as Variant
color(0) = "RGB"
color(1) = 1#
color(2) = 0#
color(3) = 0#
anot.strokeColor = color
```

You can use this technique anywhere a color array is needed as a parameter to a `JSObject` routine. The example sets the colorspace to RGB and specifies floating point values ranging from 0 to 1 for red, green, and blue. Note the use of the `#` character following the color values. These are required, since they tell Visual Basic that the array element should be set to a floating point value, rather than an integer. It is also important to declare the array as containing `Variant`s, because it contains both strings and floating point values. The other color spaces ("T", "G", "CMYK") have varying requirements for array length. For more information, refer to the `Color` object in the JavaScript for Acrobat API Reference.

**Note:** If you want users to be able to edit annotations, set the JavaScript property `Collab.showAnnotsToolsWhenNoCollab` to `true`.

### Spell-checking a document

Acrobat DC includes a plug-in that can scan a document for spelling errors. The plug-in also provides JavaScript methods that can be accessed using `JSObject`. In this example, you start with the source code from the example Adding an annotation and make the following changes:

- Add a List View control to the main form. Keep the default name `ListView1` for the control.
- Replace the code in the existing `Command1_Click` routine with the following:
Example: Spell-checking a document

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim pdDoc As Acrobat.CAcroPDDoc
Dim jso As Object
Dim path As String
Dim count As Integer
Dim i As Integer, j As Integer
Dim word As Variant
Dim result As Variant
Dim foundErr As Boolean

OpenFileDialog1.ShowDialog()
path = OpenFileDialog1.FileName
foundErr = False
pdDoc = CreateObject("AcroExch.PDDoc")

If pdDoc.Open(path) Then
    jso = pdDoc.GetJSObject
    If Not jso Is Nothing Then
        count = jso.getPageNumWords(0)
        For i = 0 To count - 1
            word = jso.getPageNthWord(0, i)
            If VarType(word) = vbString Then
                result = jso.spell.checkWord(word)
                If IsArray(result) Then
                    foundErr = True
                    ListView1.Items.Add (word & " is misspelled."
                ListView1.Items.Add (“Suggestions:”)
                For j = LBound(result) To UBound(result)
                    ListView1.Items.Add (result(j))
                Next j
                ListView1.Items.Add ("
            End If
        Next i
    End If
    jso = Nothing
    pdDoc.Close
    If Not foundErr Then
        ListView1.Items.Add (“No spelling errors found in " & path)
    End If
Else
    MsgBox "Failed to open " & path
End If

pdDoc = Nothing
End Sub

In this example, note the use of the Spell object’s check method. As described in the JavaScript for Acrobat API Reference, this method takes a word as input, and returns a null object if the word is found in the dictionary, or an array of suggested words if the word is not found.
The safest approach when storing the return value of a `JSObject` method call is to use a Variant. You can use the `IsArray` function to determine if the Variant is an array, and write code to handle that situation accordingly. In this simple example, if the program finds an array of suggested words, it dumps them out to the List View control.

**Tips for translating JavaScript to `JSObject`**

Covering every method available to `JSObject` is beyond the scope of this document. However, the JavaScript for Acrobat API Reference covers the subject in detail, and much can be inferred from the reference by keeping a few basic facts in mind:

- Most of the objects and methods in the reference are available in Visual Basic, but not all. In particular, any JavaScript object that requires the `new` operator for construction cannot be created in Visual Basic. This includes the `Report` object.
- The `Annots` object is unusual in that it requires `JSObject` to set and get its properties as a separate object using the `getProps` and `setProps` methods.
- If you are unsure what type to use to declare a variable, declare it as a Variant. This gives Visual Basic more flexibility for type conversion, and helps prevent runtime errors.
- `JSObject` cannot add new properties, methods, or objects to JavaScript. Due to this limitation, the `global.setPersistent` property is not meaningful.
- `JSObject` is case-insensitive. Visual Basic often capitalizes leading characters of an identifier and prevents you from changing its case. Don’t be concerned about this, since `JSObject` ignores case when matching the identifier to its JavaScript equivalent.
- `JSObject` always returns values as Variants. This includes property gets as well as return values from method calls. An empty Variant is used when a null return value is expected. When `JSObject` returns an array, each element in the array is a Variant. To determine the actual data type of a Variant, use the utility functions `IsArray`, `IsNumeric`, `IsEmpty`, `IsObject`, and `VarType` from the Information module of the Visual Basic for Applications (VBA) library.
- `JSObject` can process most elemental Visual Basic types for setting properties and for and input parameters for method calls, including Variant, Array, Boolean, String, Date, Double, Long, Integer, and Byte. `JSObject` can accept Object parameters, but only when the Object is the result of a property get or method call to a `JSObject`. `JSObject` fails to accept values of type Error and Currency.
Other development topics

This section contains a variety of topics related to developing OLE applications.

Synchronous messaging

The Acrobat DC OLE automation implementation is based on a synchronous messaging scheme. When an application sends a request to Acrobat DC, the application processes that request and returns control to the application. Only then can the application send Acrobat DC another message. If your application sends one message followed immediately by another, the second message may not be properly received: instead of generating a server busy error, it fails with no error message.

For example, this can occur with the AVDoc.OpenInWindowEx method, where a large volume of information regarding drawing position and mouse clicks is exchanged, and with the usage of the PDPage.DrawEx method on especially complex pages. With the DrawEx method, the problem arises when a WM_PAINT message is generated. If the page is complex and the environment is multi-threaded, the application may not finish drawing the page before the application generates another WM_PAINT message. Because the application is single-threaded, multi-thread applications must handle this situation appropriately.

MDI applications

Suppose you create a multiple document interface (MDI) application that creates a static window into which Acrobat DC is displayed using the OpenInWindowEx call, and this window is based on the CFormView OLE class. If another window is placed on top of that window and is subsequently removed, the Acrobat DC window does not repaint correctly.

To fix this, assign the Clip Children style to the dialog box template (on which CFormView is based). Otherwise, the dialog box erases the background of all child windows, including the one containing the PDF file, which wipes out the previously covered part of the PDF window.

Event handling in child windows

When a PDF file is opened with OpenInWindowEx, Acrobat DC creates a child window on top of it. This allows the application to receive events for this window directly. However, an application must also handle the following events: resize, key up, and key down.

The following example from the ActiveView sample shows how to handle a resize event:

Example: Handling resize events

```c++
void CActiveViewVw::OnSize(UINT nType, int cx, int cy)
{
    CWnd* pWndChild = GetWindow(GW_CHILD);
    if (!pWndChild)
        return;
    CRect rect;
    GetClientRect(&rect);
    pWndChild->
        SetWindowPos(NULL,0,0,rect.Width,rect.Height,
                    SWP_NOZORDER | SWP_NOMOVE);
```
CView::OnSize(nType, cx, cy);
}

After sending the message to the child window, it also does a resize. This results in both windows being resized, which is the desired effect.

Determining if an Acrobat DC application is running

Use the Windows `FindWindow` method with the Acrobat DC class name. You can use the Microsoft Spy++ utility to determine the class name for the version of the application.

Exiting from an application

When a user exits from an application using OLE automation, Acrobat DC itself or a web browser displaying a PDF document can be affected:

- If no PDF documents are open in Acrobat DC, the application quits.
- If a web browser is displaying a PDF document, the display goes blank. The user can refresh the page to redisplay it.
Summary of OLE objects and methods

OLE automation support is provided by a set of classes in the Acrobat DC API.

The following diagram shows the objects and methods that are used in OLE. The arrows indicate bridge methods, which are methods that can get an object from a related object of a different layer. For example, if you want to get the `PDDoc` associated with a particular `AVDoc` object, you can use the `GetPDDoc` method in the `AcroExch.AVDoc` object.

For complete descriptions, see the OLE automation sections of the IAC API Reference.
Although DDE is supported, you should use OLE automation instead of DDE whenever possible because DDE is not a COM technology.

For complete descriptions of the parameters associated with DDE messages, see the DDE sections of the *IAC API Reference*.

For all DDE messages, the service name is `acroview`, the transaction type is `XTYPE_EXECUTE`, and the topic name is `control`. The data is the command to be executed, enclosed within square brackets. The item argument in the `DdeClientTransaction` call is `NULL`.

The following example sets up a DDE message:

**Example: Setting up a DDE message**

```plaintext
DDE_SERVERNAME = "acroview";
DDE_TOPICNAME = "control";
DDE_ITEMNAME = "[AppHide()]";
```

The square bracket characters in DDE messages are mandatory. DDE messages are case-sensitive and must be used exactly as described.

To be able to use DDE messages on a document, you must first open the document using the `DocOpen` DDE message. You cannot use DDE messages to close a document that a user opened manually.

You can use `NULL` for pathnames, in which case the DDE message operates on the front document.

If more than one command is sent at once, the commands are executed sequentially, and the results appear to the user as a single action. You can use this feature, for example, to open a document to a certain page and zoom level.

Page numbers are zero-based: the first page in a document is page 0. Quotation marks are needed only if a parameter contains white space.

The document manipulation methods, such as those for deleting pages or scrolling, work only on documents that are already open.
You can use several objects and events to develop Acrobat DC applications for Mac OS. Some of the objects and events in the Apple event registry are supported, as well as Acrobat DC-specific objects and events. Acrobat DC supports the following categories of Apple events:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required events</td>
<td>Events that the Finder sends to all applications.</td>
</tr>
<tr>
<td>Core events</td>
<td>Events that are common to a wide variety of applications, though not universally applicable to all applications.</td>
</tr>
<tr>
<td>Acrobat DC-specific events</td>
<td>Events that are specific to Acrobat DC.</td>
</tr>
<tr>
<td>Miscellaneous Apple events</td>
<td>Events that are not in one of the preceding categories.</td>
</tr>
</tbody>
</table>

When programming for Mac OS, use AppleScript with Acrobat DC whenever possible. For Apple events that are not available through AppleScript, handle them with C or other programming languages.

For information on Apple events supported by the Acrobat DC Search plug-in, see the [PDF Library documentation](http://developer.apple.com/documentation/mac/IAC/IAC-2.html).


For more information on the core and required Apple events, see the Apple event registry for Mac OS. This file is in the AppleScript 1.3.4 SDK, which is currently available at [http://developer.apple.com/sdk/](http://developer.apple.com/sdk/).
OLE Automation

This chapter describes the objects, data types, and methods in the OLE automation interface.

The names AcroExch.App and AxAcroPDFLib.AxAcroPDF are the external strings OLE clients use to create objects of certain types. The Acrobat DC developer type libraries call them CAcro.App and AcroPDFLib, respectively.

Acrobat DC supports dual interfaces, so the methods all have a return type of HRESULT.

The following table summarizes the available objects and data types.

| Object                  | Description                                               |
|-------------------------|---------------|-------------------------------------------------|
| AcroExch.App            | The application itself.                                  |
| AcroExch.AVDoc          | A document as seen in the user interface.                 |
| AcroExch.PDDoc          | The underlying PDF representation of a document.         |
| AcroExch.HiliteList     | An entry in a highlight list.                            |
| AcroExch.AVPageView     | The area of the window that displays the contents of a page. |
| AcroExch.PDPage         | A single page in the PDF representation of a document.   |
| AcroExch.PDAannot       | An annotation on a page in the PDF file.                 |
| AcroExch.PDBookmark     | A bookmark in a PDF file.                                |
| AcroExch.PDTextSelect   | A selection of text on a single page.                    |
| AxAcroPDFLib.AxAcroPDF  | An object containing PDF browser controls.                |
| AcroExch.Point          | A point, specified by its x–coordinate and y–coordinate.|
| AcroExch.Rect           | A rectangle, specified by the top-left and bottom-right points. |
| AcroExch.Time           | A specified time, accurate to the millisecond.            |

**AcroExch.App**

The Acrobat DC application itself. This is a creatable interface. From the application layer, you can control the appearance of Acrobat DC, whether Acrobat DC appears, and the size of the application window. This object provides access to the menu bar and the toolbar, as well as the visual representation of a PDF file on the screen (through an AVDoc object).

**Methods**

The App object has the following methods.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseAllDocs</td>
<td>Closes all open documents.</td>
</tr>
<tr>
<td>Exit</td>
<td>Exits Acrobat DC.</td>
</tr>
<tr>
<td>GetActiveDoc</td>
<td>Gets the frontmost document.</td>
</tr>
<tr>
<td>GetActiveTool</td>
<td>Gets the name of the currently active tool.</td>
</tr>
<tr>
<td>GetAVDoc</td>
<td>Gets an AcroExch.AVDoc object via its index within the list of open AVDoc objects.</td>
</tr>
<tr>
<td>GetFrame</td>
<td>Gets the window's frame.</td>
</tr>
<tr>
<td>GetInterface</td>
<td>Gets an IDispatch interface for a named object, typically a third-party plug-in.</td>
</tr>
<tr>
<td>GetLanguage</td>
<td>Gets a code that specifies which language the Acrobat DC application's user interface is using.</td>
</tr>
<tr>
<td>GetNumAVDocs</td>
<td>Gets the number of open AcroExch.AVDoc objects.</td>
</tr>
<tr>
<td>GetPreference</td>
<td>Gets a value from the preferences file.</td>
</tr>
<tr>
<td>GetPreferenceEx</td>
<td>Gets the specified application preference, using the VARIANT type to pass values.</td>
</tr>
<tr>
<td>Hide</td>
<td>Hides the Acrobat DC application.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks the Acrobat DC application.</td>
</tr>
<tr>
<td>Minimize</td>
<td>Minimizes the Acrobat DC application.</td>
</tr>
<tr>
<td>Maximize</td>
<td>Maximizes the Acrobat DC application.</td>
</tr>
<tr>
<td>MenuItemExecute</td>
<td>Executes the menu item whose language-independent menu item name is specified.</td>
</tr>
<tr>
<td>MenuItemIsEnabled</td>
<td>Determines whether the specified menu item is enabled.</td>
</tr>
<tr>
<td>MenuItemIsMarked</td>
<td>Determines whether the specified menu item is marked.</td>
</tr>
<tr>
<td>MenuItemRemove</td>
<td>Removes the menu item whose language-independent menu item is specified.</td>
</tr>
<tr>
<td>Restore</td>
<td>Restores the main window of the Acrobat DC application.</td>
</tr>
<tr>
<td>SetActiveTool</td>
<td>Sets the active tool according to the specified name, and determines whether the tool is to be used only once or should remain active after being used (persistent).</td>
</tr>
<tr>
<td>setFrame</td>
<td>Sets the window's frame to the specified rectangle.</td>
</tr>
<tr>
<td>SetPreference</td>
<td>Sets a value in the preferences file.</td>
</tr>
</tbody>
</table>
CloseAllDocs

Closes all open documents. You can close each individual AVDoc object by calling AVDoc.Close.

You must explicitly close all documents or call App.CloseAllDocs. Otherwise, the process never exits.

Syntax

VARIANT_BOOL CloseAllDocs();

Returns

-1 if successful, 0 if not.

Related methods

AVDoc.Close
AVDoc.Open
AVDoc.OpenInWindow
AVDoc.OpenInWindowEx
PDDoc.Close
PDDoc.Open
PDDoc.OpenAVDoc

Exit

Exits Acrobat DC. Applications should call App.Exit before exiting.

Note: Use App.CloseAllDocs to close all the documents before calling this method.
Syntax

    VARIANT_BOOL Exit();

Returns

Returns -1 if the entire shutdown process succeeded. This includes closing any open documents, releasing OLE references, and finally exiting the application. If any step fails, the function returns 0, and the application continues running. This method does not work if the application is visible (if the user is in control of the application). In such cases, if the Show method had previously been called, you can call Hide and then Exit.

Related methods

App.CloseAllDocs

GetActiveDoc

Gets the frontmost document.

Syntax

    LPDISPATCH GetActiveDoc();

Returns

The LPDISPATCH for the frontmost AcroExch.AVDoc object. If there are no documents open, it returns NULL.

Related methods

App.GetAVDoc

GetActiveTool

Gets the name of the currently active tool.

Syntax

    BSTR GetActiveTool();

Returns

Returns NULL if there is no active tool. Returns the name of the currently active tool otherwise. See the PDF Library documentation for a list of tool names.

Related methods

App.SetActiveTool
GetAVDoc

Gets an AcroExch.AVDoc object from its index within the list of open AVDoc objects. Use App.GetNumAVDocs to determine the number of AcroExch.AVDoc objects.

Syntax

LPDISPATCH GetAVDoc(long nIndex);

Parameters

nIndex         The index of the document to get.

Returns

The LPDISPATCH for the specified AcroExch.AVDoc document, or NULL if nIndex is greater than the number of open documents.

Related methods

App.GetActiveTool

GetFrame

Gets the window’s frame.

GetFrame is not useful when the PDF file was opened with AVDoc.OpenInWindow. GetFrame returns the application window’s frame (not the document window’s frame). However, the application’s window is hidden when a document is opened using OpenInWindow, and does not change in size as document windows are moved and resized.

This method is also not useful if the Acrobat DC application is in single document interface (SDI) mode.

Syntax

LPDISPATCH GetFrame();

Returns

The LPDISPATCH for the window’s frame, specified as an AcroExch.Rect.

If the Acrobat DC application is in SDI mode, a [0,0,0,0] Rect is returned.

Related methods

App.Maximize

App.SetFrame
GetInterface

Gets an IDispatch interface for a named object, typically a third-party plug-in. This is an entry point to functionality that is undefined and which must be provided by the plug-in author. If you are accessing third-party functionality through GetInterface, ask the author for additional information.

Syntax

LPDISPATCH GetInterface (BSTR szName);

Parameters

| szName | Name of the object. |

Returns

The LPDISPATCH for the object's interface or NULL if the object was not found.

GetLanguage

Gets a code that specifies which language the Acrobat DC application's user interface is using.

Syntax

BSTR GetLanguage();

Returns

String containing a three-letter language code. Must be one of the following:

- DEU – German
- ENU – English
- ESP – Spanish
- FRA – French
- ITA – Italian
- NLD – Dutch
- SVE – Swedish

Related methods

App.GetPreference

App.SetPreference
GetNumAVDocs

Gets the number of open Acrobat.AVDoc objects. The maximum number of documents the Acrobat DC application can open at a time is specified by the avpMaxOpenDocuments preference, which can be obtained with App.GetPreferenceEx and set by App.SetPreferenceEx.

Syntax

long GetNumAVDocs();

Returns

The number of open Acrobat.AVDoc objects.

Related methods

App.GetActiveDoc
App.GetAVDoc

GetPreference

Note: This method is deprecated; use GetPreferenceEx instead. GetPreference is unable to accept important data types such as strings, but GetPreferenceEx can convert many data types into acceptable formats.

Gets a value from the preferences file. Zoom values (used in avpDefaultZoomScale and avpMaxPageCacheZoom) are returned as percentages (for example, 1.00 is returned as 100). Colors (used in avpNoteColor -- PDcolorValue) are automatically converted to RGB values from the representation used in the preferences file.

Syntax

long GetPreference(short nType);

Parameters

| nType         | The preferences item whose value is set. For more information, see the PDF Library documentation. |

Returns

The value of the specified preference item.

Related methods

App.GetLanguage
App.SetPreference
GetPreferenceEx

Gets the specified application preference, using the VARIANT type to pass values.

Syntax

VARIANT GetPreferenceEx(short nType);

Parameters

| nType | The name of the preferences item whose value is obtained. |

Returns

The value of the specified preference item.

Related methods

App.GetLanguage

App.SetPreferenceEx

Hide

Hides the Acrobat DC application. When the viewer is hidden, the user has no control over it, and the Acrobat DC application exits when the last automation object is closed.

Syntax

VARIANT_BOOL Hide();

Returns

-1 if successful, 0 if not.

Related methods

App.Show

Lock

Locks the Acrobat DC application. Typically, this method is called when using AVDoc.OpenInWindowEx to draw into another application's window. If you call App.Lock, you should call App.UnlockEx when you are done using OLE automation.

There are some advantages and disadvantages of locking the viewer when using AVDoc.OpenInWindowEx. You must consider these before deciding whether to lock the viewer:

- Locking prevents problems that can sometimes occur if two processes are trying to open a file at the same time.
- Locking prevents a user from using Acrobat DC’s user interface (such as adding annotations) in your application’s window.
- Locking can prevent any other application, including the Acrobat DC application, from opening PDF files. This problem can be minimized by calling `App.UnlockEx` as soon as the file has been opened.

**Syntax**

```vbnet
VARIANT_BOOL Lock(BSTR szLockedBy);
```

**Parameters**

| szLockedBy   | A string that is used as the name of the application that has locked the Acrobat DC application. |

**Returns**

-1 if the Acrobat DC application was locked successfully, 0 otherwise. Locking fails if the Acrobat DC application is visible.

**Related methods**

`App.UnlockEx`

**Minimize**

Minimizes the Acrobat DC application.

**Syntax**

```vbnet
VARIANT_BOOL Minimize(long BMinimize);
```

**Parameters**

| BMinimize   | If a positive number, the Acrobat DC application is minimized. If 0, the Acrobat DC application is returned to its normal state. |

**Returns**

-1 if successful, 0 if not.

**Related methods**

`App.GetFrame`

`App.SetFrame`
Maximize

Maximizes the Acrobat DC application.

Syntax

VARIANT_BOOL Maximize(long bMaximize);

Parameters

| bMaximize | If a positive number, the Acrobat DC application is maximized. If 0, the Acrobat DC application is returned to its normal state. |

Returns

-1 if successful, 0 if not.

Related methods

App.GetFrame
App.SetFrame

MenuItemExecute

Executes the menu item whose language-independent menu item name is specified.

Syntax

VARIANT_BOOL MenuItemExecute(BSTR szMenuItemName);

Parameters

| szMenuItemName | The language-independent name of the menu item to execute. See the PDF Library documentation for a list of menu item names. |

Returns

Returns -1 if the menu item executes successfully, or 0 if the menu item is missing or is not enabled.

Related methods

App.MenuItemIsEnabled
App.MenuItemIsMarked
App.MenuItemRemove
**MenuItemsIsEnabled**

Determines whether the specified menu item is enabled.

**Syntax**

```c
VARIANT_BOOL MenuItemIsEnabled(BSTR szMenuItemName);
```

**Parameters**

- `szMenuItemName` The language-independent name of the menu item whose enabled state is obtained. See the [PDF Library documentation](#) for a list of menu item names.

**Returns**

- `-1` if the menu item is enabled, `0` if it is disabled or does not exist.

**Related methods**

- App.[MenuItemExecute](#)
- App.[MenuItemIsMarked](#)
- App.[MenuItemRemove](#)

**MenuItemsIsMarked**

Determines whether the specified menu item is marked.

**Syntax**

```c
VARIANT_BOOL MenuItemIsMarked(BSTR szMenuItemName);
```

**Parameters**

- `szMenuItemName` The language-independent name of the menu item whose marked state is obtained. See the [PDF Library documentation](#) for a list of menu item names.

**Returns**

- `-1` if the menu item is marked, `0` if it is not marked or does not exist.

**Related methods**

- App.[MenuItemExecute](#)
- App.[MenuItemIsEnabled](#)
- App.[MenuItemRemove](#)
**MenuItemRemove**

Removes the menu item whose language-independent menu item is specified.

**Syntax**

```csharp
VARIANT_BOOL MenuItemRemove(BSTR szMenuItemName);
```

**Parameters**

| szMenuItemName | The language-independent name of the menu item to remove. See the [PDF Library documentation](https://example.com) for a list of menu item names. |

**Returns**

-1 if the menu item was removed, 0 if the menu item does not exist.

**Related methods**

- App.[MenuItemExecute](#)
- App.[MenuItemIsEnabled](#)
- App.[MenuItemIsMarked](#)

**Restore**

Restores the main window of the Acrobat DC application. Calling this with `bRestore` set to a positive number causes the main window to be restored to its original size and position and to become active.

**Syntax**

```csharp
VARIANT_BOOL Restore(long bRestore);
```

**Parameters**

| bRestore | If a positive number, the Acrobat DC application is restored, 0 otherwise. |

**Returns**

-1 if successful, 0 if not.

**Related methods**

- App.[GetFrame](#)
- App.[SetFrame](#)
SetActiveTool

Sets the active tool according to the specified name, and determines whether the tool is to be used only once or should remain active after being used (persistent).

Syntax

```
VARIANT_BOOL SetActiveTool(BSTR szButtonName, 
                          long bPersistent);
```

Parameters

<table>
<thead>
<tr>
<th>szButtonName</th>
<th>The name of the tool to set as the active tool. See the PDF Library documentation for a list of tool names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bPersistent</td>
<td>A request indicating whether the tool should be persistent. A positive number indicates a request to the Acrobat DC application for the tool to remain active after it has been used. If 0 is specified, the Acrobat DC application reverts to the previously active tool after this tool is used once.</td>
</tr>
</tbody>
</table>

Returns

-1 if the tool was set, 0 otherwise.

Related methods

- App.GetActiveTool
- App.ToolButtonIsEnabled
- App.ToolButtonRemove

SetFrame

Sets the window's frame to the specified rectangle. This method has no effect if the Acrobat DC application is in single document interface (SDI) mode.

Syntax

```
VARIANT_BOOL SetFrame(LPDISPATCH iAcroRect);
```

Parameters

| iAcroRect | The LPDISPATCH for an AcroExch.Rect specifying the window frame. iAcroRect contains the instance variable m_lpDispatch, which contains the LPDISPATCH. |

Returns

-1 if the frame was set, 0 if iAcroRect is not of type AcroExch.Rect.
Related methods

- App.GetFrame
- App.Maximize

SetPreference

**Note:** This method is deprecated; use `SetPreferenceEx` instead. `SetPreference` is unable to accept important data types such as strings, but `SetPreferenceEx` can convert many data types into acceptable formats.

Sets a value in the preferences file. Zoom values (used in `avpDefaultZoomScale` and `avpMaxPageCacheZoom`) must be passed as percentages and are automatically converted to fixed point numbers (for example, 100 is automatically converted to 1.0). Colors (used in `avpHighlightColor` or `avpNoteColor`) are automatically converted from RGB values to the representation used in the preferences file.

**Syntax**

```c
VARIANT_BOOL SetPreference(short nType, long nValue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nType</td>
<td>The preferences item whose value is set. See the <a href="#">PDF Library documentation</a> for a list of preference items.</td>
</tr>
<tr>
<td>nValue</td>
<td>The value to set.</td>
</tr>
</tbody>
</table>

**Returns**

-1 if successful, 0 if not.

**Related methods**

- App.GetLanguage
- App.GetPreferenceEx

**SetPreferenceEx**

Sets the application preference specified by `nType` to the value stored at `pVal`. If `pVal` has a non-conforming `VARTYPE`, `SetPreferenceEx` performs type conversion. For example, a string representation of an integer is converted to an actual integer.

**Syntax**

```c
VARIANT_BOOL SetPreferenceEx(short nType, VARIANT* pVal);
```
Parameters

| nType | The preferences item whose value is set. See the PDF Library documentation for a list of preference items. |
| pVal  | The value to set. |

Returns

Returns -1 if nType is a supported type or the type conversion is successful, 0 otherwise.

Related methods

App.GetLanguage

App.GetPreferenceEx

Show

Shows the Acrobat DC application. When the viewer is shown, the user is in control, and the Acrobat DC application does not automatically exit when the last automation object is destroyed. However, it will exit if no documents are being displayed.

Syntax

VARIANT_BOOL Show();

Returns

-1 if successful, 0 if not.

Related methods

App.Hide

ToolButtonIsEnabled

Determines whether the specified toolbar button is enabled.

Syntax

VARIANT_BOOL ToolButtonIsEnabled(BSTR szButtonName);

Parameters

| szButtonName | The name of the button whose enabled state is checked. See the PDF Library documentation for a list of toolbar button names. |
Returns

-1 if the button is enabled, 0 if it is not enabled or does not exist.

Related methods

App.GetActiveTool
App.SetActiveTool
App.ToolButtonRemove

ToolButtonRemove

Removes the specified button from the toolbar.

Syntax

VARIANT_BOOL ToolButtonRemove(BSTR szButtonName);

Parameters

| szButtonName | The name of the button to remove. See the PDF Library documentation for a list of toolbar button names. |

Returns

-1 if the button was removed, 0 otherwise.

Related methods

App.GetActiveTool
App.SetActiveTool
App.ToolButtonIsEnabled

Unlock

Note: In version 4.0 or later, use App.UnlockEx instead.

Unlocks the Acrobat DC application if it was previously locked. This method clears a flag that indicates the viewer is locked. If you called App.Lock, you should call App.Unlock when you are done using OLE automation.

Use App.Lock and App.UnlockEx if you call OpenInWindow.

Typically, you call App.Lock when your application initializes and App.Unlock in your application’s destructor method.
UnlockEx

Unlocks the Acrobat DC application if it was previously locked.

Syntax

VARIANT_BOOL UnlockEx (BSTR szLockedBy);

Parameters

<table>
<thead>
<tr>
<th>szLockedBy</th>
<th>A string indicating the name of the application to be unlocked.</th>
</tr>
</thead>
</table>

Retruns

-1 if successful, 0 if not.

Related methods

- App.Lock
- App.UnlockEx

AcroExch.AVDoc

A view of a PDF document in a window. This is a creatable interface. There is one AVDoc object per displayed document. Unlike a PDDoc object, an AVDoc object has a window associated with it.

Methods

The AVDoc object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BringToFront</td>
<td>Brings the window to the front.</td>
</tr>
<tr>
<td>ClearSelection</td>
<td>Clears the current selection.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Close</td>
<td>Closes a document.</td>
</tr>
<tr>
<td>FindText</td>
<td>Finds the specified text, scrolls so that it is visible, and highlights it.</td>
</tr>
<tr>
<td>GetAVPageView</td>
<td>Gets the <code>AcroExch.AVPageView</code> associated with an <code>AcroExch.AVDoc</code>.</td>
</tr>
<tr>
<td>GetFrame</td>
<td>Gets the rectangle specifying the window's size and location.</td>
</tr>
<tr>
<td>GetPDDoc</td>
<td>Gets the <code>AcroExch.PDDoc</code> associated with an <code>AcroExch.AVDoc</code>.</td>
</tr>
<tr>
<td>GetTitle</td>
<td>Gets the window's title.</td>
</tr>
<tr>
<td>GetViewMode</td>
<td>Gets the current document view mode (pages only, pages and thumbnails, or pages and bookmarks).</td>
</tr>
<tr>
<td>IsValid</td>
<td>Determines whether the <code>AcroExch.AVDoc</code> is still valid.</td>
</tr>
<tr>
<td>Maximize</td>
<td>Maximizes the window if <code>bMaxSize</code> is a positive number.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens a file.</td>
</tr>
<tr>
<td>OpenInWindow</td>
<td>Opens a PDF file and displays it in a user-specified window.</td>
</tr>
<tr>
<td>OpenInWindowEx</td>
<td>Opens a PDF file and displays it in a user-specified window.</td>
</tr>
<tr>
<td>PrintPages</td>
<td>Prints a specified range of pages displaying a print dialog box.</td>
</tr>
<tr>
<td>PrintPagesEx</td>
<td>Prints a specified range of pages, displaying a print dialog box.</td>
</tr>
<tr>
<td>PrintPagesSilent</td>
<td>Prints a specified range of pages without displaying any dialog box.</td>
</tr>
<tr>
<td>PrintPagesSilentEx</td>
<td>Prints a specified range of pages without displaying any dialog box.</td>
</tr>
<tr>
<td>SetFrame</td>
<td>Sets the window's size and location.</td>
</tr>
<tr>
<td>SetTextSelection</td>
<td>Sets the document's selection to the specified text selection.</td>
</tr>
<tr>
<td>setTitle</td>
<td>Sets the window's title.</td>
</tr>
<tr>
<td>SetViewMode</td>
<td>Sets the mode in which the document will be viewed (pages only, pages and thumbnails, or pages and bookmarks)</td>
</tr>
<tr>
<td>ShowTextSelect</td>
<td>Changes the view so that the current text selection is visible.</td>
</tr>
</tbody>
</table>

**BringToFront**

Brings the window to the front.

**Syntax**

```
VARIANT_BOOL BringToFront();
```
Returns

Returns 0 if no document is open, -1 otherwise.

ClearSelection

Clears the current selection.

Syntax

VARIANT_BOOL ClearSelection();

Returns

Returns -1 if the selection was cleared, 0 if no document is open or the selection could not be cleared.

Related methods

AVDoc.SetTextSelection
AVDoc.ShowTextSelect
PDDoc.CreateTextSelect
PDPage.CreatePageHilite
PDPage.CreateWordHilite
PDTextSelect.Destroy
PDTextSelect.GetBoundingRect
PDTextSelect.GetNumText
PDTextSelect.GetPage
PDTextSelect.GetText

Close

Closes a document. You can close all open AVDoc objects by calling App.CloseAllDocs.

To reuse an AVDoc object, close it with AVDoc.Close, then use the AVDoc object's LPDISPATCH for AVDoc.OpenInWindow.

Syntax

VARIANT_BOOL Close(long bNoSave);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bNoSave</td>
<td>If a positive number, the document is closed without saving it. If 0 and the document has been modified, the user is asked whether or not the file should be saved.</td>
</tr>
</tbody>
</table>

Returns

Always returns -1, even if no document is open.

Related methods

App. `CloseAllDocs`
AVDoc. `Open`
AVDoc. `OpenInWindow`
AVDoc. `OpenInWindowEx`
PDDoc. `Close`
PDDoc. `Open`
PDDoc. `OpenAVDoc`

FindText

Finds the specified text, scrolls so that it is visible, and highlights it.

Syntax

```plaintext
VARIANT_BOOL FindText(BSTR szText, long bCaseSensitive, long bWholeWordsOnly, long bReset);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szText</td>
<td>The text to be found.</td>
</tr>
<tr>
<td>bCaseSensitive</td>
<td>If a positive number, the search is case-sensitive. If 0, it is case-insensitive.</td>
</tr>
<tr>
<td>bWholeWordsOnly</td>
<td>If a positive number, the search matches only whole words. If 0, it matches partial words.</td>
</tr>
<tr>
<td>bReset</td>
<td>If a positive number, the search begins on the first page of the document. If 0, it begins on the current page.</td>
</tr>
</tbody>
</table>

Returns

-1 if the text was found, 0 otherwise.
GetAVPageView

Gets the AcroExch.AVPageView associated with an AcroExch.AVDoc.

Syntax

LPDISPATCH GetAVPageView();

Returns

The LPDISPATCH for the AcroExch.AVPageView or NULL if no document is open.

Related methods

AVDoc.GetPDDoc
AVDoc.SetViewMode
AVPageView.GetAVDoc
AVPageView.GetDoc

GetFrame

Gets the rectangle specifying the window's size and location.

Syntax

LPDISPATCH GetFrame();

Returns

The LPDISPATCH for an AcroExch.Rect containing the frame, or NULL if no document is open.

Related methods

AVDoc.SetFrame

GetPDDoc

Gets the AcroExch.PDDoc associated with an AcroExch.AVDoc.

Syntax

LPDISPATCH GetPDDoc();

Returns

The LPDISPATCH for the AcroExch.PDDoc or NULL if no document is open.
Related methods

AVDoc. GetAVPageView
AVPageView. GetAVDoc
AVPageView. GetDoc

GetTitle

Gets the window's title.

Syntax

BSTR GetTitle();

Returns

The window's title or NULL if no document is open.

Related methods

AVDoc. Open
AVDoc. SetTitle
PDDoc. OpenAVDoc

GetViewMode

Gets the current document view mode (pages only, pages and thumbnails, or pages and bookmarks).

Syntax

long GetViewMode();

Returns

The current document view mode or 0 if no document is open. The return value is one of the following:

PDDontCare: 0 — leave the view mode as it is
PDUseNone: 1 — display without bookmarks or thumbnails
PDUseThumbs: 2 — display using thumbnails
PDUseBookmarks: 3 — display using bookmarks
PDFullScreen: 4 — display in full screen mode

Related methods

AVDoc. GetAVPageView
AVDoc. SetViewMode
IsValid

Determines whether the AcroExch.AVDoc is still valid. This method only checks if the document has been closed or deleted; it does not check the internal structure of the document.

Syntax

VARIANT_BOOL IsValid();

Returns

-1 if the document can still be used, 0 otherwise.

Related methods

App.GetAVDoc
AVPageView.GetAVDoc

Maximize

Maximizes the window if bMaxSize is a positive number.

Syntax

VARIANT_BOOL Maximize(long bMaxSize);

Parameters

<table>
<thead>
<tr>
<th>bMaxSize</th>
<th>Indicates whether the window should be maximized.</th>
</tr>
</thead>
</table>

Returns

-1 if a document is open, 0 otherwise.

Related methods

AVDoc.GetFrame
AVDoc.SetFrame

Open

Opens a file. A new instance of AcroExch.AVDoc must be created for each displayed PDF file.

Note: An application must explicitly close any AVDoc that it opens by calling AVDoc.Close (the destructor for the AcroExch.AVDoc class does not call AVDoc.Close).
Syntax

```
VARIANT_BOOL Open(BSTR szFullPath, BSTR szTempTitle);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szFullPath</td>
<td>The full path of the file to open.</td>
</tr>
<tr>
<td>szTempTitle</td>
<td>An optional title for the window in which the file is opened. If szTempTitle is NULL or the empty string, it is ignored. Otherwise, szTempTitle is used as the window title.</td>
</tr>
</tbody>
</table>

Returns

-1 if the file was opened successfully, 0 otherwise.

Related methods

- `App.CloseAllDocs`
- `AVDoc.Close`
- `AVDoc.GetTitle`
- `AVDoc.OpenInWindow`
- `AVDoc.OpenInWindowEx`
- `AVDoc.SetTitle`
- `PDDoc.Close`
- `PDDoc.Open`
- `PDDoc.OpenAVDoc`

OpenInWindow

**Note:** As of Acrobat DC 3.0, this method simply returns `false`. Use the method `AVDoc.OpenInWindowEx` instead.

Syntax

```
VARIANT_BOOL OpenInWindow(BSTR fileName, short hWnd);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>The full path of the file to open.</td>
</tr>
<tr>
<td>hWnd</td>
<td>Handle for the window in which the file is displayed.</td>
</tr>
</tbody>
</table>
Returns

-1

Related methods

App.CloseAllDocs
AVDoc.Close
AVDoc.Open
AVDoc.OpenInWindowEx
PDDoc.Close
PDDoc.Open
PDDoc.OpenAVDoc

OpenInWindowEx

Opens a PDF file and displays it in a user-specified window. The default Windows file system is used to open the file.

**Note:** Acrobat DC uses only its built-in implementation of the file opening code—not any replacement file system version that a developer might have added with a plug-in.

An application must explicitly close any AVDoc that it opens by calling AVDoc.Close (the destructor for the AcroExch.AVDoc class does not call AVDoc.Close).

Do not set the view mode to Close with AVDoc.SetViewMode when using AVDoc.OpenInWindowEx; this will cause the viewer and application to hang.

If you use a view mode of AV_PAGE_VIEW, the pagemode parameter will be ignored.

See AVApp.Lock for a discussion of whether to lock the viewer before making this call.

Syntax

VARIANT_BOOL OpenInWindowEx(LPCTSTR szFullPath, long hWnd,
long openFlags, long useOpenParams
long pgNum, short pageMode,
short zoomType, long zoom, short top,
short left);

Parameters

<table>
<thead>
<tr>
<th>szFullPath</th>
<th>The full path of the file to open.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hWnd</td>
<td>Handle for the window in which the file is displayed.</td>
</tr>
</tbody>
</table>
openFlags Type of window view. Must be one of the following:

- AV_EXTERNAL_VIEW — Display the AVPageView, scrollbars, toolbar, and bookmark or thumbnails pane. Annotations are active.
- AV_DOC_VIEW — Display the AVPageView, scrollbars, and bookmark or thumbnails pane. Annotations are active.
- AV_PAGE_VIEW — Display only the AVPageView (the window that displays the PDF file). Do not display scrollbars, the toolbar, and bookmark or thumbnails pane. Annotations are active.

**Note:** Use either AV_DOC_VIEW or AV_PAGE_VIEW whenever possible. Use AV_EXTERNAL_VIEW only if you do not want the application to display its own toolbar. Use AV_PAGE_VIEW to open the file with no scrollbars and no status window at the bottom of the page.

useOpenParams 0 indicates that the open action of the file is used; a positive number indicates that the action is overridden with the parameters that follow.

pgNum Page number at which the file is to be opened if useOpenParams is a positive number. The first page is zero.

pageMode Specifies page view mode if useOpenParams is a positive number. Possible values:

- PDDontCare: 0 — leave the view mode as it is
- PDUseNone: 1 — display without bookmarks or thumbnails
- PDUseThumbs: 2 — display using thumbnails
- PDUseBookmarks: 3 — display using bookmarks
- PDFullScreen: 4 — display in full screen mode

zoomType Zoom type of the page view if useOpenParams is a positive number. Possible values are:

- AVZoomFitHeight — Fits the page’s height in the window.
- AVZoomFitPage — Fits the page in the window.
- AVZoomFitVisibleWidth — Fits the page’s visible content into the window.
- AVZoomFitWidth — Fits the page’s width into the window.
- AVZoomNoVary — A fixed zoom, such as 100%.

zoom Zoom factor, used only for AVZoomNoVary if useOpenParams is a positive number.

top Used for certain zoom types (such as AVZoomNoVary) if useOpenParams is a positive number. See the PDF Reference for information on views.

left Used for certain zoom types (such as AVZoomNoVary) if useOpenParams is a positive number. See the PDF Reference for information on views.

**Returns**

-1 if the document was opened successfully, 0 otherwise.
Related methods

- App.CloseAllDocs
- AVDoc.Close
- AVDoc.Open
- AVDoc.OpenInWindow
- PDDoc.Close
- PDDoc.Open
- PDDoc.OpenAVDoc

PrintPages

Prints a specified range of pages displaying a print progress dialog box. PrintPages always uses the default printer setting. It is possible to create custom dialog boxes as shown in the ActiveViewVB sample. Such custom dialog boxes could be used in place of the print progress dialog box or any other dialog box.

Syntax

```c
VARIANT_BOOL PrintPages(long nFirstPage,
                         long nLastPage, long nPSLevel,
                         long bBinaryOk, long bShrinkToFit);
```

Parameters

- **nFirstPage**: The first page to be printed. The first page in a PDDoc object is page 0.
- **nLastPage**: The last page to be printed.
- **nPSLevel**: Valid values are 2 and 3. If 2, PostScript® Level 2 operators are used. If 3, PostScript Language Level 3 operators are also used.
- **bBinaryOk**: If a positive number, binary data can be included in the PostScript program. If 0, all data is encoded as 7-bit ASCII.
- **bShrinkToFit**: If a positive number, the page is shrunk (if necessary) to fit within the imageable area of the printed page. If 0, it is not.

Returns

0 if there were any exceptions while printing or if no document was open, -1 otherwise.

Related methods

- AVDoc.PrintPagesEx
- AVDoc.PrintPagesSilent
- AVDoc.PrintPagesSilentEx
PrintPagesEx

Prints a specified range of pages, displaying a print progress dialog box. PrintPagesEx has more parameters than PrintPages. PrintPagesEx always uses the default printer setting. It is possible to create custom dialog boxes as shown in the ActiveViewVB sample. Such custom dialog boxes could be used in place of the print progress dialog box or any other dialog box.

Syntax

VARIANT_BOOL printPagesEx(long nFirstPage, long nLastPage, long nPSLevel, long bBinaryOk, long bShrinkToFit, long bReverse, long bFarEastFontOpt, long bEmitHalftones, long iPageOption);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFirstPage</td>
<td>The first page to be printed. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nLastPage</td>
<td>The last page to be printed.</td>
</tr>
<tr>
<td>nPSLevel</td>
<td>If 2, PostScript Level 2 operators are used. If 3, PostScript Language Level 3 operators are also used.</td>
</tr>
<tr>
<td>bBinaryOk</td>
<td>If a positive number, binary data may be included in the PostScript program. If 0, all data is encoded as 7-bit ASCII.</td>
</tr>
<tr>
<td>bShrinkToFit</td>
<td>If a positive number, the page is shrunk (if necessary) to fit within the imageable area of the printed page. If 0, it is not.</td>
</tr>
<tr>
<td>bReverse</td>
<td>(PostScript printing only) If a positive number, print the pages in reverse order. If false, print the pages in the regular order.</td>
</tr>
<tr>
<td>bFarEastFontOpt</td>
<td>(PostScript printing only) Set to a positive number if the destination printer has multibyte fonts; set to 0 otherwise.</td>
</tr>
<tr>
<td>bEmitHalftones</td>
<td>(PostScript printing only) If a positive number, emit the halftones specified in the document. If 0, do not.</td>
</tr>
<tr>
<td>iPageOption</td>
<td>Pages in the range to print. Must be one of: PDAllPages, PDEvenPagesOnly, or PDOddPagesOnly.</td>
</tr>
</tbody>
</table>

Returns

0 if there were any exceptions while printing or if no document was open, -1 otherwise.

Related methods

AVDoc. PrintPages
AVDoc. PrintPagesSilent
AVDoc. PrintPagesSilentEx
PrintPagesSilent

Prints a specified range of pages without displaying any dialog box. This method is identical to AVDoc.PrintPages except for not displaying the dialog box. PrintPagesSilent always uses the default printer setting.

Syntax

VARIANT_BOOL PrintPagesSilent(long nFirstPage, long nLastPage,
   long nPSLevel, long bBinaryOk,
   long bShrinkToFit);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFirstPage</td>
<td>The first page to be printed. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nLastPage</td>
<td>The last page to be printed.</td>
</tr>
<tr>
<td>nPSLevel</td>
<td>If 2, PostScript Level 2 operators are used. If 3, PostScript Language Level 3 operators are also used.</td>
</tr>
<tr>
<td>bBinaryOk</td>
<td>If a positive number, binary data may be included in the PostScript program. If 0, all data is encoded as 7-bit ASCII.</td>
</tr>
<tr>
<td>bShrinkToFit</td>
<td>If a positive number, the page is shrunk (if necessary) to fit within the imageable area of the printed page. If 0, it is not.</td>
</tr>
</tbody>
</table>

Returns

0 if there were any exceptions while printing or if no document was open, -1 otherwise.

Related methods

AVDoc.PrintPages
AVDoc.PrintPagesEx
AVDoc.PrintPagesSilentEx

PrintPagesSilentEx

Prints a specified range of pages without displaying any dialog box. This method is identical to AVDoc.PrintPagesEx except for not displaying the dialog box. PrintPagesSilentEx has more parameters than PrintPagesSilent. PrintPagesSilentEx always uses the default printer setting.

Syntax

VARIANT_BOOL PrintPagesSilentEx(long nFirstPage,
    long nLastPage,
    long nPSLevel, long bBinaryOk,
    long bShrinkToFit, long bReverse,
    long bFarEastFontOpt,
long bEmitHalftones,
long iPageOption);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFirstPage</td>
<td>The first page to be printed.</td>
</tr>
<tr>
<td>nLastPage</td>
<td>The last page to be printed.</td>
</tr>
<tr>
<td>nPSLevel</td>
<td>If 2, PostScript Level 2 operators are used. If 3, PostScript Language Level 3 operators are also used.</td>
</tr>
<tr>
<td>bBinaryOk</td>
<td>If a positive number, binary data may be included in the PostScript program. If 0, all data is encoded as 7-bit ASCII.</td>
</tr>
<tr>
<td>bShrinkToFit</td>
<td>If a positive number, the page is shrunk (if necessary) to fit within the imageable area of the printed page. If 0, it is not.</td>
</tr>
<tr>
<td>bReverse</td>
<td>(PostScript printing only) If a positive number, print the pages in reverse order. If false, print the pages in the regular order.</td>
</tr>
<tr>
<td>bFarEastFontOpt</td>
<td>(PostScript printing only) Set to a positive number if the destination printer has multibyte fonts; set to 0 otherwise.</td>
</tr>
<tr>
<td>bEmitHalftones</td>
<td>(PostScript printing only) If a positive number, emit the halftones specified in the document. If 0, do not.</td>
</tr>
<tr>
<td>iPageOption</td>
<td>Pages in the range to print. Must be one of: PDAllPages, PDEvenPagesOnly, or PDOddPagesOnly.</td>
</tr>
</tbody>
</table>

Returns

0 if there were any exceptions while printing, -1 otherwise.

Related methods

AVDoc.PrintPages
AVDoc.PrintPagesEx
AVDoc.PrintPagesSilentEx

SetFrame

Sets the window’s size and location.

Syntax

VARIANT_BOOL SetFrame(LPDISPATCH iAcroRect);
### Parameters

- **iAcroRect**: The LPDISPATCH for an AcroExch.Rect specifying the window frame. iAcroRect’s instance variable m_lpDispatch contains this LPDISPATCH.

### Returns

Always returns -1.

### Related methods

- AVDoc\.

### SetTextSelection

Sets the document’s selection to the specified text selection. Before calling this method, use one of the following to create the text selection:

- **PDDoc.** [CreateTextSelect](#) — Creates from a rectangle.
- **PDPage.** [CreatePageHilite](#) — Creates from a list of character offsets and counts.
- **PDPage.** [CreateWordHilite](#) — Creates from a list of word offsets and counts.

After calling this method, use AVDoc\.

### Syntax

VARIANT_BOOL SetTextSelection(LPDISPATCH iAcroPDTextSelect);

### Parameters

- **iAcroPDTextSelect**: The LPDISPATCH for the text selection to use. iAcroPDTextSelect contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

### Returns

Returns -1 if successful. Returns 0 if no document is open or the LPDISPATCH is not a PDTextSelect object.

### Related methods

- AVDoc\.
- AVDoc\.
- PDDoc\.
- PDPage\.
- PDPage\.
SetTitle

Sets the window's title.

Syntax

VARIANT_BOOL SetTitle(BSTR szTitle);

Parameters

| szTitle | The title to be set. This method cannot be used for document windows, but only for windows created by Plugins. |

Returns

Returns 0 if no document is open, −1 otherwise.

Related methods

AVDoc.GetTitle

AVDoc.Open

PDDoc.OpenAVDoc

SetViewMode

Sets the mode in which the document will be viewed (pages only, pages and thumbnails, or pages and bookmarks).

Syntax

VARIANT_BOOL SetViewMode(long nType);
### Parameters

**nType**

The view mode to be set. Possible values:
- PDDontCare: 0 — leave the view mode as it is
- PDUseNone: 1 — display without bookmarks or thumbnails
- PDUseThumbs: 2 — display using thumbnails
- PDUseBookmarks: 3 — display using bookmarks

**Note:** Do not set the view mode to Close with `AVDoc.SetViewMode` when using `AVDoc.OpenInWindowEx`; this will cause the viewer and application to hang.

### Returns

0 if an error occurred while setting the view mode or if no document was open, -1 otherwise.

### Related methods

- `AVDoc.GetAVPageView`
- `AVDoc.GetViewMode`
- `ShowTextSelect`

### ShowTextSelect

Changes the view so that the current text selection is visible.

#### Syntax

```cpp
VARIANT_BOOL ShowTextSelect();
```

#### Returns

Returns 0 if no document is open, -1 otherwise.

### Related methods

- `AVDoc.ClearSelection`
- `AVDoc.SetTextSelection`
- `PDDoc.CreateTextSelect`
- `PDPage.CreatePageHilite`
- `PDPage.CreateWordHilite`
- `PDTextSelect.Destroy`
- `PDTextSelect.GetBoundingRect`
- `PDTextSelect.GetNumText`
PDTextSelect.GetPage

PDTextSelect.GetText
AcroExch.AVPageView

The area of the Acrobat DC application's window that displays the contents of a document's page. This is a non-creatable interface. Every AVDoc object has an AVPageView object and vice versa. The object provides access to the PDDoc and PDPage objects for the document being displayed.

Methods

The AVPageView object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevicePointToPage</td>
<td>Converts the coordinates of a point from device space to user space.</td>
</tr>
<tr>
<td>DoGoBack</td>
<td>Goes to the previous view on the view history stack, if any.</td>
</tr>
<tr>
<td>DoGoForward</td>
<td>Goes to the next view on the view history stack, if any.</td>
</tr>
<tr>
<td>GetAperture</td>
<td>Gets the aperture of the specified page view.</td>
</tr>
<tr>
<td>GetAVDoc</td>
<td>Gets the AcroExch.AVDoc associated with the current page.</td>
</tr>
<tr>
<td>GetDoc</td>
<td>Gets the AcroExch.PDDoc corresponding to the current page.</td>
</tr>
<tr>
<td>GetPage</td>
<td>Gets the AcroExch.PDPage corresponding to the current page.</td>
</tr>
<tr>
<td>GetPageNum</td>
<td>Gets the page number of the current page.</td>
</tr>
<tr>
<td>GetZoom</td>
<td>Gets the current zoom factor, specified as a percent.</td>
</tr>
<tr>
<td>Get zoomType</td>
<td>Gets the current zoom type.</td>
</tr>
<tr>
<td>Goto</td>
<td>Goes to the specified page.</td>
</tr>
<tr>
<td>PointToDevice</td>
<td>Deprecated. Converts the coordinates of a point from user space to device space.</td>
</tr>
<tr>
<td>ReadPageDown</td>
<td>Scrolls forward through the document by one screen area.</td>
</tr>
<tr>
<td>ReadPageUp</td>
<td>Scrolls backward through the document by one screen area.</td>
</tr>
<tr>
<td>ScrollTo</td>
<td>Scrolls to the specified location on the current page.</td>
</tr>
<tr>
<td>ZoomTo</td>
<td>Zooms to the specified magnification.</td>
</tr>
</tbody>
</table>

DevicePointToPage

Converts the coordinates of a point from device space to user space.

Syntax

LPDISPATCH DevicePointToPage(LPDISPATCH iAcroPoint);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iAcroPoint</td>
<td>The LPDISPATCH for the AcroExch.Point whose coordinates are converted. iAcroPoint contains the instance variable m_lpDispatch, which contains the LPDISPATCH.</td>
</tr>
</tbody>
</table>

Returns

The LPDISPATCH for an AcroExch.Point containing the converted coordinates.

Related methods

AVPageView.PointToDevice

DoGoBack

Goes to the previous view on the view history stack, if any.

Syntax

VARIANT_BOOL DoGoBack();

Returns

Always returns -1.

Related methods

AVPageView.DoGoForward

DoGoForward

Goes to the next view on the view history stack, if any.

Syntax

VARIANT_BOOL DoGoForward();

Returns

Always returns -1.

Related methods

AVPageView.DoGoBack
GetAperture

Gets the aperture of the specified page view. The aperture is the rectangular region of the window in which the document is drawn, measured in device space units.

Syntax

CAcroRect* GetAperture();

Returns

A pointer to the aperture rectangle. Its coordinates are specified in device space.

Related methods

AVDoc.GetAVPageView
AVPageView.GetAVDoc
AVPageView.GetDoc
AVPageView.GetPage
AVPageView.GetZoomType

GetAVDoc

Gets the AcroExch.AVDoc associated with the current page.

Syntax

LPDISPATCH GetAVDoc();

Returns

The LPDISPATCH for the AcroExch.AVDoc.

Related methods

AVDoc.GetAVPageView
AVDoc.GetPDDoc
AVPageView.GetDoc

GetDoc

Gets the AcroExch.PDDoc corresponding to the current page.

Syntax

LPDISPATCH GetDoc();
Returns

The LPDISPATCH for the AcroExch.PDDoc.

Related methods

AVDoc.GetAVPageView
AVDoc.GetPDDoc
AVPageView.GetAVDoc

GetPage

Gets the AcroExch.PDPage corresponding to the current page.

Syntax

LPDISPATCH GetPage();

Returns

The LPDISPATCH for the AcroExch.PDPage.

Related methods

AVPageView.GetPageNum
PDDoc.AcquirePage
PDDoc.GetNumPages
PDPage.GetDoc
PDPage.GetNumber
PDPage.GetRotate
PDPage.GetSize
PDTextSelect.GetPage

GetPageNum

Gets the page number of the current page. The first page in a document is page zero.

Syntax

long GetPageNum();
Returns

The current page’s page number.

Related methods

AVPageView.GetPage
PDDoc.AcquirePage
PDDoc.GetNumPages
PDPage.GetDoc
PDPage.GetNumber
PDPage.GetRotate
PDPage.GetSize
PDTextSelect.GetPage

GetZoom

Gets the current zoom factor, specified as a percent. For example, 100 is returned if the magnification is 1.0.

Syntax

long GetZoom();

Returns

The current zoom factor.

Related methods

App.GetPreference
AVPageView.GetZoomType
AVPageView.ZoomTo

GetZoomType

Gets the current zoom type.

Syntax

short GetZoomType();
Returns

Zoom type. The value is one of the following:

AVZoomFitHeight — Fits the page’s height in the window.
AVZoomFitPage — Fits the page in the window.
AVZoomFitVisibleWidth — Fits the page’s visible content into the window.
AVZoomFitWidth — Fits the page’s width into the window.
AVZoomNoVary — A fixed zoom, such as 100%.

Related methods

App.GetPreference
AVPageView.GetZoomType
AVPageView.ZoomTo

Goto

Goes to the specified page.

Syntax

VARIANT_BOOL GoTo(long nPage);

Parameters

nPage Page number of the destination page. The first page in a PDDoc object is page 0.

Returns

-1 if the Acrobat DC application successfully went to the page, 0 otherwise.

Related methods

AVPageView.DoGoBack
AVPageView.DoGoForward
AVPageView.ReadPageDown
AVPageView.ReadPageUp
AVPageView.ScrollTo
AVPageView.ZoomTo
PointToDevice

Converts the coordinates of a point from user space to device space.

**Note:** Deprecated. Do not use this method.

**Syntax**

```cpp
LPDISPATCH PointToDevice(LPDISPATCH iAcroPoint);
```

**Parameters**

- `iAcroPoint` The `LPDISPATCH` for the `AcroExch.Point` whose coordinates are converted. `iAcroPoint` contains the instance variable `m_lpDispatch`, which contains this `LPDISPATCH`.

**Returns**

The `LPDISPATCH` for an `AcroExch.Point` containing the converted coordinates.

**Related methods**

`AVPageView.DevicePointToPage`

---

ReadPageDown

Scrolls forward through the document by one screen area.

**Syntax**

```cpp
VARIANT_BOOL ReadPageDown();
```

**Returns**

Always returns -1.

**Related methods**

`AVPageView.DoGoBack`
`AVPageView.DoGoForward`
`AVPageView.Goto`
`AVPageView.ReadPageUp`
`AVPageView.ScrollTo`
`AVPageView.ZoomTo`
ReadPageUp

Scrolls backward through the document by one screen area.

Syntax

VARIANT_BOOL ReadPageUp();

Returns

Always returns -1.

Related methods

AVPageView.DoGoBack
AVPageView.DoGoForward
AVPageView.Goto
AVPageView.ReadPageDown
AVPageView.ScrollTo
AVPageView.ZoomTo

ScrollTo

Scrolls to the specified location on the current page.

Syntax

VARIANT_BOOL ScrollTo(short nX, short nY);

Parameters

<table>
<thead>
<tr>
<th>nX</th>
<th>The x–coordinate of the destination.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nY</td>
<td>The y–coordinate of the destination.</td>
</tr>
</tbody>
</table>

Returns

-1 if the Acrobat DC application successfully scrolled to the specified location, 0 otherwise.

Related methods

AVPageView.DoGoBack
AVPageView.DoGoForward
AVPageView.Goto
AVPageView.ReadPageDown
AVPageView.ReadPageUp
AVPageView.ZoomTo

ZoomTo

Zooms to the specified magnification.

Syntax

VARIANT_BOOL ZoomTo(short nType, short nScale);

Parameters

<table>
<thead>
<tr>
<th>nType</th>
<th>Zoom type. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVZoomFitHeight — Fits the page’s height into the window.</td>
<td></td>
</tr>
<tr>
<td>AVZoomFitPage — Fits the page into the window.</td>
<td></td>
</tr>
<tr>
<td>AVZoomFitVisibleWidth — Fits the page’s visible content into the window.</td>
<td></td>
</tr>
<tr>
<td>AVZoomFitWidth — Fits the page’s width into the window.</td>
<td></td>
</tr>
<tr>
<td>AVZoomNoVary — A fixed zoom, such as 100%.</td>
<td></td>
</tr>
</tbody>
</table>

| nScale   | The desired zoom factor, expressed as a percentage. For example, 100 is a magnification of 1.0. |

Returns

-1 if the magnification was set successfully, 0 otherwise.

Related methods

AVPageView.GetZoomType
AVPageView.Goto
AVPageView.ScrollTo
AcroExch.HiliteList

A highlighted region of text in a PDF document, which may include one or more contiguous groups of characters or words on a single page. This is a creatable interface. This object has a single method, Add, and is used by the PDPage object to create PDTextSelect objects.

Add

Adds the highlight specified by nOffset and nLength to the current highlight list. Highlight lists are used to highlight one or more contiguous groups of characters or words on a single page.

Highlight lists are used both for character-based and word-based highlighting, although a single highlight list cannot contain a mixture of character and word highlights. After creating a highlight list, use PDPage.CreatePageHilite or PDPage.CreateWordHilite (depending on whether the highlight list is used for characters or words) to create a text selection from the highlight list.

Syntax

VARIANT_BOOL Add(short nOffset, short nLength);

Parameters

- nOffset: Offset of the first word or character to be highlighted, the first of which has an offset of zero.
- nLength: The number of consecutive words or characters to be highlighted.

Returns

Always returns -1.

Related methods

PDPage.CreatePageHilite
PDPage.CreateWordHilite

AcroExch.PDAnnot

An annotation on a page in a PDF file. This is a non-creatable interface. Acrobat DC applications have two built-in annotation types: PDTextAnnot and PDLInkAnnot. The object provides access to the physical attributes of the annotation. Plugins may add movie and Widget (form field) annotations, and developers can define new annotation subtypes by creating new annotation handlers.

Methods

The PDAnnot object has the following methods.
**Method** | **Description**
---|---
**GetColor** | Gets an annotation’s color.
**GetContents** | Gets a text annotation’s contents.
**GetDate** | Gets an annotation’s date.
**GetRect** | Gets an annotation’s bounding rectangle.
**GetSubtype** | Gets an annotation’s subtype.
**GetTitle** | Gets a text annotation’s title.
**IsEqual** | Determines whether an annotation is the same as the specified annotation.
**IsOpen** | Tests whether a text annotation is open.
**IsValid** | Tests whether an annotation is still valid.
**Perform** | Performs a link annotation’s action.
**SetColor** | Sets an annotation’s color.
**SetContents** | Sets a text annotation’s contents.
**SetDate** | Sets an annotation’s date.
**SetOpen** | Opens or closes a text annotation.
**SetRect** | Sets an annotation’s bounding rectangle.
**SetTitle** | Sets a text annotation’s title.

**GetColor**

Gets an annotation’s color.

**Syntax**

```c
long GetColor();
```

**Returns**

The annotation’s color, a long value of the form 0x00BBGGRR where the first byte from the right (RR) is a relative value for red, the second byte (GG) is a relative value for green, and the third byte (BB) is a relative value for blue. The high-order byte must be 0.

**Related methods**

PDAnnot. **SetColor**
GetContents

Gets a text annotation's contents.

Syntax

BSTR GetContents();

Returns

The annotation's contents.

Related methods

PDAnnot.SetContents
PDAnnot.GetDate
PDAnnot.GetRect
PDAnnot.GetSubtype
PDAnnot.GetTitle

GetDate

Gets an annotation's date.

Syntax

LPDISPATCH GetDate();

Returns

The LPDISPATCH for an AcroExch.Time object containing the date.

Related methods

PDAnnot.GetContents
PDAnnot.GetRect
PDAnnot.GetSubtype
PDAnnot.GetTitle
PDAnnot.SetDate

GetRect

Gets an annotation's bounding rectangle.
Syntax

LPDISPATCH GetRect();

Returns

The LPDISPATCH for an AcroExch.Rect containing the annotation’s bounding rectangle.

Related methods

PDAnnot.GetContents
PDAnnot.GetDate
PDAnnot.GetSubtype
PDAnnot.GetTitle
PDAnnot.SetRect

GetSubtype

Gets an annotation’s subtype.

Syntax

BSTR GetSubtype();

Returns

The annotation’s subtype. The built-in subtypes are Text and Link.

Related methods

PDAnnot.GetContents
PDAnnot.GetDate
PDAnnot.GetRect
PDAnnot.GetTitle

GetTitle

Gets a text annotation’s title.

Syntax

BSTR GetTitle();
Returns

The annotation’s title.

Related methods

PDAnnot. GetContents
PDAnnot. GetDate
PDAnnot. GetRect
PDAnnot. GetSubtype
PDAnnot. SetTitle

IsEqual

Determines whether an annotation is the same as the specified annotation.

Syntax

VARIANT_BOOL IsEqual(LPDISPATCH PDAnnot);

Parameters

| PDAnnot | The LPDISPATCH for the AcroExch.PDAnnot to be tested. PDAnnot contains the instance variable m_lpDispatch, which contains the LPDISPATCH. |

Returns

-1 if the annotations are the same, 0 otherwise.

Related methods

PDAnnot. GetContents
PDAnnot. GetDate
PDAnnot. GetRect
PDAnnot. GetSubtype
PDAnnot. GetTitle
PDAnnot. IsOpen
PDAnnot. IsValid

IsOpen

Tests whether a text annotation is open.
Syntax
VARIANT_BOOL IsOpen();

Returns
-1 if open, 0 otherwise.

Related methods
PDAnnot.GetContents
PDAnnot.GetDate
PDAnnot.GetRect
PDAnnot.GetSubtype
PDAnnot.GetTitle
PDAnnot.AreEqual
PDAnnot.IsValid
PDAnnot.IsOpen

IsValid
Tests whether an annotation is still valid. This method is intended only to test whether the annotation has been deleted, not whether it is a completely valid annotation object.

Syntax
VARIANT_BOOL IsValid();

Returns
-1 if the annotation is valid, 0 otherwise.

Related methods
PDAnnot.GetContents
PDAnnot.GetDate
PDAnnot.GetRect
PDAnnot.GetSubtype
PDAnnot.GetTitle
PDAnnot.AreEqual
PDAnnot.IsValid
PDAnnot.IsOpen
Perform

Performs a link annotation’s action.

Syntax

VARIANT_BOOL Perform(LPDISPATCH iAcroAVDoc);

Parameters

| iAcroAVDoc | The LPDISPATCH for the AcroExch.AVDoc in which the annotation is located. iAcroAVDoc contains the instance variable m_lpDispatch, which contains the LPDISPATCH. |

Returns

-1 if the action was executed successfully, 0 otherwise.

Related methods

PDAnnot. IsValid

SetColor

Sets an annotation’s color.

Syntax

VARIANT_BOOL SetColor(long nRGBColor);

Parameters

| nRGBColor | The color to use for the annotation. |

Returns

-1 if the annotation’s color was set, 0 if the Acrobat DC application does not support editing.

nRGBColor is a long value with the form 0x00BBGGRR where the first byte from the right (RR) is a relative value for red, the second byte (GG) is a relative value for green, and the third byte (BB) is a relative value for blue. The high-order byte must be 0.

Related methods

PDAnnot. GetColor
PDAnnot. SetContents
PDAnnot. SetDate
SetContents

Sets a text annotation's contents.

Syntax

\[
\text{VARIANT\_BOOL \ SetContents}(\text{BSTR szContents});
\]

Parameters

| szContents | The contents to use for the annotation. |

Returns

0 if the Acrobat DC application does not support editing, -1 otherwise.

Related methods

- PDAnnot.SetContents
- PDAnnot.SetColor
- PDAnnot.SetDate
- PDAnnot.SetOpen
- PDAnnot.SetRect
- PDAnnot.SetTitle

SetDate

Sets an annotation's date.

Syntax

\[
\text{VARIANT\_BOOL \ SetDate}(\text{LPDISPATCH iAcroTime});
\]

Parameters

| iAcroTime | The LPDISPATCH for the date and time to use for the annotation. |
|           | iAcroTime’s instance variable m_lpDispatch contains this LPDISPATCH. |
Returns

-1 if the date was set, 0 if the Acrobat DC application does not support editing.

Related methods

PDAnnot.SetOpen
PDAnnot.SetColor
PDAnnot.SetContents
PDAnnot.SetOpen
PDAnnot.SetRect
PDAnnotSetTitle

SetOpen

Opens or closes a text annotation.

Syntax

VARIANT_BOOL SetOpen(long bIsOpen);

Parameters

| bIsOpen | If a positive number, the annotation is open. If 0, the annotation is closed. |

Returns

Always returns -1.

Related methods

PDAnnot.IsOpen
PDAnnot.SetColor
PDAnnot.SetContents
PDAnnot.SetDate
PDAnnot.SetRect
PDAnnotSetTitle

SetRect

Sets an annotation’s bounding rectangle.
Syntax

VARIANT_BOOL SetRect(LPDISPATCH iAcroRect);

Parameters

| iAcroRect | The LPDISPATCH for the bounding rectangle (AcroExch.Rect) to set. iAcroRect contains the instance variable m_lpDispatch, which contains the LPDISPATCH. |

Returns

-1 if a rectangle was supplied, 0 otherwise.

Related methods

PDAnnot.SetRect
PDAnnot.SetColor
PDAnnot.SetContents
PDAnnot.SetDate
PDAnnot.SetOpen
PDAnnotSetTitle

SetTitle

Sets a text annotation's title.

Syntax

VARIANT_BOOL SetTitle(BSTR szTitle);

Parameters

| szTitle | The title to use. |

Returns

-1 if the title was set, 0 if the Acrobat DC application does not support editing.

Related methods

PDAnnot.GetByTitle
PDAnnotSetColor
PDAnnot.SetContents
AcroExch.PDBookmark

A bookmark for a page in a PDF file. This is a creatable interface. Each bookmark has a title that appears on screen, and an action that specifies what happens when a user clicks on the bookmark.

Bookmarks can either be created interactively by the user through the Acrobat DC application's user interface or programmatically generated. The typical action for a user-created bookmark is to move to another location in the current document, although any action can be specified. It is not possible to create a bookmark with OLE—only to destroy one.

Methods

The PDBookmark object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Destroy</strong></td>
<td>Destroys a bookmark.</td>
</tr>
<tr>
<td><strong>GetByTitle</strong></td>
<td>Gets the bookmark that has the specified title.</td>
</tr>
<tr>
<td><strong>GetTitle</strong></td>
<td>Gets a bookmark’s title.</td>
</tr>
<tr>
<td><strong>IsValid</strong></td>
<td>Determines whether the bookmark is valid.</td>
</tr>
<tr>
<td><strong>Perform</strong></td>
<td>Performs a bookmark’s action.</td>
</tr>
<tr>
<td><strong>SetTitle</strong></td>
<td>Sets a bookmark’s title.</td>
</tr>
</tbody>
</table>

**Destroy**

Destroys a bookmark.

**Syntax**

```c++
VARIANT_BOOL Destroy();
```

**Returns**

0 if the Acrobat DC application does not support editing (making it impossible to delete the bookmark), -1 otherwise.

**Related methods**

PDBookmark. **IsValid**
GetByTitle

Gets the bookmark that has the specified title. The AcroExch.PDBookmark object is set to the specified bookmark as a side effect of the method; it is not the method’s return value. You cannot enumerate bookmark titles with this method.

Syntax

VARIANT_BOOL GetByTitle(LPDISPATCH iAcroPDDoc, BSTR bookmarkTitle);

Parameters

- **iAcroPDDoc**: The LPDISPATCH for the document (AcroExch.PDDoc object) containing the bookmark. iAcroPDDoc contains the instance variable m_lpDispatch, which contains the LPDISPATCH.
- **bookmarkTitle**: The title of the bookmark to get. The capitalization of the title must match that in the bookmark.

Returns

-1 if the specified bookmark exists (the method determines this using the PDBookmark.IsValid method), 0 otherwise.

Related methods

- PDBookmark.GetTitle
- PDBookmark.SetTitle

Example

```cpp
CAcroPDBookmark* bookmark = new CAcroPDBookmark;

bookmark->CreateDispatch("AcroExch.PDBookmark");

bookmark->GetByTitle(m_pAcroAVDoc->GetPDDoc(), "Name of Bookmark");

if (bookmark->IsValid())
    bookmark->Perform(m_pAcroAVDoc->m_lpDispatch);
else
    AfxMessageBox("Bookmark not valid");
```

GetTitle

Gets a bookmark’s title.

Syntax

BSTR GetTitle();
IsValid

Determines whether the bookmark is valid. This method only checks whether the bookmark has been deleted; it does not thoroughly check the bookmark's data structures.

Syntax

VARIANT_BOOL IsValid();

Returns

-1 if the bookmark is valid, 0 otherwise.

Related methods

PDBookmark.SetTitle

Perform

Performs a bookmark's action.

Syntax

VARIANT_BOOL Perform(LPDISPATCH iAcroAVDoc);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iAcroAVDoc</td>
<td>The LPDISPATCH for the AcroExch.AVDoc in which the bookmark is located. iAcroAVDoc contains the instance variable m_lpDispatch, which contains the LPDISPATCH.</td>
</tr>
</tbody>
</table>

Returns

-1 if the action was executed successfully, 0 otherwise.
Related methods

PDBookmark.IsValid

SetTitle

Sets a bookmark's title.

Syntax

VARIANT_BOOL SetTitle(BSTR szNewTitle);

Parameters

| szNewTitle   | The title to set. |

Returns

0 if the Acrobat DC application does not support editing, -1 otherwise.

Related methods

PDBookmark.GetByTitle

PDBookmark.GetTitle

AcroExch.PDDoc

The underlying PDF representation of a document. This is a creatable interface. There is a correspondence between a PDDoc object and an ASFile object (an opaque representation of an open file made available through an interface encapsulating Acrobat DC’s access to file services), and the PDDoc object is the hidden object behind every AVDoc object. An ASFile object may have zero or more underlying files, so a PDF file does not always correspond to a single disk file. For example, an ASFile object may provide access to PDF data in a database.

Through PDDoc objects, your application can perform most of the Document menu items from Acrobat DC (delete pages, replace pages, and so on), create and delete thumbnails, and set and retrieve document information fields.

Methods

The PDDoc object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcquirePage</td>
<td>Acquires the specified page.</td>
</tr>
<tr>
<td>ClearFlags</td>
<td>Clears a document's flags.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes a file.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Create</td>
<td>Creates a new AcroExch.PDDoc.</td>
</tr>
<tr>
<td>CreateTextSelect</td>
<td>Creates a text selection from the specified rectangle on the specified page.</td>
</tr>
<tr>
<td>CreateThumbs</td>
<td>Creates thumbnail images for the specified page range in a document.</td>
</tr>
<tr>
<td>CropPages</td>
<td>Crops the pages in a specified range in a document.</td>
</tr>
<tr>
<td>DeletePages</td>
<td>Deletes pages from a file.</td>
</tr>
<tr>
<td>DeleteThumbs</td>
<td>Deletes thumbnail images from the specified pages in a document.</td>
</tr>
<tr>
<td>GetFileName</td>
<td>Gets the name of the file associated with this AcroExch.PDDoc.</td>
</tr>
<tr>
<td>GetFlags</td>
<td>Gets a document’s flags.</td>
</tr>
<tr>
<td>GetInfo</td>
<td>Gets the value of a specified key in the document’s Info dictionary.</td>
</tr>
<tr>
<td>GetInstanceID</td>
<td>Gets the instance ID (the second element) from the ID array in the document’s trailer.</td>
</tr>
<tr>
<td>GetJSObject</td>
<td>Gets a dual interface to the JavaScript object associated with the PDDoc.</td>
</tr>
<tr>
<td>GetNumPages</td>
<td>Gets the number of pages in a file.</td>
</tr>
<tr>
<td>GetPageMode</td>
<td>Gets a value indicating whether the Acrobat DC application is currently displaying only pages, pages and thumbnails, or pages and bookmarks.</td>
</tr>
<tr>
<td>GetPermanentID</td>
<td>Gets the permanent ID (the first element) from the ID array in the document’s trailer.</td>
</tr>
<tr>
<td>InsertPages</td>
<td>Inserts the specified pages from the source document after the indicated page within the current document.</td>
</tr>
<tr>
<td>MovePage</td>
<td>Moves a page to another location within the same document.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens a file.</td>
</tr>
<tr>
<td>OpenAVDoc</td>
<td>Opens a window and displays the document in it.</td>
</tr>
<tr>
<td>ReplacePages</td>
<td>Replaces the indicated pages in the current document with those specified from the source document.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves a document.</td>
</tr>
<tr>
<td>SetFlags</td>
<td>Sets a document’s flags indicating whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file.</td>
</tr>
<tr>
<td>SetInfo</td>
<td>Sets the value of a key in a document’s Info dictionary.</td>
</tr>
<tr>
<td>SetPageMode</td>
<td>Sets the page mode in which a document is to be opened: display only pages, pages and thumbnails, or pages and bookmarks.</td>
</tr>
</tbody>
</table>
AcquirePage

Acquires the specified page.

Syntax

LPDISPATCH AcquirePage(long nPage);

Parameters

<table>
<thead>
<tr>
<th>nPage</th>
<th>The number of the page to acquire. The first page is page 0.</th>
</tr>
</thead>
</table>

Returns

The LPDISPATCH for the AcroExch.PDPage object for the acquired page. Returns NULL if the page could not be acquired.

Related methods

AVPageView.GetPage
AVPageView.GetPageNum
PDDoc.GetNumPages
PDPAGE.GetDoc
PDPAGE.GetNumber
PDPAGE.GetRotate
PDPAGE.GetSize
PDTextSelect.GetPage

ClearFlags

Clears a document’s flags. The flags indicate whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file. This method can be used only to clear, not to set, the flag bits.

Syntax

VARIANT_BOOL ClearFlags(long nFlags);

Parameters

| nFlags         | Flags to be cleared. See PDDoc.GetFlags for a description of the flags. The flags PDDocWasRepaired, PDDocNewMajorVersion, PDDocNewMinorVersion, and PDDocOldVersion are read-only and cannot be cleared. |
Returns

Always returns -1.

Related methods

PDDoc.GetFlags
PDDoc.SetFlags

Close

Closes a file.

Note: If PDDoc and AVDoc are constructed with the same file, PDDoc.Close destroys both objects (which closes the document in the viewer).

Syntax

VARIANT_BOOL Close();

Returns

-1 if the document was closed successfully, 0 otherwise.

Related methods

App.CloseAllDocs
AVDoc.Close
AVDoc.Open
AVDoc.OpenInWindow
AVDoc.OpenInWindowEx
PDDoc.Open
PDDoc.OpenAVDoc

Create

Creates a new AcroExch.PDDoc.

Syntax

VARIANT_BOOL Create();
Returns

-1 if the document is created successfully, 0 if it is not or if the Acrobat DC application does not support editing.

CreateTextSelect

Creates a text selection from the specified rectangle on the specified page. After creating the text selection, use the AVDoc.SetTextSelection method to use it as the document’s selection, and use AVDoc.ShowTextSelect to show the selection.

Syntax

LPDISPATCH CreateTextSelect(long nPage, LPDISPATCH iAcroRect);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The page on which the selection is created. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>iAcroRect</td>
<td>The LPDISPATCH for the AcroExch.Rect enclosing the region to select. iAcroRect contains the instance variable m_lpDispatch, which contains the LPDISPATCH.</td>
</tr>
</tbody>
</table>

Returns

The LPDISPATCH for an AcroExch.PDTextSelect containing the text selection. Returns NULL if the text selection was not created successfully.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
PDPage.CreatePageHilite
PDPage.CreateWordHilite
PDTextSelect_Destroy
PDTextSelect_GetBoundingRect
PDTextSelect_GetNumText
PDTextSelect_GetPage
PDTextSelect_GetText
CreateThumbs

Creates thumbnail images for the specified page range in a document.

Syntax

VARIANT_BOOL CreateThumbs(long nFirstPage, long nLastPage);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFirstPage</td>
<td>First page for which thumbnail images are created. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nLastPage</td>
<td>Last page for which thumbnail images are created.</td>
</tr>
</tbody>
</table>

Returns

-1 if thumbnail images were created successfully, 0 if they were not or if the Acrobat DC application does not support editing.

Related methods

PDDoc.DeleteThumbs

CropPages

Crops the pages in a specified range in a document. This method ignores the request if either the width or height of the crop box is less than 72 points (one inch).

Syntax

VARIANT_BOOL CropPages(long nStartPage, long nEndPage, short nEvenOrOddPagesOnly, LPDISPATCH iAcroRect);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartPage</td>
<td>First page that is cropped. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nEndPage</td>
<td>Last page that is cropped.</td>
</tr>
<tr>
<td>nEvenOrOddPagesOnly</td>
<td>Value indicating which pages in the range are cropped. Must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>0 — crop all pages in the range</td>
</tr>
<tr>
<td></td>
<td>1 — crop only odd pages in the range</td>
</tr>
<tr>
<td></td>
<td>2 — crop only even pages in the range</td>
</tr>
<tr>
<td>iAcroRect</td>
<td>An LPDISPATCH for a CAcroRect specifying the cropping rectangle, which is specified in user space.</td>
</tr>
</tbody>
</table>
Returns

-1 if the pages were cropped successfully, 0 otherwise.

Related methods

PDPages.CropPages

DeletePages

Deletes pages from a file.

Syntax

VARIANT_BOOL DeletePages(long nStartPage, long nEndPage);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartPage</td>
<td>The first page to be deleted. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nEndPage</td>
<td>The last page to be deleted.</td>
</tr>
</tbody>
</table>

Returns

-1 if the pages were successfully deleted. Returns 0 if they were not or if the Acrobat DC application does not support editing.

Related methods

PDDoc.AcquirePage
PDDoc.DeletePages
PDDoc.GetNumPages
PDDoc.InsertPages
PDDoc.MovePage
PDDoc.ReplacePages

DeleteThumbs

Deletes thumbnail images from the specified pages in a document.

Syntax

VARIANT_BOOL DeleteThumbs(long nStartPage, long nEndPage);
GetFileName

Gets the name of the file associated with this AcroExch.PDDoc.

Syntax

BSTR GetFileName();

Returns

The file name, which can currently contain up to 256 characters.

Related methods

PDDoc.CreateThumbs

GetFlags

Gets a document’s flags. The flags indicate whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file.

Syntax

long GetFlags();

Returns

The document’s flags, containing an OR of the following:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDocNeedsSave</td>
<td>Document has been modified and needs to be saved.</td>
</tr>
</tbody>
</table>
GetInfo

Gets the value of a specified key in the document’s Info dictionary. A maximum of 512 bytes are returned.

Syntax

BSTR GetInfo(BSTR szInfoKey);

Parameters

szInfoKey The key whose value is obtained.

Returns

The string if the value was read successfully. Returns an empty string if the key does not exist or its value cannot be read.

Related methods

PDDoc.SetInfo

GetInstanceID

Gets the instance ID (the second element) from the ID array in the document's trailer.
Syntax

BSTR GetInstanceID();

Returns

A string whose maximum length is 32 characters, containing the document’s instance ID.

Related methods

PDDoc.GetPermanentID

GetJSObject

Gets a dual interface to the JavaScript object associated with the PDDoc. This allows automation clients full access to both built-in and user-defined JavaScript methods available in the document.

Syntax

LDispatch* GetJSObject();

Returns

The interface to the JavaScript object if the call succeeded, NULL otherwise.

GetNumPages

Gets the number of pages in a file.

Syntax

long GetNumPages();

Returns

The number of pages, or -1 if the number of pages cannot be determined.

Related methods

AVPageView.GetPage
AVPageView.GetPageNum
PDDoc.AcquirePage
PDPage.GetNumber
PDTextSelect.GetPage
GetPageMode

Gets a value indicating whether the Acrobat DC application is currently displaying only pages, pages and thumbnails, or pages and bookmarks.

Syntax

long GetPageMode();

Returns

The current page mode. Will be one of the following values:
- PDDontCare: 0 — leave the view mode as it is
- PDUseNone: 1 — display without bookmarks or thumbnails
- PDUseThumbs: 2 — display using thumbnails
- PDUseBookmarks: 3 — display using bookmarks
- PDFullScreen: 4 — display in full screen mode

Related methods

PDDoc.SetPageMode

GetPermanentID

 Gets the permanent ID (the first element) from the ID array in the document’s trailer.

Syntax

BSTR GetPermanentID();

Returns

A string whose maximum length is 32 characters, containing the document’s permanent ID.

Related methods

PDDoc.GetInstanceID

InsertPages

Inserts the specified pages from the source document after the indicated page within the current document.

Syntax

VARIANT_BOOL InsertPages(long nInsertPageAfter, 
LPDISPATCH iPDDocSource, long nStartPage, 
long nNumPages, long bBookmarks);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nInsertPageAfter</td>
<td>The page in the current document after which pages from the source document are inserted. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>iPDDocSource</td>
<td>The LPDISPATCH for the AcroExch.PDoc containing the pages to insert. iPDDocSource contains the instance variable m_lpDispatch, which contains the LPDISPATCH.</td>
</tr>
<tr>
<td>nStartPage</td>
<td>The first page in iPDDocSource to be inserted into the current document.</td>
</tr>
<tr>
<td>nNumPages</td>
<td>The number of pages to be inserted.</td>
</tr>
<tr>
<td>bBookmarks</td>
<td>If a positive number, bookmarks are copied from the source document. If 0, they are not.</td>
</tr>
</tbody>
</table>

Returns

-1 if the pages were successfully inserted. Returns 0 if they were not or if the Acrobat DC application does not support editing.

Related methods

- PDDoc.AcquirePage
- PDDoc.DeletePages
- PDDoc.GetNumPages
- PDDoc.MovePage
- PDDoc.ReplacePages

MovePage

Moves a page to another location within the same document.

Syntax

VARIANT_BOOL MovePage(long nMoveAfterThisPage, long nPageToMove);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMoveAfterThisPage</td>
<td>The page being moved is placed after this page number. The first page in a PDDoc object is page 0.</td>
</tr>
<tr>
<td>nPageToMove</td>
<td>Page number of the page to be moved.</td>
</tr>
</tbody>
</table>

Returns

0 if the Acrobat DC application does not support editing, -1 otherwise.
Related methods

PDDoc.AcquirePage
PDDoc.DeletePages
PDDoc.GetNumPages
PDDoc.InsertPages
PDDoc.ReplacePages

Open

Opens a file. A new instance of AcroExch.PDDoc must be created for each open PDF file.

Syntax

VARIANT_BOOL Open(BSTR szFullPath);

Parameters

| szFullPath | Full path of the file to be opened. |

Returns

-1 if the document was opened successfully, 0 otherwise.

Related methods

App.CloseAllDocs
AVDoc.Close
AVDoc.Open
AVDoc.OpenInWindow
AVDoc.OpenInWindowEx
PDDoc.Close
PDDoc.OpenAVDoc

OpenAVDoc

Opens a window and displays the document in it.

Syntax

LPDISPATCH OpenAVDoc(BSTR szTitle);
Parameters

| szTitle                              | The title to be used for the window. A default title is used if szTitle is NULL or an empty string. |

Returns

The LPDISPATCH for the AcroExch.AVDoc that was opened, or NULL if the open fails.

Related methods

App.CloseAllDocs
AVDoc.Close
AVDoc.GetTitle
AVDoc.Open
AVDoc.OpenInWindow
AVDoc.OpenInWindowEx
AVDoc.SetTitle
PDDoc.Close
PDDoc.Open

ReplacePages

Replaces the indicated pages in the current document with those specified from the source document. No links or bookmarks are copied from iPDDocSource, but text annotations may optionally be copied.

Syntax

VARIANT_BOOL ReplacePages(long nStartPage,
                          LPDISPATCH iPDDocSource,
                          long nStartSourcePage, long nNumPages,
                          long bMergeTextAnnotations);

Parameters

<table>
<thead>
<tr>
<th>nStartPage</th>
<th>The first page within the source file to be replaced. The first page in a PDDoc object is page 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPDDocSource</td>
<td>The LPDISPATCH for the AcroExch.PDDoc containing the new copies of pages that are replaced. iPDDocSource contains the instance variable m_lDispatch, which contains the LPDISPATCH.</td>
</tr>
<tr>
<td>nStartSourcePage</td>
<td>The first page in iPDDocSource to use as a replacement page.</td>
</tr>
</tbody>
</table>
Returns

-1 if the pages were successfully replaced. Returns 0 if they were not or if the Acrobat DC application does not support editing.

Related methods

PDDoc.AcquirePage
PDDoc.DeletePages
PDDoc.GetNumPages
PDDoc.InsertPages
PDDoc.MovePage

Save

Saves a document.

Syntax

VARIANT_BOOL Save(short nType, BSTR szFullPath);
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **nType**  | Specifies the way in which the file should be saved.  
  *nType* is a logical OR of one or more of the following flags:  
  - `PDSaveIncremental` — Write changes only, not the complete file. This will always result in a larger file, even if objects have been deleted.  
  - `PDSaveFull` — Write the entire file to the filename specified by `szFullPath`.  
  - `PDSaveCopy` — Write a copy of the file into the file specified by `szFullPath`, but keep using the old file. This flag can only be specified if `PDSaveFull` is also used.  
  - `PDSaveCollectGarbage` — Remove unreferenced objects; this often reduces the file size, and its usage is encouraged. This flag can only be specified if `PDSaveFull` is also used.  
  - `PDSaveLinearized` — Save the file optimized for the web, providing hint tables. This allows the PDF file to be byte-served. This flag can only be specified if `PDSaveFull` is also used.  
  **Note:** If you save a file optimized for the web using the `PDSaveLinearized` flag, you must follow this sequence:  
  1. Open the PDF file with `PDDoc.Open`.  
  2. Call `PDDoc.Save` using the `PDSaveLinearized` flag.  
  This allows batch optimization of files. |
| **szFullPath** | The new path to the file, if any. |

### Returns

-1 if the document was successfully saved. Returns 0 if it was not or if the Acrobat DC application does not support editing.

### Related methods

- `PDDoc.GetFileName`

### SetFlags

Sets a document’s flags indicating whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file. This method can be used only to set, not to clear, the flag bits.

### Syntax

```csharp
VARIANT_BOOL SetFlags(long nFlags);
```
Parameters

| nFlags | Flags to be set. See PDDoc.GetFlags for a description of the flags. The flags PDDocWasRepaired, PDDocNewMajorVersion, PDDocNewMinorVersion, and PDDocOldVersion are read-only and cannot be set. |

Related methods

- PDDoc.ClearFlags
- PDDoc.GetFlags

SetInfo

Sets the value of a key in a document’s Info dictionary.

Syntax

VARIANT_BOOL SetInfo(BSTR szInfoKey, BSTR szBuffer); 

Parameters

| szInfoKey | The key whose value is set. |
| szBuffer | The value to be assigned to the key. |

Returns

-1 if the value was added successfully, 0 if it was not or if the Acrobat DC application does not support editing.

Related methods

- PDDoc.GetInfo

SetPageMode

Sets the page mode in which a document is to be opened: display only pages, pages and thumbnails, or pages and bookmarks.

Syntax

VARIANT_BOOL SetPageMode(long nPageMode);
Parameters

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPageMode</td>
<td>The page mode to be set. Possible values:</td>
</tr>
<tr>
<td></td>
<td>PDDontCare: 0 — leave the view mode as it is</td>
</tr>
<tr>
<td></td>
<td>PDUseNone: 1 — display without bookmarks or thumbnails</td>
</tr>
<tr>
<td></td>
<td>PDUseThumbs: 2 — display using thumbnails</td>
</tr>
<tr>
<td></td>
<td>PDUseBookmarks: 3 — display using bookmarks</td>
</tr>
</tbody>
</table>

Returns

Always returns -1.

Related methods

PDDoc.GetPageMode
PDDoc.SetPageMode

AcroExch.PDPage

A single page in the PDF representation of a document. This is a non-creatable interface. Just as PDF files are partially composed of their pages, PDDoc objects are composed of PDPage objects. A page contains a series of objects representing the objects drawn on the page (PDGraphic objects), a list of resources used in drawing the page, annotations (PDAnnot objects), an optional thumbnail image of the page, and the threads used in any articles that occur on the page. The first page in a PDDoc object is page 0.

Methods

The PDPage object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddAnnot</td>
<td>Adds a specified annotation at a specified location in the page's annotation array</td>
</tr>
<tr>
<td>AddNewAnnot</td>
<td>Creates a new text annotation and adds it to the page.</td>
</tr>
<tr>
<td>CopyToClipboard</td>
<td>Copies a PDF image to the clipboard without requiring an hWnd or hDC from the client.</td>
</tr>
<tr>
<td>CreatePageHilite</td>
<td>Creates a text selection from a list of character offsets and character counts on a single page.</td>
</tr>
<tr>
<td>CreateWordHilite</td>
<td>Creates a text selection from a list of word offsets and word counts on a single page.</td>
</tr>
<tr>
<td>CropPage</td>
<td>Crops the page.</td>
</tr>
<tr>
<td>Draw</td>
<td>Deprecated. Draws page contents into a specified window.</td>
</tr>
</tbody>
</table>
AddAnnot

Add a specified annotation at a specified location in the page’s annotation array.

Syntax

VARIANT_BOOL AddAnnot(long nIndexAddAfter,
                        LPDISPATCH iPDAnnot);

Parameters

nIndexAddAfter Location in the page’s annotation array to add the annotation. The first annotation on a page has an index of zero.

iPDAnnot The LPDISPATCH for the AcroExch.PDAnnot to add. iPDAnnot contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Returns

0 if the Acrobat DC application does not support editing, -1 otherwise.

Related methods

PDPage. AddNewAnnot
PDPage. RemoveAnnot
AddNewAnnot

Creates a new text annotation and adds it to the page.

The newly-created text annotation is not complete until PDAnnot.SetContents has been called to fill in the /Contents key.

Syntax

LPDISPATCH AddNewAnnot(long nIndexAddAfter, BSTR szSubType, LPDISPATCH iAcroRect);

Parameters

- nIndexAddAfter: Location in the page’s annotation array after which to add the annotation. The first annotation on a page has an index of zero.
- szSubType: Subtype of the annotation to be created. Must be text.
- iAcroRect: The LPDISPATCH for the AcroExch.Rect bounding the annotation’s location on the page. iAcroRect contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Returns

The LPDISPATCH for an AcroExch.PDAnnot object, or NULL if the annotation could not be added.

Related methods

- PDAnnot.SetContents
- PDPage.AddAnnot
- PDPage.RemoveAnnot

CopyToClipboard

Copies a PDF image to the clipboard without requiring an hWnd or hDC from the client. This method is only available on 32-bit systems.

Syntax

VARIANT_BOOL CopyToClipboard(LPDISPATCH boundRect, short nXOrigin, short nYOrigin, short nZoom);
CreatePageHilite

Creates a text selection from a list of character offsets and character counts on a single page. The text selection can then be set as the current selection using AVDoc.SetTextSelection, and the view can be set to show the selection using AVDoc.ShowTextSelect.

Syntax

LPDISPATCH CreatePageHilite(LPDISPATCH iAcroHiliteList);

Parameters

iAcroHiliteList The LPDISPATCH for the highlight list for which a text selection is created. iAcroHiliteList contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Use HiliteList.Add to create a highlight list.

Returns

The LPDISPATCH for the AcroExch.PDTextSelect containing the text selection, or NULL if the selection could not be created.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
HiliteList.Add
PDDoc.CreateTextSelect
PDPage.CreateWordHilite
PDTextSelect.Destroy
PDTextSelect.GetBoundingRect
PDTextSelect.GetNumText
PDTextSelect.GetPage
PDTextSelect.GetText

CreateWordHilite

Creates a text selection from a list of word offsets and word counts on a single page. The text selection can then be set as the current selection using AVDoc.SetTextSelection, and the view can be set to show the selection using AVDoc.ShowTextSelect.

Syntax

LPDISPATCH CreateWordHilite(LPDISPATCH iAcroHiliteList);

Parameters

iAcroHiliteList  The LPDISPATCH for the highlight list for which a text selection is created. iAcroHiliteList contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Use HiliteList.Add to create a highlight list.

Returns

The LPDISPATCH for the AcroExch.PDTextSelect, or NULL if the selection could not be created.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
HiliteList.Add
PDDoc.CreateTextSelect
PDPage.CreatePageHilite
PDTextSelect.Destroy
CropPage

Crops the page. This method ignores the request if either the width or height of the crop box is less than 72 points (one inch).

Syntax

VARIANT_BOOL CropPage(LPDISPATCH iAcroRect);

Parameters

| iAcroRect | An LPDISPATCH for a CAcroRect specifying the cropping rectangle, which is specified in user space. |

Returns

-1 if the page was cropped successfully, 0 otherwise.

Related methods

PDDoc.CropPages

Draw

Note: Deprecated. As of Acrobat DC 3.0, this method simply returns false. Use the method AVDoc.DrawEx instead.

Syntax

VARIANT_BOOL Draw(short window, short displayContext, short XOrigin, short YOrigin, short zoom);

Parameters

| window | HWND into which the page is to be drawn. |
| displayContext | hDC to use for drawing. If NULL, the hDC for window is used. |
| displayContext | cannot be reliably used as the hDC for a printer device. In particular, Visual Basic applications cannot use Draw to print. |
| XOrigin | The x-coordinate of the portion of the page to be drawn. |
Returns

-1 if the page is successfully drawn, 0 otherwise.

Related methods

PDPage. CopyToClipboard
PDPage. DrawEx

DrawEx

Draws page contents into a specified window.

You can use PDPage. CopyToClipboard to copy page contents to the clipboard without an hWnd or hDC from the client.

Syntax

VARIANT_BOOL DrawEx(long window, long displayContext, LPDISPATCH updateRect, short xOrigin, short yOrigin, short zoom);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>Handle for the window (HWND) into which the page is drawn.</td>
</tr>
<tr>
<td>displayContext</td>
<td>This parameter is invalid; do not use it. Assign it a NULL value. If it is not assigned NULL, an exception is thrown.</td>
</tr>
<tr>
<td>yOrigin</td>
<td>The y-coordinate of the portion of the page to be drawn.</td>
</tr>
<tr>
<td>zoom</td>
<td>Zoom factor at which the page is to be drawn, specified as a percent. For example, 100 corresponds to a magnification of 1.0.</td>
</tr>
<tr>
<td>Note: displayContext cannot be reliably used as the hDC for a printer device. In particular, Visual Basic applications cannot use DrawEx to print.</td>
<td></td>
</tr>
</tbody>
</table>
Returns

A positive number if the page is successfully drawn, 0 otherwise.

Related methods

PDPAGE.CopyToClipboard

GetAnnot

Gets the specified annotation from the page's array of annotations.

Syntax

LPDISPATCH GetAnnot(long nIndex);

Parameters

nIndex Index (in the page's annotation array) of the annotation to be retrieved. The first annotation in the array has an index of zero.

Returns

The LPDISPATCH for the AcroExch.PDAnnot object.
Related methods

PDPages. GetAnnotIndex
PDPaged. GetNumAnnots

GetAnnotIndex

Gets the index (within the page’s annotation array) of the specified annotation.

Syntax

long GetAnnotIndex(LPDISPATCH iPDAnnot);

Parameters

iPDAnnot LPDISPATCH for the AcroExch.PDAannot whose index is obtained. iPDAnnot contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Returns

The annotation’s index.

Related methods

PDPages. GetAnnot
PDPages. GetNumAnnots

GetDoc

Gets the AcroExch.PDDoc associated with the page.

Syntax

LPDISPATCH GetDoc();

Returns

The LPDISPATCH for the page’s AcroExch.PDDoc.

Related methods

AVPageView. GetPage
AVPageView. GetPageNum
PDDocs. AcquirePage
PDDocs. GetNumPages
GetNumAnnots

Gets the number of annotations on the page.

Annotations that have associated pop-up windows, such as a strikeout, count as two annotations. Also note that widget annotations (Acrobat DC form fields) are included.

Syntax

```csharp
long GetNumAnnots();
```

Returns

The number of annotations on the page.

Related methods

- PDPage.GetAnnot
- PDPage.GetAnnotIndex

GetNumber

Gets the page number of the current page. The first page in a document is page zero.

Syntax

```csharp
long GetNumber();
```

Returns

The page number of the current page. The first page in a PDDoc object is page 0.

Related methods

- AVPageView.GetPage
- AVPageView.GetPageNum
- PDDoc.AcquirePage
- PDDoc.GetNumPages
- PDPage.GetDoc
GetRotate

Gets the rotation value, in degrees, for the current page.

Syntax

```c
short GetRotate();
```

Returns

Rotation value.

Related methods

- AVPageView.GetPage
- AVPageView.GetPageNum
- PDDoc.AcquirePage
- PDPage.GetNumber
- PDPage.GetSize
- PDPage.SetRotate
- PDTextSelect.GetPage

GetSize

Gets a page's width and height in points.

Syntax

```c
LPDISPATCH GetSize();
```

Returns

The LPDISPATCH for an AcroExch.Point containing the width and height, measured in points. Point x contains the width, point y the height.

Related methods

- AVPageView.GetPage
- AVPageView.GetPageNum
RemoveAnnot

Removes the specified annotation from the page’s annotation array.

Syntax

VARIANT_BOOL RemoveAnnot(long nIndex);

Parameters

nIndex

Index within the page’s annotation array of the annotation to be deleted. The first annotation on a page has an index of zero.

Returns

0 if the Acrobat DC application does not support editing, a positive number otherwise.

Related methods

PDPage.AddAnnot
PDPage.AddNewAnnot
PDPage.GetAnnotIndex

SetRotate

Sets the rotation, in degrees, for the current page.

Syntax

VARIANT_BOOL SetRotate(short nRotate);

Parameters

nRotate

Rotation value of 0, 90, 180, or 270.

Returns

0 if the Acrobat DC application does not support editing, -1 otherwise.
Related methods

PDPage::GetRotate
AcroExch.PDTextSelect

A selection of text on a single page that may contain more than one disjointed group of words. This is a non-creatable interface. A text selection is specified by one or more ranges of text, with each range containing the word numbers of the selected words. Each range specifies a start and end word, where “start” is the number of the first word of a series of selected words and “end” is the number of the next word after the last word in the selection.

Methods

The PDTextSelect object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroy</td>
<td>Destroys a text selection object.</td>
</tr>
<tr>
<td>GetBoundingRect</td>
<td>Gets a text selection’s bounding rectangle.</td>
</tr>
<tr>
<td>GetNumText</td>
<td>Gets the number of text elements in a text selection.</td>
</tr>
<tr>
<td>GetPage</td>
<td>Gets the page number on which the text selection is located.</td>
</tr>
<tr>
<td>GetText</td>
<td>Gets the text from the specified element of a text selection.</td>
</tr>
</tbody>
</table>

Destroy

Destroys a text selection object.

Syntax

VARIANT_BOOL Destroy();

Returns

Always returns -1.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
PDDoc.CreateTextSelect
PDPPage.CreatePageHilite
PDPPage.CreateWordHilite
PDTextSelect.GetBoundingRect
PDTextSelect.GetNumText
GetBoundingRect

Gets a text selection’s bounding rectangle.

Syntax

LPDISPATCH GetBoundingRect();

Returns

The LPDISPATCH for an AcroExch.Rect corresponding to the text selection’s bounding rectangle.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
PDDoc.CreateTextSelect
PDPage.CreatePageHilite
PDPage.CreateWordHilite
PDTextSelect.Destroy
PDTextSelect.GetNumText
PDTextSelect.GetPage
PDTextSelect.GetText

GetNumText

Gets the number of text elements in a text selection. Use this method to determine how many times to call the PDTextSelect.GetText method to obtain all of a text selection’s text.

Note: A text element is not necessarily a word. A text element consists of characters of the same font, size and style; therefore, there may be more than one text element in a word.

Syntax

long GetNumText();

Returns

The number of elements in the text selection.
Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
PDDoc.CreateTextSelect
PDPage.CreatePageHilite
PDPage.CreateWordHilite
PDTextSelect.Destroy
PDTextSelect.GetBoundingRect
PDTextSelect.GetPage
PDTextSelect.GetText

GetPage

 gets the page number on which the text selection is located.

Syntax

long GetPage();

Returns

the text selection's page number. The first page in a PDDoc object is page 0.

Related methods

AVDoc.ClearSelection
AVDoc.SetTextSelection
AVDoc.ShowTextSelect
AVPageView.GetPage
AVPageView.GetPageNum
PDDoc.CreateTextSelect
PDDoc.GetNumPages
PDPage.CreatePageHilite
PDPage.CreateWordHilite
PDPage.GetNumber
GetText

Gets the text from the specified element of a text selection. To obtain all the text within the text selection, use PDTextSelect.GetNumText to determine the number of elements in the text selection, then call this method in a loop to obtain each of the elements.

Syntax

BSTR GetText(long nTextIndex);

Parameters

- nTextIndex: The element of the text selection to get.

Returns

The text, or an empty string if nTextIndex is greater than the number of elements in the text selection.

Related methods

- AVDoc.ClearSelection
- AVDoc.SetTextSelection
- AVDoc.ShowTextSelect
- PDPage.CreatePageHilite
- PDDoc.CreateTextSelect
- PDPage.CreateWordHilite
- PDTextSelect.Destroy
- PDTextSelect.GetBoundingRect
- PDTextSelect.GetNumText
- PDTextSelect.GetPage
AcroExch.Point

Defines the location of an AcroPoint.

Properties

The Point object has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Gets or sets the x-coordinate of an AcroPoint.</td>
</tr>
<tr>
<td>Y</td>
<td>Gets or sets the y-coordinate of an AcroPoint.</td>
</tr>
</tbody>
</table>

X

Gets or sets the x-coordinate of an AcroPoint.

Syntax

[get/set] Short

Return

The x-coordinate of the AcroPoint.

Y

Gets or sets the y-coordinate of an AcroPoint.

Syntax

[get/set] Short

Return

The y-coordinate of the AcroPoint.

AcroExch.Rect

Defines the location of an AcroRect.

The Rect object has the following properties.
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bottom</strong></td>
<td>Gets or sets the bottom y-coordinate of an AcroRect.</td>
</tr>
<tr>
<td><strong>Left</strong></td>
<td>Gets or sets the left x-coordinate of an AcroRect.</td>
</tr>
<tr>
<td><strong>Right</strong></td>
<td>Gets or sets the right x-coordinate of an AcroRect.</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>Gets or sets the top y-coordinate of an AcroRect.</td>
</tr>
</tbody>
</table>

**Bottom**

Gets or sets the bottom y-coordinate of an AcroRect.

**Syntax**

\[\text{[get/set]} \text{ Short}\]

**Returns**

The y-coordinate of the bottom of the AcroRect.

**Left**

Gets or sets left x-coordinate of an AcroRect.

**Syntax**

\[\text{[get/set]} \text{ Short}\]

**Returns**

The x-coordinate of the left side of the AcroRect.

**Right**

Gets or sets the right x-coordinate of an AcroRect.

**Syntax**

\[\text{[get/set]} \text{ Short}\]

**Returns**

The x-coordinate of the right side of the AcroRect.
Top

Gets or sets the top y-coordinate of an AcroRect.

Syntax

[get/set] Short

Returns

The y-coordinate of the top of the AcroRect.

AcroExch.Time

 Defines a specified time, accurate to the millisecond.

Properties

The Time object has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Gets or sets the date from an AcroTime.</td>
</tr>
<tr>
<td>Hour</td>
<td>Gets or sets the hour from an AcroTime.</td>
</tr>
<tr>
<td>Millisecond</td>
<td>Gets or sets the milliseconds from an AcroTime.</td>
</tr>
<tr>
<td>Minute</td>
<td>Gets or sets the minutes from an AcroTime.</td>
</tr>
<tr>
<td>Month</td>
<td>Gets or sets the month from an AcroTime.</td>
</tr>
<tr>
<td>Second</td>
<td>Gets or sets the seconds from an AcroTime.</td>
</tr>
<tr>
<td>Year</td>
<td>Gets or sets the year from an AcroTime.</td>
</tr>
</tbody>
</table>

Date

Gets or sets the date from an AcroTime.

Syntax

[get/set] Short

Returns

The date from the AcroTime. The date runs from 1 to 31.
Hour

Gets or sets the hour from an AcroTime.

Syntax

[get/set] Short

Returns

The hour from the AcroTime. The hour runs from 0 to 23.

Millisecond

Gets or sets the milliseconds from an AcroTime.

Syntax

[get/set] Short

Returns

The milliseconds from the AcroTime. Milliseconds run from 0 to 999.

Minute

Gets or sets the minutes from an AcroTime.

Syntax

[get/set] Short

Returns

The minutes from the AcroTime. Minutes run from 0 to 59.

Month

Gets or sets the month from an AcroTime.

Syntax

[get/set] Short

Returns

The month from the AcroTime. The month runs from 1 to 12, where 1 is January and 12 is December.
Second

Gets or sets the seconds from an AcroTime.

Syntax

[get/set] Short

Returns

The seconds from the AcroTime. Seconds run from 0 to 59.

Year

Gets or sets the year from an AcroTime.

Syntax

[get/set] Short

Returns

The year from the AcroTime. The Year runs from 1 to 32767.

AxAcroPDFLib.AxAcroPDF

An object containing a set of methods that provide access to PDF browser controls. This is a creatable interface. This object makes it possible to load a file, move to various pages within the file, and specify various display and print options.

Methods

The AxAcroPDF object has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVersions</td>
<td>Deprecated</td>
</tr>
<tr>
<td>GoBackwardStack</td>
<td>Goes to the previous view on the view stack, if the previous view exists.</td>
</tr>
<tr>
<td>GoForwardStack</td>
<td>Goes to the next view on the view stack, if the next view exists.</td>
</tr>
<tr>
<td>GotoFirstPage</td>
<td>Goes to the first page in the document, maintaining the current location within the page and zoom level.</td>
</tr>
<tr>
<td>GotoLastPage</td>
<td>Goes to the last page in the document, maintaining the current location within the page and zoom level.</td>
</tr>
<tr>
<td>GotoNextPage</td>
<td>Goes to the next page in the document, if it exists. Maintains the current location within the page and zoom level.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>GotoPreviousPage</strong></td>
<td>Goes to the previous page in the document, if it exists. Maintains the</td>
</tr>
<tr>
<td></td>
<td>current location within the page and zoom level.</td>
</tr>
<tr>
<td><strong>LoadFile</strong></td>
<td>Opens and displays the specified document within the browser.</td>
</tr>
<tr>
<td><strong>Print</strong></td>
<td>Prints the document according to the options selected in a user dialog box.</td>
</tr>
<tr>
<td><strong>PrintAll</strong></td>
<td>Prints the entire document without displaying a user dialog box.</td>
</tr>
<tr>
<td><strong>PrintAllFit</strong></td>
<td>Prints the entire document without displaying a user dialog box, and the</td>
</tr>
<tr>
<td></td>
<td>pages are shrunk, if necessary, to fit into the imageable area of a page in</td>
</tr>
<tr>
<td></td>
<td>the printer.</td>
</tr>
<tr>
<td><strong>PrintPages</strong></td>
<td>Prints the specified pages without displaying a user dialog box.</td>
</tr>
<tr>
<td><strong>PrintPagesFit</strong></td>
<td>Prints the specified pages without displaying a user dialog box.</td>
</tr>
<tr>
<td><strong>PrintWithDialog</strong></td>
<td>Prints the document according to the options selected in a user dialog box.</td>
</tr>
<tr>
<td><strong>SetCurrentHighlight</strong></td>
<td>Highlights the text selection within the specified bounding rectangle on</td>
</tr>
<tr>
<td></td>
<td>the current page.</td>
</tr>
<tr>
<td><strong>SetCurrentPage</strong></td>
<td>Goes to the specified page in the document.</td>
</tr>
<tr>
<td><strong>SetLayoutMode</strong></td>
<td>Sets the layout mode for a page view according to the specified string.</td>
</tr>
<tr>
<td><strong>SetNamedDest</strong></td>
<td>Changes the page view to the named destination in the specified string.</td>
</tr>
<tr>
<td><strong>SetPageMode</strong></td>
<td>Sets the page mode according to the specified string.</td>
</tr>
<tr>
<td><strong>SetShowScrollbars</strong></td>
<td>Determines whether scrollbars will appear in the document view.</td>
</tr>
<tr>
<td><strong>SetShowToolbar</strong></td>
<td>Determines whether a toolbar will appear in the viewer.</td>
</tr>
<tr>
<td><strong>SetView</strong></td>
<td>Sets the view of a page according to the specified string.</td>
</tr>
<tr>
<td><strong>SetViewRect</strong></td>
<td>Sets the view rectangle according to the specified coordinates.</td>
</tr>
<tr>
<td><strong>SetViewScroll</strong></td>
<td>Sets the view of a page according to the specified string.</td>
</tr>
<tr>
<td><strong>SetZoom</strong></td>
<td>Sets the magnification according to the specified value.</td>
</tr>
<tr>
<td><strong>SetZoomScroll</strong></td>
<td>Sets the magnification according to the specified value, and scrolls the</td>
</tr>
<tr>
<td></td>
<td>page view both horizontally and vertically according to the specified</td>
</tr>
<tr>
<td></td>
<td>amounts.</td>
</tr>
</tbody>
</table>

**Properties**

The AxAcroPDF object has the following property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Src</strong></td>
<td>Gets or sets the URL for the document.</td>
</tr>
</tbody>
</table>
GetVersions

**Note:** Deprecated. This method is no longer available.

**Syntax**

```c
VARIANT GetVersions();
```

GoBackwardStack

Goes to the previous view on the view stack, if the previous view exists. The previous view may be in a different document.

**Syntax**

```c
void GoBackwardStack();
```

**Related methods**

AcroPDF.**GoForwardStack**

GoForwardStack

Goes to the next view on the view stack, if the next view exists. The next view may be in a different document.

**Syntax**

```c
void GoForwardStack();
```

**Related methods**

AcroPDF.**GoBackwardStack**

GotoFirstPage

Goes to the first page in the document, maintaining the current location within the page and the current zoom level.

**Syntax**

```c
void gotoFirstPage();
```

**Related methods**

AcroPDF.**GotoLastPage**
AcroPDF.**GotoNextPage**
AcroPDF.**GotoPreviousPage**
AcroPDF.SetCurrentPage

GotoLastPage

Goes to the last page in the document, maintaining the current location within the page and the current zoom level.

Syntax

void gotoLastPage();

Related methods

AcroPDF.GotoFirstPage
AcroPDF.GotoNextPage
AcroPDF.GotoPreviousPage
AcroPDF.SetCurrentPage

GotoNextPage

Goes to the next page in the document, if it exists. Maintains the current location within the page and the current zoom level.

Syntax

void gotoNextPage();

Related methods

AcroPDF.GotoFirstPage
AcroPDF.GotoLastPage
AcroPDF.GotoPreviousPage
AcroPDF.SetCurrentPage

GotoPreviousPage

Goes to the previous page in the document, if it exists. Maintains the current location within the page and the current zoom level.

Syntax

void gotoPreviousPage();
Related methods

AcroPDF.**GotoFirstPage**

AcroPDF.**GotoLastPage**

AcroPDF.**GotoNextPage**

AcroPDF.**setCurrentPage**

**LoadFile**

Opens and displays the specified document within the browser.

**Syntax**

```c
VARIANT_BOOL LoadFile(BSTR fileName);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>The path of the file to be opened.</td>
</tr>
</tbody>
</table>

**Returns**

0 if the file could not be opened, -1 otherwise.

**Print**

Prints the document according to the options selected in a user dialog box. The options include embedded printing (printing within a bounding rectangle on a given page), as well as interactive printing to a specified printer. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```c
void Print();
```

**Related methods**

AcroPDF.**PrintAll**

AcroPDF.**PrintAllFit**

AcroPDF.**PrintPages**

AcroPDF.**PrintPagesFit**

AcroPDF.**PrintWithDialog**
PrintAll

Prints the entire document without displaying a user dialog box. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```cpp
void printAll();
```

**Related methods**

- `AcroPDF.Print`
- `AcroPDF.PrintAllFit`
- `AcroPDF.PrintPages`
- `AcroPDF.PrintPagesFit`
- `AcroPDF.PrintWithDialog`

PrintAllFit

Prints the entire document without displaying a user dialog box, and the pages are shrunk, if necessary, to fit into the imageable area of a page in the printer. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```cpp
void printAllFit(VARIANT_BOOL bOn);
```

**Parameters**

| bOn       | Determines whether to scale the imageable area when printing the document. A value of 0 indicates that no scaling should be used, and a positive value indicates that the pages are shrunk, if necessary, to fit into the imageable area of a page in the printer. |

**Related methods**

- `AcroPDF.Print`
- `AcroPDF.PrintAll`
- `AcroPDF.PrintPages`
- `AcroPDF.PrintPagesFit`
- `AcroPDF.PrintWithDialog`
PrintPages

Prints the specified pages without displaying a user dialog box. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```java
void printPages( Long nFrom, Long nTo);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFrom</td>
<td>The page number of the first page to be printed. The first page in a document is page 0.</td>
</tr>
<tr>
<td>nTo</td>
<td>The page number of the last page to be printed.</td>
</tr>
</tbody>
</table>

**Related methods**

- `AcroPDF.Print`
- `AcroPDF.PrintAll`
- `AcroPDF.PrintAllFit`
- `AcroPDF.PrintPagesFit`
- `AcroPDF.PrintWithDialog`

PrintPagesFit

Prints the specified pages without displaying a user dialog box. The current printer, page settings, and job settings are used. A parameter specifies whether to shrink pages, if necessary. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```java
void printPagesFit( Long nFrom, Long nTo,
                   VARIANT_BOOL bShrinkToFit);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nFrom</td>
<td>The page number of the first page to be printed. The first page in a document is page 0.</td>
</tr>
<tr>
<td>nTo</td>
<td>The page number of the last page to be printed.</td>
</tr>
<tr>
<td>bShrinkToFit</td>
<td>Specifies whether the pages will be shrunk, if necessary, to fit into the imageable area of a page in the printer.</td>
</tr>
</tbody>
</table>
PrintWithDialog

Prints the document according to the options selected in a user dialog box. The options include embedded printing (printing within a bounding rectangle on a given page), as well as interactive printing to a specified printer. This method returns immediately, even if the printing has not completed.

**Note:** If security settings do not allow printing, this method is ignored.

**Syntax**

```csharp
void printWithDialog();
```

**Related methods**

- AcroPDF.**Print**
- AcroPDF.**PrintAll**
- AcroPDF.**PrintAllFit**
- AcroPDF.**PrintPages**
- AcroPDF.**PrintWithDialog**

SetCurrentHighlight

Highlights the text selection within the specified bounding rectangle on the current page.

**Syntax**

```csharp
void setCurrentHighlight(LONG nLeft, LONG nTop, 
                        LONG nRight, LONG nBottom);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLeft</td>
<td>The distance in points from the left side of the page.</td>
</tr>
<tr>
<td>nTop</td>
<td>The distance in points from the top of the page.</td>
</tr>
</tbody>
</table>
SetCurrentPage

Goes to the specified page in the document. Maintains the current location within the page and the current zoom level.

**Syntax**

```c
void setCurrentPage(LONG nPage);
```

**Parameters**

- **nPage**: The page number of the destination page. The first page in a document is page 0.

**Related methods**

- AcroPDF.GotoFirstPage
- AcroPDF.GotoLastPage
- AcroPDF.GotoNextPage
- AcroPDF.GotoPreviousPage

SetLayoutMode

Sets the layout mode for a page view according to the specified string.

**Syntax**

```c
void setLayoutMode(BSTR szLayoutMode);
```

**Parameters**

<table>
<thead>
<tr>
<th>szLayoutMode</th>
<th>Possible values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRight</td>
<td>The width of the bounding rectangle.</td>
</tr>
<tr>
<td>nBottom</td>
<td>The height of the bounding rectangle.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>szLayoutMode</th>
<th>Possible values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DontCare</td>
<td>use the current user preference</td>
</tr>
<tr>
<td>SinglePage</td>
<td>use single page mode (as it would have appeared in pre-Acrobat DC 3.0 viewers)</td>
</tr>
<tr>
<td>OneColumn</td>
<td>use one-column continuous mode</td>
</tr>
<tr>
<td>TwoColumnLeft</td>
<td>use two-column continuous mode with the first page on the left</td>
</tr>
<tr>
<td>TwoColumnRight</td>
<td>use two-column continuous mode with the first page on the right</td>
</tr>
</tbody>
</table>
Related methods

AcroPDF. SetNamedDest
AcroPDF. SetView
AcroPDF. SetViewRect
AcroPDF. SetViewScroll

SetNamedDest

Changes the page view to the named destination in the specified string.

Syntax

void setNamedDest(BSTR szNamedDest);

Parameters

| szNamedDest | The named destination to which the viewer will go. |

Related methods

AcroPDF. SetLayoutMode
AcroPDF. SetView
AcroPDF. SetViewRect
AcroPDF. SetViewScroll

SetPageMode

Sets the page mode according to the specified string.

Syntax

void setPageMode(BSTR szPageMode);

Parameters

<table>
<thead>
<tr>
<th>szPageMode</th>
<th>Possible values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none — displays the document, but does not display bookmarks or thumbnails (default)</td>
</tr>
<tr>
<td></td>
<td>bookmarks — displays the document and bookmarks</td>
</tr>
<tr>
<td></td>
<td>thumbs — displays the document and thumbnails</td>
</tr>
</tbody>
</table>
Related methods

AcroPDF.SetShowScrollbars
AcroPDF.SetShowToolbar

SetShowScrollbars

Determines whether scrollbars will appear in the document view.

Syntax

void setShowScrollbars(VARIANT_BOOL bOn);

Parameters

bOn A positive value indicates that scrollbars will appear, 0 indicates that they will not.

Related methods

AcroPDF.SetPageMode
AcroPDF.SetShowToolbar

SetShowToolbar

Determines whether a toolbar will appear in the viewer.

Syntax

void setShowToolbar(VARIANT_BOOL bOn);

Parameters

bOn A positive value indicates that the toolbar will appear, 0 indicates that it will not.

Related methods

AcroPDF.SetPageMode
AcroPDF.SetShowScrollbars

SetView

Sets the view of a page according to the specified string.

Syntax

void setView(BSTR szViewMode);
Parameters

<table>
<thead>
<tr>
<th>szViewMode</th>
<th>Possible values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fit</td>
<td>Fits the entire page within the window both vertically and horizontally.</td>
</tr>
<tr>
<td>FitH</td>
<td>Fits the entire width of the page within the window.</td>
</tr>
<tr>
<td>FitV</td>
<td>Fits the entire height of the page within the window.</td>
</tr>
<tr>
<td>FitB</td>
<td>Fits the bounding box within the window both vertically and horizontally.</td>
</tr>
<tr>
<td>FitBH</td>
<td>Fits the entire width of the bounding box within the window.</td>
</tr>
<tr>
<td>FitB</td>
<td>Fits the entire height of the bounding box within the window.</td>
</tr>
</tbody>
</table>

Related methods

- `AcroPDF.SetLayoutMode`
- `AcroPDF.SetNamedDest`
- `AcroPDF.SetViewRect`
- `AcroPDF.SetViewScroll`

**SetViewRect**

Sets the view rectangle according to the specified coordinates.

**Syntax**

```c
void setViewRect(FLOAT left, FLOAT top,
                  FLOAT width, FLOAT height);
```

**Parameters**

- `left` The upper left horizontal coordinate.
- `top` The vertical coordinate in the upper left corner.
- `width` The horizontal width of the rectangle.
- `height` The vertical height of the rectangle.

Related methods

- `AcroPDF.SetLayoutMode`
- `AcroPDF.SetNamedDest`
- `AcroPDF.SetView`
- `AcroPDF.SetViewScroll`
SetViewScroll

Sets the view of a page according to the specified string. Depending on the view mode, the page is either scrolled to the right or scrolled down by the amount specified in `offset`.

Syntax

```c
void setViewRect(BSTR szViewMode, FLOAT offset);
```

Parameters

- `szViewMode`: Possible values:
  - Fit — Fits the entire page within the window both vertically and horizontally.
  - FitH — Fits the entire width of the page within the window.
  - FitV — Fits the entire height of the page within the window.
  - FitB — Fits the bounding box within the window both vertically and horizontally.
  - FitBH — Fits the entire width of the bounding box within the window.
  - FitBV — Fits the entire height of the bounding box within the window.

- `offset`: The horizontal or vertical coordinate positioned either at the left or top edge.

Related methods

- `AcroPDF.SetLayoutMode`
- `AcroPDF.SetNamedDest`
- `AcroPDF.SetView`
- `AcroPDF.SetViewRect`

SetZoom

Sets the magnification according to the specified value.

Syntax

```c
void setZoom(FLOAT percent);
```

Parameters

- `percent`: The desired zoom factor, expressed as a percentage. For example, 1.0 represents a magnification of 100%.

Related methods

- `AcroPDF.SetZoomScroll`
SetZoomScroll

Sets the magnification according to the specified value, and scrolls the page view both horizontally and vertically according to the specified amounts.

Syntax

void setZoomScroll(FLOAT percent, FLOAT left, FLOAT top);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>percent</td>
<td>The desired zoom factor, expressed as a percentage. For example, 1.0 represents a magnification of 100%.</td>
</tr>
<tr>
<td>left</td>
<td>The horizontal coordinate positioned at the left edge.</td>
</tr>
<tr>
<td>top</td>
<td>The vertical coordinate positioned at the top edge.</td>
</tr>
</tbody>
</table>

Related methods

AcroPDF.SetZoom

Src

Gets or sets the URL for the document.

Syntax

[get/set] src

Returns

The URL for the document, formatted as a string.
DDE Messages

This chapter lists all DDE messages supported by Acrobat DC.

These DDE messages handle the display of the Acrobat DC application:

- AppExit
- AppHide
- AppShow
- CloseAllDocs
- HideToolbar
- MenuitemExecute
- ShowToolbar

These DDE messages control the display of the document:

- DocClose
- DocDeletePages
- DocInsertPages
- DocOpen
- DocReplacePages
- DocSave
- DocSaveAs
- DocSetViewMode
- FileOpen
- FileOpenEx

These DDE messages handle printing of a document:

- DocPrint
- FilePrint
- FilePrintEx
- FilePrintSilent
- FilePrintSilentEx
- FilePrintTo
- FilePrintToEx

These DDE messages control the view of a document:

- DocGoTo
- DocGoToNameDest
- DocPageDown
- DocPageLeft
This DDE message is used for searching:

- **DocFind**

Acrobat DC Reader supports the following subset of DDE messages:

- **AppExit**
- **CloseAllDocs**
- **DocClose**
- **DocGoTo**
- **DocGoToNameDest**
- **DocOpen**
- **FileOpen**
- **FileOpenEx**
- **FilePrint**
- **FilePrintEx**
- **FilePrintSilent**
- **FilePrintSilentEx**
- **FilePrintTo**
- **FilePrintToEx**

### AppExit

Exits the Acrobat DC application.

**AppExit** is also supported in Acrobat DC Reader.

#### Syntax

```
[AppExit()]
```

#### Returns

*true* if the Acrobat DC application exits successfully, *false* otherwise.

#### Related methods

- **AppHide**
- **AppShow**
AppHide

Iconifies or hides the Acrobat DC application.

Syntax

[AppHide()]

Returns

true if the Acrobat DC application is hidden successfully, false otherwise.

Related methods

AppExit
AppShow

AppShow

Shows the Acrobat DC application.

Syntax

[AppShow()]

Returns

true if the Acrobat DC application is shown successfully, false otherwise.

Related methods

AppExit
AppHide

CloseAllDocs

Closes all open documents.

CloseAllDocs is also supported in Acrobat DC Reader.

Syntax

[CloseAllDocs()]

Returns

true if the documents are closed successfully, false otherwise.
Related methods

**DocClose**

Closes the specified document without saving it, and without prompting the user to save the document if it has been modified.

**DocClose** is also supported in Acrobat DC Reader.

**Syntax**

```
[DocClose(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be closed.</td>
</tr>
</tbody>
</table>

**Returns**

true if the document is closed successfully, false if the document does not exist or is not closed successfully.

**Related methods**

**CloseAllDocs**

**DocOpen**

**FileOpen**

---

**DocDeletePages**

Deletes the specified pages in the document. Requests to delete all pages in a document are ignored because a document must have at least one page.

**Syntax**

```
[DocDeletePages(char* fullPath, long fromPage, long toPage)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the document.</td>
</tr>
</tbody>
</table>
Returns

true if the pages are deleted successfully. Returns false if the document specified by fullPath does not exist, if the request was to delete all the document’s pages, or if the pages are not deleted successfully.

Related methods

DocInsertPages
DocReplacePages

DocFind

Finds a string in a specified file. This does not use a cross-document search, but instead performs a page-by-page search of the specified file.

Syntax

[DocFind(char* fullPath, char* string, boolean caseSensitive, boolean wholeWords, boolean bReset)]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be searched.</td>
</tr>
<tr>
<td>string</td>
<td>The string to be found.</td>
</tr>
<tr>
<td>caseSensitive</td>
<td>true if the search is case-sensitive, false otherwise.</td>
</tr>
<tr>
<td>wholeWords</td>
<td>true if the search will only match whole words, false otherwise.</td>
</tr>
<tr>
<td>bReset</td>
<td>true if the search begins on the first page of the document, false if the search begins on the current page.</td>
</tr>
</tbody>
</table>

Returns

false if the document specified by fullPath does not exist or if the text is not found, true otherwise.

DocGoTo

Goes to the specified page.

DocGoTo is also supported in Acrobat DC Reader.
### Syntax

```
[DocGoTo(char* fullPath, long pageNum)]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file.</td>
</tr>
<tr>
<td>pageNum</td>
<td>The page number of the destination page.</td>
</tr>
</tbody>
</table>

### Returns

- false if the document specified by `fullPath` does not exist, true otherwise.

### DocGoToNameDest

**Goes to the specified named destination.**

`DocGoToNameDest` is also supported in Acrobat DC Reader.

#### Syntax

```
[DocGoToNameDest(char* fullPath, char* nameDest)]
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file.</td>
</tr>
<tr>
<td>nameDest</td>
<td>The named destination.</td>
</tr>
</tbody>
</table>

#### Returns

- false if the document specified by `fullPath` does not exist, true otherwise.

### DocInsertPages

**Inserts pages from one file into another.**

#### Syntax

```
[DocInsertPages(char* fullPath, long insertAfterPage, char* sourcePath)]
```
Parameters

- **fullPath**: The full path of the target document, which must already be open in the Acrobat DC application.

- **insertAfterPage**: The page number after which pages are being inserted. Possible values can be a page number or one of the following:
  - PDBeforeFirstPage — Pages are inserted at the beginning of the document.
  - PDLastPage — Pages are inserted at the end of the document.

- **sourcePath**: The full path of the source document. This file need not be open in the Acrobat DC application.

Returns

- `true` if the pages are inserted successfully, `false` if the document does not exist or the pages are not inserted successfully.

Related methods

- [DocDeletePages](#)
- [DocReplacePages](#)

**DocOpen**

Opens a document and adds it to the list of documents known to DDE, allowing it to be manipulated by other DDE messages (see [FileOpen](#)).

DocOpen is also supported in Acrobat DC Reader.

**Syntax**

```
[DocOpen(char* fullPath)]
```

**Parameters**

- **fullPath**: The full path of the file to be opened.

**Returns**

- `true` if the file is opened successfully, `false` otherwise.

**Related methods**

- [CloseAllDocs](#)
- [DocClose](#)
DocPageDown

Scrolls forward through the document by one screen area.

Syntax

[DocPageDown(char* fullPath)]

Parameters

| fullPath   | The full path of the document. |

Returns

false if the document specified by fullPath does not exist, true otherwise.

Related methods

DocPageLeft
DocPageRight
DocPageUp
DocScrollTo

DocPageLeft

Scrolls to the left by a small amount.

Syntax

[DocPageLeft(char* fullPath)]

Parameters

| fullPath   | The full path of the document. |

Returns

false if the document specified by fullPath does not exist, true otherwise.

Related methods

DocPageDown
**DocPageRight**

Scrolls to the right by a small amount.

**Syntax**

```c
[DocPageRight(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the document.</td>
</tr>
</tbody>
</table>

**Returns**

false if the document specified by `fullPath` does not exist, true otherwise.

**Related methods**

- [DocPageDown](#)
- [DocPageLeft](#)
- [DocPageUp](#)

**DocPageUp**

Scrolls backward through the document by one screen area.

**Syntax**

```c
[DocPageUp(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the document.</td>
</tr>
</tbody>
</table>

**Returns**

false if the document specified by `fullPath` does not exist, true otherwise.
Related methods

- DocPageDown
- DocPageLeft
- DocPageRight
- DocScrollTo

DocPrint

Prints a specified range of pages from a document, without displaying any modal Print dialog box to the user. For PostScript printing, only Level 1 operators are used, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

Syntax

```
[DocPrint(char* fullPath, long startPage, long endPage)]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of document.</td>
</tr>
<tr>
<td>startPage</td>
<td>The page number of the first page to be printed.</td>
</tr>
<tr>
<td>endPage</td>
<td>The page number of the last page to be printed.</td>
</tr>
</tbody>
</table>

Returns

false if the document specified by fullPath does not exist, true otherwise.

Related methods

- FilePrint
- FilePrintSilent
- FilePrintTo

DocReplacePages

Replaces pages in the target document using the specified pages from the source document.

Syntax

```
[DocReplacePages(char* fullPath, long startDestPage,
                 char* sourcePath, long startSourcePage,
                 long endSourcePage)]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the target document. This file must already be open in the Acrobat DC application.</td>
</tr>
<tr>
<td>startDestPage</td>
<td>The page number of the first page in the target document to be replaced.</td>
</tr>
<tr>
<td>sourcePath</td>
<td>The full path of the source document. This file does not have to be already open in the Acrobat DC application.</td>
</tr>
<tr>
<td>startSourcePage</td>
<td>The page number of the first page in the source document to use as a replacement page.</td>
</tr>
<tr>
<td>endSourcePage</td>
<td>The page number of the last page in the source document to use as a replacement page.</td>
</tr>
</tbody>
</table>

Returns

*true* if the pages are replaced successfully. *false* if the document does not exist or the pages are not replaced successfully.

Related methods

- [DocDeletePages](#)
- [DocInsertPages](#)

**DocSave**

Saves the specified file. The user is not warned if there are any problems saving the file.

Syntax

```c
[DocSave(char* fullPath)]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be saved.</td>
</tr>
</tbody>
</table>

Returns

*true* if the document is saved successfully, *false* if the document does not exist or is not saved successfully.

Related methods

- [DocSaveAs](#)
DocSaveAs

Saves an open file to a new path. The user is not warned if there are any problems saving the file.

**Syntax**

```c
[DocSaveAs(char* fullPath, char* newPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the existing file.</td>
</tr>
<tr>
<td>newPath</td>
<td>The full path of the new file.</td>
</tr>
</tbody>
</table>

**Returns**

*true* if the document is saved successfully, *false* if the document does not exist or is not saved successfully.

**Related methods**

[DocSave]

DocScrollTo

Scrolls the view of the current page to the specified location.

**Syntax**

```c
[DocScrollTo(char* fullPath, int x, int y)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the document.</td>
</tr>
<tr>
<td>x</td>
<td>The destination’s x-coordinate.</td>
</tr>
<tr>
<td>y</td>
<td>The destination’s y-coordinate.</td>
</tr>
</tbody>
</table>

**Returns**

*false* if the document specified by *fullPath* does not exist, *true* otherwise.

**Related methods**

[DocPageDown]

[DocPageLeft]

[DocPageRight]
**DocSetViewMode**

Determines whether bookmarks, thumbnail images, or neither are shown in addition to the document.

**Syntax**

```
[DocSetViewMode(char* fullPath, char* viewType)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the document.</td>
</tr>
<tr>
<td>viewType</td>
<td>The view mode to be used. Must be one of the following: PDUseThumbs — Displays pages and thumbnail images. PDUseNone — Displays only pages. PDUseBookmarks — Displays pages and bookmarks.</td>
</tr>
</tbody>
</table>

**Returns**

true if the view mode is set successfully, false if the document specified by fullPath does not exist or an unknown view mode is specified.

**Related methods**

- FullMenus
- ShortMenus

**DocZoomTo**

Sets the zoom for a specified document.

**Syntax**

```
[DocZoomTo(char* fullPath, char* zoomType, int scale)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file whose zoom to set.</td>
</tr>
</tbody>
</table>
Returns

false if the document specified by fullPath does not exist, or if zoomType has an unknown value. Returns true otherwise.

FileOpen

Opens and displays the specified document. If the file is already open, it becomes the active document and appears in the front. This DDE message does not add the document to the list that can be manipulated using DDE messages; use DocOpen to do that.

FileOpen is also supported in Acrobat DC Reader.

Syntax

[FileOpen(char* fullPath)]

Parameters

fullPath The full path of the file to be opened.

Returns

ture if the file is opened successfully, false otherwise.

Related methods

CloseAllDocs

DocClose

DocOpen

FileOpenEx

Opens and displays a file. If the file is already open, it becomes the active document and appears in the front. This DDE message does not add the document to the list that can be manipulated using DDE messages; use DocOpen to do that.
This method allows documents that either take a long time to open or are password-protected to open without stopping the flow of DDE messages. Documents opened with `FileOpenEx` are opened during an idle period. This is useful in situations in which several DDE messages are sent at once, such as a multiple file select from Windows Explorer.

`FileOpenEx` is also supported in Acrobat DC Reader.

**Syntax**

```c
[FileOpenEx(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>fullPath</th>
<th>The full path of the file to be opened.</th>
</tr>
</thead>
</table>

**Returns**

`true` is always returned. The specified file may not actually open.

**Related methods**

- `FileOpen`
- `CloseAllDocs`
- `DocClose`
- `DocOpen`

**FilePrint**

Prints all pages in a document, displaying a modal print dialog box to the user. For PostScript printing, only Level 1 operators are used, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

`FilePrint` is also supported in Acrobat DC Reader.

**Syntax**

```c
[FilePrint(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>fullPath</th>
<th>The full path of the file to be printed.</th>
</tr>
</thead>
</table>

**Returns**

`false` if the document specified by `fullPath` does not exist, `true` otherwise.
Related methods

FilePrintSilent
FilePrintTo

FilePrintEx

Prints all pages in a document, displaying a modal print dialog box to the user. For PostScript printing, only Level 1 operators are used, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

Similar to FileOpenEx, this is a special DDE command that returns true right away and performs the action during idle periods. This ensures that no DDE commands are lost when printing a large number of files simultaneously.

FilePrintEx is also supported in Acrobat DC Reader.

Syntax

[FilePrintEx(char* fullPath)]

Parameters

fullPath The full path of the file to print.

Returns

ture is always returned.

Related methods

DocPrint
FileOpenEx
FilePrint
FilePrintSilent
FilePrintSilentEx
FilePrintTo
FilePrintToEx
FilePrintSilent

Prints all pages in a document, without displaying a print dialog box to the user. For PostScript printing, only Level 1 operators are used, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

FilePrintSilent is also supported in Acrobat DC Reader.

Syntax

[FilePrintSilent(char* fullPath)]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be printed.</td>
</tr>
</tbody>
</table>

Returns

false if the document specified by fullPath does not exist, true otherwise.

Related methods

DocPrint
FilePrint
FilePrintTo

FilePrintSilentEx

Prints all pages in a document, without displaying a print dialog box to the user. For PostScript printing, only Level 1 operators are used, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

Similar to FileOpenEx, this is a DDE command that returns true right away and does the action during idle periods. This is to ensure that no DDE commands are lost when printing a large number of files simultaneously.

FilePrintSilentEx is also supported in Acrobat DC Reader.

Syntax

[FilePrintSilentEx(char* fullPath)]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be printed.</td>
</tr>
</tbody>
</table>
Returns

true is always returned.

Related methods

DocPrint
FileOpenEx
FilePrintEx
FilePrintSilent
FilePrintTo
FilePrintToFEx

FilePrintTo

Prints all pages in a document to a specified printer, using a specified driver and port, displaying a modal print dialog box to the user. For PostScript printing, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

FilePrintTo is also supported in Acrobat DC Reader.

Syntax

[FilePrintTo(char* fullPath, char* printName,
            char* driverName, char* portName)]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be printed.</td>
</tr>
<tr>
<td>printName</td>
<td>The name of the printer. Required for Windows 95 and later.</td>
</tr>
<tr>
<td>driverName</td>
<td>Printer driver name.</td>
</tr>
<tr>
<td>portName</td>
<td>Port name. Required for Windows NT.</td>
</tr>
</tbody>
</table>

Returns

false if the document specified by fullPath does not exist, true otherwise.

Related methods

DocPrint
FilePrint
FilePrintSilent
**FilePrintToEx**

Prints all pages in a document to a specified printer, using a specified driver and port, displaying a modal print dialog box to the user. For PostScript printing, only ASCII data is generated, and the document’s pages are not shrunk to fit into the imageable area of the printed page.

Similar to **FileOpenEx**, this is a DDE command that returns `true` right away and does the action during idle periods. This is to ensure that no DDE commands are lost when printing a large number of files simultaneously.

**FilePrintToEx** is also supported in Acrobat DC Reader.

**Syntax**

```c
[FilePrintToEx(char* fullPath, char* printName, char* driverName, char* portName)]
```

**Parameters**

- `fullPath` - The full path of the file to be printed.
- `printName` - The name of the printer. Required for Windows 95 and later.
- `driverName` - Printer driver name.
- `portName` - Port name. Required for Windows NT.

**Returns**

`true` is always returned.

**Related methods**

- **DocPrint**
- **FileOpenEx**
- **FilePrintEx**
- **FilePrintSilentEx**
- **FilePrintTo**
- **FilePrintToEx**

**FullMenus**

Displays full menus, and sets this option in the Acrobat DC application’s preferences file.

With Acrobat DC 3.0 or later, all menus are displayed, and this function is ignored.
**Syntax**

```c
[FullMenus()]
```

**Returns**

`true` if full menus are set successfully, `false` otherwise.

**Related methods**

- `DocSetViewMode`
- `ShortMenus`

---

**HideToolbar**

Hides the toolbar.

**Syntax**

```c
[HideToolbar()]
```

**Returns**

`true` if the toolbar is hidden successfully, `false` otherwise.

**Related methods**

- `ShowToolbar`

---

**MenuitemExecute**

Executes the menu item specified by its language-independent name.

**Syntax**

```c
[MenuitemExecute(char* menuItemName)]
```

**Parameters**

| menuItemName | The language-independent name of the menu item to execute. See the [Acrobat and PDF Library API Reference](https://www.adobe.com/products/acrobat/pdf-library-api-reference.html) for a list of menu item names. |

---

**ShortMenus**

Displays short menus, and sets this option in the Acrobat DC application's preferences file.

With Acrobat DC 3.0 or later, all menus are displayed, and this function is ignored.
Syntax
[ShortMenus()]

Returns
true if short menus are set successfully, false otherwise.

Related methods
DocSetViewMode
FullMenus

ShowToolbar

Shows the toolbar.

Syntax
[ShowToolbar()]

Returns
true if the toolbar is shown successfully, false otherwise.

Related methods
HideToolbar
This chapter describes the supported Apple event objects, with descriptions of each object’s elements and properties, and the supported Apple events.

Objects

Acrobat DC presents the following objects to the Apple event interface:

- annotation
- application
- bookmark
- conversion
- document
- Link Annotation
- menu
- menu item
- page
- PDF Window
- Text Annotation

annotation

An annotation on a page in a PDF file that corresponds to PDAnnot, an internal Acrobat DC class. This object was formerly known as PDAnnot.

Acrobat DC also has two built-in annotation objects. For more information, see “Link Annotation” on page 179 and “Text Annotation” on page 184.

Plural form

Annotations

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>bounds</td>
<td>a list of small real</td>
<td>The boundary rectangle for the annotation in PDF space (left, top, right, bottom).</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>Property</td>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>color</td>
<td>'RGB'</td>
<td>The color of the border around the annotation.</td>
</tr>
<tr>
<td>contents</td>
<td>international text</td>
<td>Text annotations only. The textual contents of the note.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>destination page</td>
<td>integer</td>
<td>Link annotations only. The page number to appear in the PDF window when the annotation link is activated.</td>
</tr>
<tr>
<td>rectangle</td>
<td>a list of small real</td>
<td>Link annotations only. The boundary rectangle (specified in user space) for the view of the destination. Coordinates are specified in the following order: left, top, right, bottom.</td>
</tr>
<tr>
<td>fit type</td>
<td>constant</td>
<td>Link annotations only. Determines how the destination rectangle is fitted to the window when the link is activated. Values are: Left Top Zoom, Fit Page, Fit Width, Fit Height, Fit Rect, Fit BBox, Fit BB Width, Fit BB Height. These are described in the PDF Reference.</td>
</tr>
<tr>
<td>index</td>
<td>integer [r/o]</td>
<td>The annotation's index within the page object.</td>
</tr>
<tr>
<td>modification date</td>
<td>date</td>
<td>The date and time the annotation was last modified.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Text annotations only. The annotation's label.</td>
</tr>
<tr>
<td>open state</td>
<td>Boolean</td>
<td>Text annotations only. Whether the annotation is open.</td>
</tr>
<tr>
<td>subtype</td>
<td>international text [r/o]</td>
<td>The subtype of the annotation.</td>
</tr>
<tr>
<td>zoom factor</td>
<td>small real</td>
<td>Link annotations only. If fit type is Left Top Zoom, this specifies the zoom factor; otherwise it is ignored. Setting this property automatically sets fit type to Left Top Zoom.</td>
</tr>
</tbody>
</table>

**Related methods**

- [delete](#)
- [perform](#)

**application**

The Acrobat DC or Acrobat DC Reader application itself.
## Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Accessed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>name, numeric index</td>
</tr>
<tr>
<td>PDF Window</td>
<td>name, numeric index</td>
</tr>
<tr>
<td>menu</td>
<td>name, numeric index</td>
</tr>
<tr>
<td>menu item</td>
<td>name</td>
</tr>
</tbody>
</table>

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active doc</td>
<td>reference</td>
<td>The active document.</td>
</tr>
<tr>
<td>active tool</td>
<td>international text</td>
<td>The type of the currently active tool. See the Acrobat and PDF Library API Reference for a list of tool names.</td>
</tr>
<tr>
<td>anti_alias text</td>
<td>Boolean</td>
<td>Determines whether to anti-alias text and monochrome images.</td>
</tr>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>case sensitivity</td>
<td>Boolean</td>
<td>Determines whether searches are case-sensitive.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>default zoom factor</td>
<td>small real</td>
<td>The default zoom factor, in percent, used for displaying new documents. For example, a value of 100 corresponds to a zoom factor of 1.0 (100%).</td>
</tr>
<tr>
<td>default zoom type</td>
<td>constant</td>
<td>The default zoom type when opening a new document. Valid values are no vary, fit page, fit width, fit height, and fit visible width.</td>
</tr>
<tr>
<td>download entire file</td>
<td>Boolean</td>
<td>Determines whether to download the entire file.</td>
</tr>
<tr>
<td>frontmost</td>
<td>Boolean</td>
<td>Determines whether Acrobat DC is the frontmost application. Value can be set to true only.</td>
</tr>
<tr>
<td>fullscreen click advances</td>
<td>Boolean</td>
<td>Determines whether mouse click advances in fullscreen mode.</td>
</tr>
<tr>
<td>Property</td>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fullscreen cursor</td>
<td>Boolean</td>
<td>Determines whether to hide the cursor in fullscreen mode.</td>
</tr>
<tr>
<td>fullscreen escape</td>
<td>Boolean</td>
<td>Determines whether the Esc key can be used to exit fullscreen mode.</td>
</tr>
<tr>
<td>fullscreen loop</td>
<td>Boolean [r/o]</td>
<td>Determines whether the document’s pages are displayed in a loop while in fullscreen mode.</td>
</tr>
<tr>
<td>fullscreen timer delay</td>
<td>integer</td>
<td>The number of seconds to advance to the next page in fullscreen mode.</td>
</tr>
<tr>
<td>fullscreen transition</td>
<td>international text [r/o]</td>
<td>Default fullscreen transition.</td>
</tr>
<tr>
<td>highlight color</td>
<td>'RGB '</td>
<td>Color used to highlight selections.</td>
</tr>
<tr>
<td>maximum documents</td>
<td>integer [r/o]</td>
<td>Maximum number of open documents.</td>
</tr>
<tr>
<td>name</td>
<td>string [r/o]</td>
<td>The application’s name.</td>
</tr>
<tr>
<td>note color</td>
<td>'RGB '</td>
<td>A list of three values between 0 and 65535 representing the color of the border around text annotations. For example, the following sets the note color to deep blue: set the note color to {0, 0, 32768}.</td>
</tr>
<tr>
<td>note font name</td>
<td>international text</td>
<td>Deprecated.</td>
</tr>
<tr>
<td>note font size</td>
<td>integer</td>
<td>Deprecated.</td>
</tr>
<tr>
<td>open in place</td>
<td>Boolean</td>
<td>Determines whether to open cross-document links in the same window.</td>
</tr>
<tr>
<td>page layout</td>
<td>international text</td>
<td>Default page layout. Values are: Single Page, Continuous, Facing, and Continuous - Facing.</td>
</tr>
<tr>
<td>page units</td>
<td>international text</td>
<td>Default page display units: Points, Inches or Millimeters.</td>
</tr>
<tr>
<td>PS level</td>
<td>integer</td>
<td>Deprecated. Set the PostScript level when using save or print_pages commands.</td>
</tr>
<tr>
<td>save as linearize</td>
<td>Boolean</td>
<td>Determines whether to save the document as optimized for the web.</td>
</tr>
<tr>
<td>show splash at startup</td>
<td>Boolean</td>
<td>Determines whether the splash screen is shown at startup.</td>
</tr>
<tr>
<td>skip warnings</td>
<td>Boolean</td>
<td>Determines whether to skip warning dialog boxes during program execution.</td>
</tr>
</tbody>
</table>
### Related methods

- `close all docs`
- `count`
- `make`
- `open`
- `print`
- `quit`
- `run`

#### AVPageView

**Note:** Deprecated. Use `PDF Window` instead.

### bookmark

A bookmark on a page in a PDF file. Corresponds to Acrobat DC’s `PDBbookmark` object.

**Note:** This object was formerly known as `PDBbookmark`.

#### Plural form

Bookmarks
## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>destination page number</td>
<td>integer</td>
<td>The page number to which the PDF Window goes when the bookmark’s action is performed.</td>
</tr>
<tr>
<td>destination rectangle</td>
<td>list of small real</td>
<td>Boundary rectangle (specified in user space) for the view of the destination when the bookmark’s action is performed. Coordinates are specified in the following order: (left, top, right, bottom). <strong>Note:</strong> Set this only after setting fit type.</td>
</tr>
<tr>
<td>fit type</td>
<td>constant</td>
<td>Controls how the destination rectangle is fitted to the window when the bookmark’s action is performed. Possible values: Left Top Zoom — Sets a specified zoom and a specified location on the page. Fit Page — Sets the zoom factor so that the entire page fits into the window. Fit Width — Sets the zoom factor so that the width of the page fits into the window. Fit Height — Sets the zoom factor so that the height of the page fits into the window. Fit Rect — Sets the zoom factor so that the specified rectangle fits into the window. Fit BBox — Sets the zoom so that the rectangle enclosing all marks on the page (known as the bounding box) fits into the window. Fit BB Width — Sets the zoom factor so that the width of the bounding box fits into the window. Fit BB Height — Sets the zoom factor so that the height of the bounding box fits into the window.</td>
</tr>
<tr>
<td>index</td>
<td>integer [r/o]</td>
<td>The bookmark’s index within the document.</td>
</tr>
<tr>
<td>name</td>
<td>international text</td>
<td>The bookmark’s title.</td>
</tr>
<tr>
<td>zoom factor</td>
<td>small real</td>
<td>The zoom factor used when fit type is Left Top Zoom; ignored otherwise. Setting this property automatically sets fit type to Left Top Zoom.</td>
</tr>
</tbody>
</table>
Related methods

- convert

conversion

A file type converter that exports PDF files into other formats. Conversions correspond to the list of formats specified in the Acrobat DC Save As menu. A list of formats can be obtained as follows:

- get every conversion

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>index</td>
<td>integer [r/o]</td>
<td>The index number of the converter.</td>
</tr>
<tr>
<td>name</td>
<td>international text</td>
<td>The conversion's description.</td>
</tr>
</tbody>
</table>

Related methods

- save

document

Represents a single open document in Acrobat DC or Acrobat DC Reader.

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Accessed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>Numeric index. The first page in a document is page 1.</td>
</tr>
<tr>
<td>bookmark</td>
<td>Name or numeric index.</td>
</tr>
<tr>
<td>PDF Window</td>
<td>An index of 1 or with the some keyword in AppleScript. No document has more than one PDF Window.</td>
</tr>
</tbody>
</table>

Plural form
documents
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>bounds</td>
<td>bounding rectangle [r/o]</td>
<td>The boundary rectangle for the document’s window, in screen coordinates (left, top, right, bottom).</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>file alias</td>
<td>alias [r/o]</td>
<td>An alias for the file to which the document will be saved if no other name is specified; this is usually the same path from which the document was read.</td>
</tr>
<tr>
<td>modified</td>
<td>Boolean [r/o]</td>
<td>Determines whether the document has been modified and should be saved.</td>
</tr>
<tr>
<td>name</td>
<td>international text [r/o]</td>
<td>The document’s name as it appears in the window’s titlebar.</td>
</tr>
<tr>
<td>view mode</td>
<td>constant</td>
<td>The viewing mode of the document. Possible values: just pages, pages and thumbs, or pages and bookmarks.</td>
</tr>
</tbody>
</table>

Related methods

- bring to front
- clear selection
- close
- count
- create thumbs
- delete
- delete pages
- delete thumbs
- find next note
- find text
- get info
- insert pages
- maximize
- print pages
replace_pages
save
set_info

**EPS Conversion**

A file type converter that exports PDF files into EPS format.

**Properties**

Inherits from `PostScript_Conversion`.

**Related methods**

save

**Link Annotation**

A link annotation on a page in a PDF file. Can only be used as the target of a `make` event. All other access is via the `annotation` class.

**Note:** This object was formerly known as `PDLinkAnnot`.

**Properties**

Inherits from `annotation`.

**Related methods**

delete
perform

**menu**

A menu in the Acrobat DC or Acrobat DC Reader menu bar.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Accessed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu_item</td>
<td>name, numeric index.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>name</td>
<td>international text [r/o]</td>
<td>The menu’s name (a language-independent name that uniquely identifies the menu). See the Acrobat and PDF Library API Reference for a list of menu names.</td>
</tr>
<tr>
<td>title</td>
<td>string [r/o]</td>
<td>The menu’s title as it would appear in the user interface.</td>
</tr>
</tbody>
</table>

Related methods

execute

menu item

A menu item contained within a menu in Acrobat DC or Acrobat DC Reader.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>enabled</td>
<td>Boolean [r/o]</td>
<td>Determines whether the menu item is enabled.</td>
</tr>
<tr>
<td>has submenu</td>
<td>Boolean [r/o]</td>
<td>Determines whether the menu item has a hierarchical sub-menu.</td>
</tr>
<tr>
<td>marked</td>
<td>Boolean [r/o]</td>
<td>Determines whether the menu item is checked.</td>
</tr>
<tr>
<td>name</td>
<td>international text [r/o]</td>
<td>The menu item’s language-independent name. See the Acrobat and PDF Library API Reference for a list of menu item names.</td>
</tr>
<tr>
<td>title</td>
<td>string [r/o]</td>
<td>The menu’s title as it would appear in the user interface.</td>
</tr>
</tbody>
</table>
Related methods

**execute**

**page**

A single page in the PDF representation of a document. Corresponds to Acrobat DC's internal `PDPage` object.

**Note:** This object was formerly known as `PDPage`.

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Accessed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>numeric index.</td>
</tr>
</tbody>
</table>

Plural form

Pages

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>bounds</td>
<td>list of small real</td>
<td>The boundary rectangle for the page in user space (left, top, right, bottom).</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>page number</td>
<td>integer [r/o]</td>
<td>The page’s number. The first page in a document is page 1.</td>
</tr>
<tr>
<td>rotation</td>
<td>integer</td>
<td>The rotation angle of the page in degrees (0, 90, 180, or 270).</td>
</tr>
</tbody>
</table>

Related methods

**delete pages**

**insert pages**

**replace pages**

**goto**

**move**
### PDAnnot

**Note:** Deprecated. Use annotation instead.

### PDBookMark

**Note:** Deprecated. Use bookmark instead.

### PDLinkAnnot

**Note:** Deprecated. Use Link Annotation instead.

### PDPage

**Note:** Deprecated. Use page instead.

### PDTextAnnot

**Note:** Deprecated. Use Text Annotation instead.

### PDF Window

The area of the Acrobat DC or Acrobat DC Reader window that displays the contents of a page within the document. Corresponds to the Acrobat DC internal AvPageView object. A document that is not visible does not have a PDF Window.

**Note:** This object was formerly known as AVPageView.

### Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Accessed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>numeric index. The first page in a document is page 1.</td>
</tr>
</tbody>
</table>

### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>best type</td>
<td>type class [r/o]</td>
<td>The best descriptor type.</td>
</tr>
<tr>
<td>bounds</td>
<td>bounding rectangle</td>
<td>The boundary rectangle for the window.</td>
</tr>
<tr>
<td>class</td>
<td>type class [r/o]</td>
<td>The class.</td>
</tr>
<tr>
<td>default type</td>
<td>type class [r/o]</td>
<td>The default descriptor type.</td>
</tr>
<tr>
<td>document</td>
<td>document [r/o]</td>
<td>The document that owns this window.</td>
</tr>
<tr>
<td>index</td>
<td>integer</td>
<td>The number of the window.</td>
</tr>
</tbody>
</table>
### Property Class Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>international text [r/o]</td>
<td>The document’s name as shown in the window’s titlebar.</td>
</tr>
<tr>
<td>page number</td>
<td>integer</td>
<td>The number of the currently displayed page.</td>
</tr>
<tr>
<td>position</td>
<td>point [r/o]</td>
<td>The upper left coordinates of the window.</td>
</tr>
<tr>
<td>visible</td>
<td>Boolean [r/o]</td>
<td>Whether the window is visible.</td>
</tr>
<tr>
<td>zoomed</td>
<td>Boolean</td>
<td>Whether the window is zoomed.</td>
</tr>
<tr>
<td>zoom factor</td>
<td>small real</td>
<td>The current zoom factor specified as a percentage. For example, a value of 100 corresponds to a zoom factor of 1.0 (100%).</td>
</tr>
<tr>
<td>zoom type</td>
<td>constant</td>
<td>The zooming and content fitting algorithm currently employed. Possible values: no vary, fit page, fit width, fit height, and fit visible width.</td>
</tr>
</tbody>
</table>

### Related methods

- `go backward`
- `go forward`
- `goto`
- `goto next`
- `goto previous`
- `read page down`
- `read page up`
- `scroll`
- `select text`
- `zoom`

### PostScript Conversion

A file type converter that exports PDF files into PostScript format.

### Properties

Inherits other properties from `conversion`. 
### Related methods

**save**

### Text Annotation

A PDF text annotation (note) on a page in a PDF file. Can only be used as the target of a `make` event. All other access is via the `annotation` class.

**Note:** This object was formerly known as `TextAnnot`.

### Properties

Inherits from `annotation`.

### Related methods

- `find next note`
- `perform`
- `replace pages`

### Required suite events

The following events are sent by the Finder to all applications:

- `open`
• print
• quit
• run

Note: Most of these events have counterparts in the Core suite that have greater functionality. The Required suite is not listed in the AppleScript dictionary, even though it is implemented.

Acrobat DC Reader also supports the Required suite events, but no others.

open

Opens a file.

Syntax
open [reference]

Parameters

| open | The file or files to open. |

print

Prints one or more files.

Syntax
print [reference]

Parameters

| print | The file or files to print. |

quit

Terminates an application. For information on a variant event in the Core suite that accepts options, see quit on page 190.

Syntax
quit

run

Launches the application and invokes its standard startup procedures.

Syntax
run
Core suite events

Acrobat DC supports the following subset of the Core suite of Apple events:

- close
- count
- delete
- exists
- get
- make
- move
- open
- quit
- save
- set

**close**

Closes a document.

**Syntax**

`close [reference] saving [constant] linearize [boolean]`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>The document to close.</td>
</tr>
<tr>
<td>saving</td>
<td>Determines whether to save a document that has been modified before quitting. Possible values: yes — Save the document. no — Do not save the document. ask — Ask the user whether to save the document. The default value is ask.</td>
</tr>
<tr>
<td>linearize</td>
<td>Determines whether the document should be optimized for the web when saving before closing.</td>
</tr>
</tbody>
</table>

**Related events**

- open

**count**

Counts the number of instances of a particular class.
Syntax

```plaintext
count [type class] of [reference]
```

Parameters

- **count**: The class whose instances are to be counted.
- **each**: The class whose instances are to be counted. This keyword is optional.

**Note**: There is an alternate form using the keyword `each` in which the parameters are reversed:

```plaintext
count [reference] each [type class]
```

Returns

An integer specifying the number of elements.

**AppleScript example**

```plaintext
count annotation of document "dev_acro.pdf"
count menu item of menu "View"
count document 1 each bookmark
```

**delete**

Deletes one or more objects.

Syntax

```plaintext
delete [reference]
```

Parameters

- **delete**: The object to be deleted.

**Related events**

- `make`
- `exists`

**AppleScript example**

```plaintext
delete first bookmark of document "test.pdf"
```

**exists**

Tests whether a specified object exists.
Syntax

```markdown
[reference] exists
exists [reference]
```

Parameters

| exists | Object whose existence is checked. |

Returns

`true` if the object exists, `false` otherwise.

AppleScript example

```markdown
exists second document
second document exists
```

get

Retrieves the value of an object or property.

Syntax

```markdown
get [reference] as [class]
```

*Note:* The keyword `get` is optional.

Parameters

| get | The object or property whose value is returned. |
| as  | The form in which the data is returned. |

Returns

The value of the specified property or object. If the specified object does not exist, no result is returned.

Related events

`set`

AppleScript example

```markdown
get the name of last bookmark
get the index of last bookmark as string
```

make

Creates a new object.
Syntax

```plaintext
make new [type class] at [location reference] with data [anything] with properties [record]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>make [new]</td>
<td>The class of the new object.</td>
</tr>
<tr>
<td>at</td>
<td>The location at which to insert the new object.</td>
</tr>
<tr>
<td>with data</td>
<td>The initial data for the new object.</td>
</tr>
<tr>
<td>with properties</td>
<td>The initial values for the properties of the new object.</td>
</tr>
</tbody>
</table>

Returns

A reference to the newly created object.

Related events

- `delete`
- `exists`

AppleScript example

```plaintext
set myAnnotation to make TextAnnotation at beginning
set name of myAnnotation to "Werner Heisenberg"
set contents of myAnnotation to "Might have been here"
```

**move**

Moves a `page` object.

Syntax

```plaintext
move [reference] to [location reference]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
<td>The page object to move. The first page in a document is page 1.</td>
</tr>
<tr>
<td>to</td>
<td>The new location for the page.</td>
</tr>
</tbody>
</table>

Returns

A reference to the page that is moved.

AppleScript example

```plaintext
move page 3 to before page 1
```
open

Opens a document or documents.

Syntax
open [list of alias] invisible [boolean] options [string]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>The document or documents to open.</td>
</tr>
<tr>
<td>invisible</td>
<td>Whether the opened document should be hidden. Default is false.</td>
</tr>
<tr>
<td>options</td>
<td>Optional parameter string of open actions.</td>
</tr>
</tbody>
</table>

Related events

close

quit

Causes the Acrobat DC application to quit.

Syntax
quit saving [constant]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saving</td>
<td>Determines whether to save documents that have been modified before quitting. Possible values: yes — Save the document. no — Do not save the document. ask — If the documents have been modified, ask the user whether to save them. The default value is ask.</td>
</tr>
</tbody>
</table>

AppleScript example
quit saving yes

save

Saves a document.

Syntax
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>save</td>
<td>The document to be saved.</td>
</tr>
<tr>
<td>to</td>
<td>The file into which the document is to be saved. This parameter is optional in Acrobat DC 6.0 and higher. Specifying the <code>to</code> parameter is equivalent to doing a Save As. You can save a document in one of the supported formats with the <code>using</code> parameter.</td>
</tr>
<tr>
<td>linearize</td>
<td>Determines whether the document should be optimized for the web.</td>
</tr>
<tr>
<td>using</td>
<td>The conversion method used to save the document in the desired format. Supported conversions by name are EPS Conversion and PostScript Conversion. All others can be specified by index using the conversion object.</td>
</tr>
</tbody>
</table>

AppleScript example

```applescript
save document 1 to file "MyHardDrive:tempBig.ps" using PostScript Conversion with embedded fonts, images, preview, and annotation without binary given postScript level: 1
```

set

Sets an object's data or properties.

Syntax

```applescript
set [reference] to [anything]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>The object or property whose value is set.</td>
</tr>
<tr>
<td>to</td>
<td>The new value.</td>
</tr>
</tbody>
</table>

Related events

get

AppleScript example

```applescript
set the name of first bookmark to "Chapter 1"
```

Acrobat DC application events

This section describes a number of Acrobat DC API calls for the Apple event interface that are specific to Acrobat DC applications. The supported events in this suite are:

- `bring to front`
- `clear_selection`
• close all docs
• create thumbs
• delete pages
• delete thumbs
• execute
• find next note
• find text
• get info
• go backward
• go forward
• goto
• goto next
• goto previous
• insert pages
• is toolbutton enabled
• maximize
• perform
• print pages
• read page down
• read page up
• remove toolbutton
• replace pages
• scroll
• select text
• set info
• zoom

Apple encourages the use of an application's signature as the name of its class for application-specific Apple events. The string CARO is the name of the class for Acrobat DC-specific Apple events:

```
#define kAEAcrobatViewerClass 'CARO'
```

AppleScript does not need this information.

**bring to front**

Brings the specified document's window to the front.

**Syntax**

`bring to front [reference]`
clear selection

Clears the document’s current selection, if any.

Syntax

clear selection [reference]

Parameters

| clear selection | The document containing the selection to be cleared |

Related events

select text

AppleScript example

clear selection document "PLUGINS.PDF"

Apple event ID

kAEClearSelection ('clsl')

close all docs

Closes all documents.

Syntax

close all docs saving [constant]
create thumbs

Creates thumbnail images for all pages in the document.

Syntax

create thumbs [reference]

Parameters

create thumbs The document in which thumbnails are created.

Related events

delete thumbs

AppleScript example

create thumbs document "roadmap.pdf"

Apple event ID

kAECreateThumbs ('crtb')
delete pages

Deletes the specified pages in the document.

Syntax

```plaintext
delete pages [reference] first [integer] last [integer]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete pages</td>
<td>The document containing the pages to be deleted.</td>
</tr>
<tr>
<td>first</td>
<td>The first page to be deleted. The first page in a document is page 1.</td>
</tr>
<tr>
<td>last</td>
<td>The last page to be deleted.</td>
</tr>
</tbody>
</table>

Related events

- `insert pages`
- `replace pages`

AppleScript example

```plaintext
delete pages document "AppleEvt.pdf" first 1 last 3
```

Apple event ID

```plaintext
kAEDeletePages ('dlpg')
```

Apple event parameters

- keyAEFirstPage ('frpg')
- keyAELastPage ('lapg')

delete thumbs

Deletes all thumbnails from the document.

Syntax

```plaintext
delete thumbs [reference]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete thumbs</td>
<td>The document from which thumbnails are deleted.</td>
</tr>
</tbody>
</table>

Related events

- `create thumbs`
AppleScript example
  delete thumbs document "AppleEvt.pdf"

Apple event ID
  kAEDeleteThumbs ("dltb")

class execute
  Executes the specified menu item.

Syntax
  execute [reference]

Parameters
  execute  The menu item to execute. See the Acrobat and PDF Library API Reference for a list of menu item names.

AppleScript example
  activate
  execute menu item "Open"

Apple event ID
  kAEExecute ("exec")

find next note
  Finds and selects the next text note in a document.

Syntax
  find next note [reference] wrap around [boolean]

Parameters
  find next note  The document in which to find the next text note.
  wrap around  Determines whether to continue the search at the beginning of a document if a note has not been found after the end of the document is reached. If true, the search wraps around; otherwise it does not. The default value is false.

Returns
  The text annotation found.
find text

Finds text in a document.

Syntax


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find text</td>
<td>The document to be searched.</td>
</tr>
<tr>
<td>string</td>
<td>The string to be found.</td>
</tr>
<tr>
<td>case sensitive</td>
<td>Determines whether searching is case-sensitive. The default value is false.</td>
</tr>
<tr>
<td>whole words</td>
<td>Determines whether to search only for whole words. The default value is false.</td>
</tr>
<tr>
<td>wrap around</td>
<td>Determines whether to continue the search at the beginning of a document if the specified text has not been found after the end of the document is reached. If true, the search wraps around; otherwise it does not. The default value is false.</td>
</tr>
</tbody>
</table>

Related events

find next note

AppleScript example

find next note document "dev_acro.pdf"

Apple event ID

kAEFindNextNote ('fnnt')

Apple event parameters

keyAEWrapAround ('wrar')

The document to be searched.

The string to be found.

Determines whether searching is case-sensitive. The default value is false.

Determines whether to search only for whole words. The default value is false.

Determines whether to continue the search at the beginning of a document if the specified text has not been found after the end of the document is reached. If true, the search wraps around; otherwise it does not. The default value is false.

find text document "PLUGINS.PDF" string "Develop" whole words true

kAEFindText ('ftxt')
Apple event parameters

- keyAEJSONObjectSearchString ('sstr')
- keyAEJSONObjectCaseSensitive ('case')
- keyAEJSONObjectWholeWordsOnly ('whwd')
- keyAEJSONObjectWrapAround ('wrar')

get info

Gets the value of the specified key in the document’s Info dictionary.

Syntax

get info [reference] key [international text]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get info</td>
<td>The document from which to obtain the Info dictionary entry.</td>
</tr>
<tr>
<td>key</td>
<td>The case-sensitive Info dictionary key whose value is to be obtained. The predefined keys are: Creator, Producer, CreationDate, Author, Title, Subject, and Keywords. None of these is required in the PDF file.</td>
</tr>
</tbody>
</table>

Returns

A string containing the specified key’s value, or an empty string if the key is not found.

AppleScript example

get info document "PLUGINS.PDF" key "CreationDate"

Apple event ID

kAEGetInfo ('gnfo')

Apple event parameters

keyAEInfoKey ('inky')

go backward

Goes to the previous view in the stored view history. Does nothing if the current view is the first view in the history.

Syntax

go backward [reference]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go backward</td>
<td>A PDF Window object</td>
</tr>
</tbody>
</table>
Related events

  go_forward
  goto
  goto_next
  goto_previous

AppleScript example

  go backward first PDF Window

Apple event ID

  kAEGoBack ('gbck')

**go forward**

Goes to the next view in the stored view history. Does nothing if the current view is the last view in the history.

**Syntax**

  go forward [reference]

**Parameters**

| go forward | A PDF Window object |

Related events

  go_backward
  goto
  goto_next
  goto_previous

AppleScript example

  go forward first PDF Window

Apple event ID

  kAEGoForward ('gfwd')

**goto**

Displays the page that has the specified page number.
Syntax

goto [reference] page [integer]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goto</td>
<td>The PDF Window object in which to change the page.</td>
</tr>
<tr>
<td>page</td>
<td>The page number of the page to be displayed. The first page in a document is page 1.</td>
</tr>
</tbody>
</table>

Related events

- go backward
- go forward
- goto next
- goto previous

AppleScript example

goto first PDF Window page 2

Apple event ID

kAEGoToPage ('gtpg')

Apple event parameters

keyAEPageNumber ('pg #')

goto next

Displays the next page after the one currently displayed in the PDF Window. Does nothing if the current page is the last page in the document.

Syntax

goto next [reference]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goto next</td>
<td>The PDF Window object in which to change the page.</td>
</tr>
</tbody>
</table>

Related events

- go backward
- go forward
**goto**

**goto previous**

**AppleScript example**

```applescript
goto next first PDF Window
```

**Apple event ID**

`kAEGotoNextPage ('nxpg')`

---

**goto previous**

Displays the previous page before the one currently displayed in the [PDF Window](#). Does nothing if the current page is the first page in the document.

**Syntax**

```applescript
goto previous [reference]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goto previous</td>
<td>The <a href="#">PDF Window</a> object in which to change the page.</td>
</tr>
</tbody>
</table>

**Related events**

- `go backward`
- `go forward`
- `goto`
- `goto next`

**AppleScript example**

```applescript
goto previous first PDF Window
```

**Apple event ID**

`kAEGotoPrevPage ('pvpg')`

---

**insert pages**

Inserts one or more pages from one document into another.

**Syntax**

```applescript
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert pages</td>
<td>The target document in which to insert the page or pages.</td>
</tr>
<tr>
<td>after</td>
<td>The number of the page after which the pages will be inserted. The first page in a document is page 1.</td>
</tr>
<tr>
<td>from</td>
<td>The source document containing the page or pages to be inserted.</td>
</tr>
<tr>
<td>starting with</td>
<td>The first page to be inserted.</td>
</tr>
<tr>
<td>number of pages</td>
<td>The number of pages to be inserted.</td>
</tr>
<tr>
<td>insert bookmarks</td>
<td>Determines whether to copy bookmarks that point to the inserted pages.</td>
</tr>
<tr>
<td></td>
<td>Default is true.</td>
</tr>
</tbody>
</table>

Related events

- delete pages

AppleScript example

```
insert pages document "AppleEvt.pdf" after 2 from document "dev_acro.pdf"
starting with 1 number of pages 4
```

Apple event ID

```kAEInsertPages ('inpg')```

Apple event parameters

- `keyAEInsertAfter ('inaf')`
- `keyAESourceDoc ('srdc')`
- `kAESourceStartPage ('stpg')`
- `keyAENumPages ('nmpg')`
- `keyAEInsertBookmarks ('inbm')`

is toolbutton enabled

Determines whether the specified toolbar button is enabled.

Syntax

```
is toolbutton enabled named [international text]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>named</td>
<td>Button name. See the PDF Library documentation for a list of toolbar button names.</td>
</tr>
</tbody>
</table>
Returns

true if the toolbar button is enabled, false otherwise.

Related events

remove_toolbutton

AppleScript example

is toolbutton enabled named "AcroSrch:Query"

Apple event ID

kAEIsToolButtonEnabled ('tben')

Apple event parameters

keyAEButtonname ('tbnm')

maximize

Sets the document’s window size to either its maximum or original size.

Syntax

maximize [reference] max size [integer]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize</td>
<td>The document whose window is to be resized.</td>
</tr>
<tr>
<td>max size</td>
<td>If true, the document’s window is set to full size. If false, the window is returned to its original size.</td>
</tr>
</tbody>
</table>

AppleScript example

maximize document "AppleEvt.pdf" max size false

Apple event ID

kAEMaximize ('maxi')

Apple event parameters

keyAEMaxSize ('mksz')

perform

Executes a bookmark’s or link annotation’s action.
Syntax
perform [reference]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>The bookmark or page object whose action is to be performed.</td>
</tr>
</tbody>
</table>

AppleScript example
perform last bookmark

Apple event ID
kAEPerform ('prfm')

print pages
Prints one or more pages from a document without displaying a modal Print dialog box.

Syntax
binary output [boolean] shrink to fit [boolean]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print pages</td>
<td>The document containing the page or pages to be printed. This keyword and the actual filename must be specified.</td>
</tr>
<tr>
<td>first</td>
<td>The first page to be printed. The default value is 1.</td>
</tr>
<tr>
<td>last</td>
<td>The last page to print. The default value is the number of the last page in the document.</td>
</tr>
<tr>
<td>PS Level</td>
<td>The PostScript language level (1 or 2) to use when printing to a PostScript printer. The default value is 1.</td>
</tr>
<tr>
<td>binary output</td>
<td>Determines whether binary output is permitted (used for PostScript printing only). The default value is false.</td>
</tr>
<tr>
<td>shrink to fit</td>
<td>Determines whether pages should be shrunk to fit paper in printer. The default value is false.</td>
</tr>
</tbody>
</table>

AppleScript example
print pages document "AppleEvt.pdf" first 1 last 3 PS Level 2 binary output true shrink to fit true

Apple event ID
kAEPrintPages ('prpg')
Apple event parameters

- keyAEFirstPage ('frpg')
- keyAELastPage ('lapg')
- keyAEPSLevel ('pslv')
- keyAEBinaryOK ('binO')
- keyAEShrinkToFit ('s2ft')

read page down

Scrolls forward through the document by one screen.

Syntax

```
read page down [reference]
```

Parameters

```
read page down  The PDF Window object to be scrolled.
```

Related events

- read page up
- scroll

AppleScript example

```
read page down first PDF Window
```

Apple event ID

```
kAEReadPageDown ('pgdn')
```

read page up

Scrolls backward through the document by one screen.

Syntax

```
read page up [reference]
```

Parameters

```
read page up  The PDF Window object to be scrolled.
```

Related events

- read page down
- scroll
AppleScript example
read page up first PDFPageWindow

Apple event ID
kAEReadPageUp ('pgup')

remove toolbutton
Removes the specified button from the toolbar.

Syntax
remove toolbutton named [international text]

Parameters
named The name of the toolbar button to be removed. See the Acrobat and PDF Library API Reference for a list of toolbar button names.

Related events
is toolbutton enabled

AppleScript example
remove toolbutton named "ZoomIn"

Apple event ID
kAERemoveToolButton ('rmtb')

Apple event parameters
keyAEButtonname ('tbnm')

replace pages
Replaces one or more pages in a document with pages from another document.

Syntax

Parameters
replace pages The target document whose pages are to be replaced.
over The first page to be replaced. The first page in a document is page 1.
Scrolls the view of a page by the specified amount.

Syntax


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scroll</td>
<td>The <strong>PDF Window</strong> object in which to scroll the view.</td>
</tr>
<tr>
<td>X Amount</td>
<td>The amount to scroll in the horizontal direction, in pixels. Positive values move the view to the right.</td>
</tr>
<tr>
<td>Y Amount</td>
<td>The amount to scroll in the vertical direction, in pixels. Positive values move the view down.</td>
</tr>
</tbody>
</table>
Related events

read page down
read page up

AppleScript example

scroll first PDFWindow X Amount 20 Y Amount 100

Apple event ID

kAEScroll ('scrl')

Apple event parameters

keyAEXDelta ('xdlt')
keyAEXYDelta ('ydl')

select text

Selects text as specified by either character or word offsets.

Syntax

select text [reference] from words [list of integer] from chars [list of integer]

Parameters

<table>
<thead>
<tr>
<th>select text</th>
<th>The PDF Window object in which to select text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>from words</td>
<td>The words to be selected. This consists of one or more pairs of word offsets from the beginning of the document and word lengths (the number of contiguous words).</td>
</tr>
<tr>
<td>from chars</td>
<td>Characters to be selected. This consists of one or more pairs of character offsets from the beginning of the document and character lengths (the number of contiguous characters).</td>
</tr>
</tbody>
</table>

Related events

clear selection

AppleScript example

repeat with i from 1 to 10
    repeat with j from 1 to (10 - i)
        select text from words {i, j} end repeat
end repeat
Apple event ID

kAESetTextSelection ('stxs')

Apple event parameters

keyAEWordList ('fmwd')
keyAECharList ('fmch')

set info

Sets the value of a specified key in the document’s Info dictionary

Syntax

set info [reference] key [international text] value [international text]

Parameters

| set info  | The PDF Window in which to set the value of an Info dictionary entry. |
| key       | The Info dictionary key whose value is to be set.                  |
| value     | The value to be stored.                                          |

AppleScript example

set info document "PlugIns.pdf" key "Author" value "Wolfgang Pauli"

Apple event ID

kAESetInfo ('snfo')

Apple event parameters

keyAEInfoKey ('inky')
keyAEInfoValue ('invl')

zoom

Changes the zoom level of the specified PDF Window.

Syntax

zoom [reference] to [small real]
Parameters

<table>
<thead>
<tr>
<th>zoom</th>
<th>The PDF Window object to be zoomed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>The zoom factor specified as a percentage. For example, a value of 100 (100%) displays the document with a magnification of 1.0.</td>
</tr>
</tbody>
</table>

AppleScript example

```javascript
zoom first PDFWindow to 150
```

Apple event ID

```javascript
kAEZoomTo ('zmto')
```

Apple event parameters

```javascript
keyAEZoomFactor ('zmft')
```

Miscellaneous events

Acrobat DC provides an Apple event that does not fall into one of the regular suites: `do script`

**do script**

Executes the specified JavaScript script.

**Syntax**

```javascript
do script [international text] file [alias]
```

**Parameters**

<table>
<thead>
<tr>
<th>do script</th>
<th>The JavaScript script to be executed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File holding the JavaScript script to be executed.</td>
</tr>
</tbody>
</table>

**Returns**

Result of JavaScript execution as text.

**AppleScript example**

```javascript
do script MyJavaScriptFile.js
```
This chapter describes IAC support for the Acrobat DC Catalog plug-in, which allows you to create a full-text index of a set of PDF documents. A full-text index is a searchable database of all the text in the documents. After building an index, you can use the Acrobat DC Search command to search the entire library quickly. Searches of full-text indexes created using Catalog are faster and more convenient than using the Find command.

For more information on Catalog, see the Acrobat DC Help and the Acrobat and PDF Library API Reference.

Catalog Windows messages

Catalog broadcasts a set of Windows messages when certain operations occur. These messages are broadcast whether the operations are initiated from the user interface, HFT methods, or DDE methods.

- AcrobatCatalogBuildSuccess — On every successful build.
- AcrobatCatalogBuildFail — On every failed build.
- AcrobatCatalogBuildStopped — When a build has stopped.

Catalog DDE methods

Clients can connect to the Catalog plug-in through DDE using the service name Acrobat and the topic name Control. This section lists the available DDE methods.

AppExit

Exits Acrobat DC Catalog.

Syntax

```plaintext
[AppExit()]
```

Returns

If true, Catalog exited successfully, otherwise false.

AppFront

Brings Catalog to the front.

Syntax

```plaintext
[AppExit()]
```
**FileBuild**

Builds an index based on the specified index definition file.

**Syntax**

```c
[FileBuild(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be opened, including the .pdx extension.</td>
</tr>
</tbody>
</table>

**Returns**

If `true`, the file opened successfully, otherwise `false`.

**FileOpen**

Opens an index definition file and displays the Edit Index Definition dialog box.

**Syntax**

```c
[FileOpen(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be opened, including the .pdx extension.</td>
</tr>
</tbody>
</table>

**Returns**

`true` if the file opened successfully, otherwise `false`.

**FilePurge**

Purges an index definition file.

**Syntax**

```c
[FilePurge(char* fullPath)]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullPath</td>
<td>The full path of the file to be purged, including the .pdx extension.</td>
</tr>
</tbody>
</table>

**Returns**

`true` if the file was successfully purged, otherwise `false`. 
Acrobat DC Forms Plug-In

The Acrobat DC Forms plug-in allows a PDF document to act as a form; that is, the Acrobat DC equivalent of a paper form with fields. This chapter describes the OLE automation methods exported by the Acrobat DC AcroForm plug-in.

The Forms plug-in for Acrobat DC (versions 4.0 and above) allows users to author form fields. For Acrobat DC Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat DC forms. The Acrobat DC Reader Forms plug-in also does not allow users to save data to the local hard disk. Both Acrobat DC and Acrobat DC Reader allow Web designers to send data from the form back to a Web server.

**Note:** Forms as used here do not refer to XObject forms as defined in the PDF Reference.

For more information on forms, see the Acrobat DC Help and the PDF Library documentation.

Forms plug-in OLE automation

The Acrobat DC Forms plug-in works as an automation server in the Windows environment. Because the automation capabilities have been added to a plug-in, rather than an executable that can be directly launched, the following steps are necessary to access them from an automation controller:

1. Instantiate the Acrobat DC application by using the Visual Basic `CreateObject` method. For example:
   ```vbnet
   CreateObject("AcroExch.App")
   ```
   This causes the Acrobat DC Forms plug-in to run, at which time it registers its class object with OLE.

2. Instantiate the main exposed object:
   ```vbnet
   CreateObject("AFormAut.App")
   ```

Registration in the Windows registry (which is different from the class object registration described above) happens every time Acrobat DC loads the plug-in. Therefore, you must run Acrobat DC at least once with the AForm32.api file in the Plugins folder before its type library can be found for object browsing within the Microsoft Visual Studio environment. This is also necessary in order to allow early binding. Declare the program variables as objects of the corresponding classes in AFORMAUTLib, and not simply as `Object`.

**Note:** Neither Acrobat DC nor the Acrobat DC Forms plug-in are thread-safe, and therefore Acrobat DC Forms OLE automation uses the single-threading model.

Exceptions

All methods and properties may return an exception. These may include standard OLE exceptions, such as:

- `E_OUTOFMEMORY` (0x8007000E)
- `E_INVALIDARG` (0x80070057)

These exceptions are not specifically listed in the descriptions of the methods and properties that appear in this chapter. Others are Acrobat DC Forms-specific, and are listed in the following table.
The actual numeric value of the returned exception is assembled as an HRESULT, uses the FACILITY_ITF, and starts with decimal 512 (hex 0x0200), as recommended by Microsoft. For example, the numeric value of the exception AutErcNoForm is 0x80040201. The important part is the right-most (0x201), which is the first error in the enumeration below.

<table>
<thead>
<tr>
<th>Exception name</th>
<th>Numeric value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutErcNoDoc</td>
<td>1</td>
<td>No document is currently open in the Acrobat DC application.</td>
</tr>
<tr>
<td>AutErcNotTerminal</td>
<td>2</td>
<td>This property or method applies to terminal fields or their annotations.</td>
</tr>
<tr>
<td>AutErcNotToThisFieldType</td>
<td>3</td>
<td>This property or method is not applicable to this type of field.</td>
</tr>
</tbody>
</table>

**AFormApp**

AFormApp is the only object the controller can externally instantiate (that is, using CreateObject). All other objects must be created by navigating down the hierarchy with the methods and properties described in this section.

**Field**

A field in the document that is currently active in Acrobat DC.

**Methods**

The Field object has the following methods.

- **PopulateListOrComboBox**
- **SetBackgroundColor**
- **SetBorderColor**
- **SetButtonCaption**
- **SetButtonIcon**
- **SetExportValues**
- **SetForegroundColor**
- **SetJavaScriptAction**
- **SetResetFormAction**
- **SetSubmitFormAction**

**PopulateListOrComboBox**

Specifies the item names and optionally exports values for a field of type listbox or combobox.
**Syntax**

```c
void PopulateListOrComboBox ( const VARIANT& arrItems,
                        const VARIANT& arrExportVal);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| arrItems    | An array of strings, with each element representing an item name.  
|             | There is a limit of 64K for string data in a combo or list box control on Windows platforms. For Mac OS systems, the limit is 200 entries for the combo or list box control. Using more than these limits degrades performance and makes the control unusable. |
| arrExportVal| Optional. An array of strings, the same size as the first parameter, with each element representing an export value.  
|             | Some of the elements in `exportString` may be empty strings.  |

**Exceptions**

Raises **AutErcNotToThisFieldType** if the field is not of type listbox or combobox.

**Related methods**

Add

**SetBackgroundColor**

Specifies the background color for a field. The background color is used to fill the field's rectangle.

**Syntax**

```c
void SetBackgroundColor (LPCTSTR bstrColorSpace, float GorRorC, float GorM, float BorY, float K);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| bstrColorSpace   | Values are defined by using a transparent, gray, RGB or CMYK color space. Valid strings include:  
|                  | • T  
|                  | • G  
|                  | • RGB  
|                  | • CMYK  |
| GorRorC          | Used if `bstrColorSpace` is set to `T`, `G`, or `RGB`. A float range between zero and one inclusive.  |
| GorM             | Used if `bstrColorSpace` is set to `G`. A float range between zero and one inclusive.  |
SetBorderColor

Specifies the border color for a field. The border color is used to stroke the field’s rectangle with a line as large as the border width. The new border color is propagated to any child annotations underneath, so the field may be non-terminal.

Syntax

void SetBorderColor (LPCTSTR bstrColorSpace, float GorRorC, float GorM, float BorY, float K);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| bstrColorSpace | Values are defined by using a transparent, gray, RGB or CMYK color space. Valid strings include:  
| GorRorC | Used if bstrColorSpace is set to T, G, or RGB. A float range between zero and one inclusive. |
| GorM | Used if bstrColorSpace is set to G. A float range between zero and one inclusive. |
| BorY | Used if bstrColorSpace is set to RGB. A float range between zero and one inclusive. |
| K | Used if bstrColorSpace is set to CMYK. A float range between zero and one inclusive. |

Related methods

SetBackgroundColor
SetForegroundColor

Example

Field.SetBackgroundColor "RGB", 0.7, 0.3, 0.6, 0

SetBorderColor
Example
Field.SetBorderColor "RGB", 0.7, 0.3, 0.6, 0

SetButtonCaption
The caption to be used for the appearance of a field of type button.

Syntax
void SetButtonCaption (LPCTSTR bstrFace, LPCTSTR bstrCaption);

Parameters

<table>
<thead>
<tr>
<th>bstrFace</th>
<th>A string that specifies the face for which the caption will be used. Valid strings include:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N — Normal appearance</td>
</tr>
<tr>
<td></td>
<td>D — Down appearance</td>
</tr>
<tr>
<td></td>
<td>R — Appearance for rollover</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bstrCaption</th>
<th>The caption for the button.</th>
</tr>
</thead>
</table>

If a button’s layout is of type icon only, the caption is not used in generating its appearance. In addition, only the Normal face is displayed, unless the Highlight is of type push.

Exceptions

Raises AutErcNotToThisFieldType if the field is not of type button. The new appearance is propagated to any child annotations underneath; the field may be non-terminal.

Related methods

SetButtonIcon

Example
Field.SetButtonCaption "D", "Submit Form"

SetButtonIcon
Specifies the icon to be used for the appearance of a field of type button.

Syntax
void SetButtonIcon (LPCTSTR bstrFace, LPCTSTR bstrFullPath, short pageNum);
Parameters

**bstrFace**  
A string that specifies the face for which the icon will be used. Valid strings include:
- N — Normal appearance
- D — Down appearance
- R — Appearance for rollover

**bstrFullPath**  
The full path of the PDF file to be used as the source of the appearance.

**pageNum**  
Used to select the page inside that PDF file (zero-based).

If a button's layout is of type icon only, the caption is not used in generating its appearance. In addition, only the Normal face is displayed, unless the Highlight is of type push.

Exceptions

Raises [AutErcNotToThisFieldType](#) if the field is not of type button. The new appearance is propagated to any child annotations underneath, so it is OK if the field is non-terminal.

Related methods

[SetButtonCaption](#)

Example

Field.SetButtonIcon "N", "c:\Clipart.pdf", 0

**SetExportValues**

Sets the export values for each of the annotations of a field of type radio button and checkbox.

For radio button fields, this is necessary to make the field work properly as a group. One button is checked at any given time, giving its value to the field as a whole.

For checkbox fields, unless an export value is specified, the default is used when the field checked is Yes. When it is unchecked, its value is Off (this is also true for a radio button field when none of its buttons are checked).

**Syntax**

```c++
void SetExportValues (const VARIANT& arrExportVal);
```

**Parameters**

**arrExportVal**  
An array of strings, which is expected to have as many elements as there are annotations in the field. The elements of the array are distributed among the individual annotations comprising the field, using their tab order.

Exceptions

Raises [AutErcNotToThisFieldType](#) if the field is not of type radio button or checkbox.
### Related methods

- Add

### Example

```vba
Dim arrExp(1) As String
arrExp(0) = "CreditCardA"
arrExp(1) = "CreditCardB"
Field.SetExportValues arrExp
```

### SetForegroundColor

Specifies the foreground color for a field. It represents the text color for text, button, combobox, or listbox fields and the check color for checkbox or radio button fields.

The parameters are similar to SetBorderColor and SetBackgroundColor, except that the transparent color space is not allowed.

**Syntax**

```vba
void SetForegroundColor (LPCTSTR bstrColorSpace, float GorRorC, float GorM, float BorY, float K);
```

**Parameters**

<table>
<thead>
<tr>
<th>bstrColorSpace</th>
<th>Values are defined by using a transparent, gray, RGB or CMYK color space. Valid strings include:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• T</td>
</tr>
<tr>
<td></td>
<td>• G</td>
</tr>
<tr>
<td></td>
<td>• RGB</td>
</tr>
<tr>
<td></td>
<td>• CMYK</td>
</tr>
<tr>
<td>GorRorC</td>
<td>Used if bstrColorSpace is set to T, G, or RGB. A float range between zero and one inclusive.</td>
</tr>
<tr>
<td>GorM</td>
<td>Used if bstrColorSpace is set to G. A float range between zero and one inclusive.</td>
</tr>
<tr>
<td>BorY</td>
<td>Used if bstrColorSpace is set to RGB. A float range between zero and one inclusive.</td>
</tr>
<tr>
<td>K</td>
<td>Used if bstrColorSpace is set to CMYK. A float range between zero and one inclusive.</td>
</tr>
</tbody>
</table>

### Related methods

- SetBackgroundColor
- SetBorderColor

### Example

```vba
Field.SetForegroundColor "CMYK", 0.25, 0.25, 0.25, 0.1
```
SetJavaScriptAction

Sets the action of the field to be of type JavaScript. When using SetJavaScriptAction within Visual Basic, you can use Chr(13) to add a <CR>, and Chr(9) for tabs, so that the function is well formatted.

Syntax

```c
void SetJavaScriptAction (LPCTSTR bstrTrigger, LPCTSTR bstrTheScript);
```

Parameters

- **bstrTrigger**: A string that specifies the trigger for the action. Valid strings include:
  - up
  - down
  - enter
  - exit
  - calculate
  - validate
  - format
  - keystroke

- **bstrTheScript**: The script itself.

If the trigger is `calculate`, an entry is added at the end of the calculation order array (see the `CalcOrderIndex` property).

Calculation script

A simple calculate script is supplied with Acrobat DC.

```c
AFSimple_Calculate(cFunction, cFields)
```

- `cFunction` is one of AVG, SUM, PRD, MIN, MAX
- `cFields` is the list of the fields to use in the calculation.

Formatting scripts

The following scripts and formats can be used for the `format` and `keystroke` triggers:

```c
AFDate_KeystrokeEx(cFormat)  A string that specifies the format.
AFDate_Format(cFormat)

AFTime_Keystroke(ptf)  A time format:
AFTime_Format(ptf)
```

- `cFormat` is one of:
  - "m/d", "m/d/yy", "mm/dd/yy", "mm/yy", "d-mmm",
    "d-mmm-yy", "dd-mmm-yy", "yy-mm-dd",
    "mmm-yy", "mmmm-yy", "mmm d, yyyy", "mmmm
d, yyyy", "m/d/yy h:MM tt", "m/d/yy HH:MM"

- `ptf` is the time format:
  - 0 = 24HR_MM [ 14:30 ]
  - 1 = 12HR_MM [ 2:30 PM ]
  - 2 = 24HR_MM_SS [ 14:30:15 ]
  - 3 = 12HR_MM_SS [ 2:30:15 PM ]`
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetResetFormAction</td>
<td>Sets the action of the field to be of type ResetForm.</td>
</tr>
</tbody>
</table>

**Syntax**

```c
void SetResetFormAction (LPCTSTR bstrTrigger, long theFlags, const VARIANT& arrFields);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bstrTrigger</td>
<td>A string that specifies which trigger is used for the action. Valid strings include: up — Mouse up, down — Mouse down, enter — Mouse enter, exit — Mouse exit</td>
</tr>
<tr>
<td>theFlags</td>
<td>When 0 (Include), <code>arrFields</code> specifies which fields to include in the reset operation. When non-zero (Exclude), <code>arrFields</code> specifies which fields to exclude from the reset operation.</td>
</tr>
</tbody>
</table>
Sets the action of the field to be of type `SubmitForm`.

**Syntax**

```c
void SetSubmitFormAction (LPCTSTR bstrTrigger, LPCTSTR bstrTheURL, long theFlags, const VARIANT& arrFields);
```

**Parameters**

- **bstrTrigger**
  - A string that specifies which trigger is used for the action. Valid strings include:
    - `up` — Mouse up
    - `down` — Mouse down
    - `enter` — Mouse enter
    - `exit` — Mouse exit

- **bstrTheURL**
  - A string containing the URL.

- **theFlags**
  - A collection of flags that define various characteristics of the action.
  - See the `PDF Reference` to learn how the binary value of this `long` is interpreted.

- **arrFields**
  - Optional. If specified, represents an array of strings for the fully-qualified names of the fields to submit when the action is executed. If the array is interpreted as fields to submit (as opposed to fields excluded from the submission, depending on the least-significant bit in the flags), then it may include the names of non-terminal fields, which is a way to cause all their children to be included in the submission.
  - If not specified, the created action does not include a `/Fields` key.

### Properties

The **Field** object has the following properties.

- **Alignment**
- **BorderStyle**
- **BorderWidth**
- **ButtonLayout**
- **CalcOrderIndex**
- **CharLimit**
Alignment

The text alignment of a text field. Valid alignments are:

- left
- center
- right

Syntax

[get/set] String

Returns

If the field is terminal and has multiple child annotations, a get returns the alignment for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

If the field is not of type text, an exception AutErcNotToThisFieldType is returned.

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned.

Example

Field.Alignment = left
BorderStyle

The border style for a field. Valid border styles include solid, dashed, beveled, inset, and underline.

Syntax

[get/set] String

Returns

If it is terminal and has multiple child annotations, a get returns the value of the border style for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

On a get, raises AutErcNotTerminal if the field is non-terminal, an exception is returned.

Example

Field.BorderStyle = "beveled"

BorderWidth

The thickness of the border when stroking the perimeter of a field’s rectangle. If the border color is transparent, this property has no effect except in the case of a beveled border. The value 0 represents no border, and the value 3 represents a thick border.

Syntax

[get/set] short

Returns

If it is terminal and has multiple child annotations, a get returns the value of the border width for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned.

Example

Field.BorderWidth = 1
ButtonLayout

The layout appearance of a button. Valid values include:

0 — Text only; the button has a caption but no icon.
1 — Icon only; the button has an icon but no caption.
2 — Icon over text; the icon should appear on top of the caption.
3 — Text over icon; the text should appear on top of the icon.
4 — Icon then text; the icon should appear to the left of the caption.
5 — Text then icon; the icon should appear to the right of the caption.
6 — Text over icon; the text should be overlaid on top of the icon.

If it is terminal and has multiple child annotations, a get returns the layout for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, therefore the field can be non-terminal.

**Syntax**

[get/set] short

**Exceptions**

If the field is not of type button, an exception AutErcNotToThisFieldType is returned.

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned.

**Example**

Field.ButtonLayout = 2

CalcOrderIndex

The zero-based calculation order of fields in the document. If you want the calculation for a field f2 to be performed after that for field f1, you need only set the CalcOrderIndex for f2 to f1’s CalcOrderIndex + 1. The elements in the calculation order array are shifted to make room for the insertion, but the first calculation is still at index 0.

**Syntax**

[get/set] short

**Example**

Set F1 = Fields("SubTotal")
Set F2 = Fields("Total")
F2.CalcOrderIndex = F1.CalcOrderIndex + 1

CharLimit

The limit on the number of characters that a user can type into a text field.

On a set, the property is propagated to any child annotations underneath, if any.
Syntax
[get/set] short

Exceptions
If the field is not of type text, an exception AutoErcNotToThisFieldType is returned.

DefaultValue
The default value of the field. It returns the empty string if the field has no default value. If the field is non-terminal, an exception AutoErcNotTerminal is returned.

Syntax
[get/set] String

See also
Value

Editable
Determines whether the user can type in a selection or must choose one of the provided selections. Comboboxes can be editable; that is, the user can type in a selection.

On a set, the property is propagated to any child annotations underneath, if any.

Syntax
[get/set] Boolean

Exceptions
Returns an exception of AutoErcNotToThisFieldType if the field is not of type combobox.

Example
Field.Editable = False

Highlight
Defines how a button reacts when a user clicks it. The four highlight modes supported are:
- none
- invert
- push
- outline

If it is terminal and has multiple child annotations, a get returns the highlight for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.
Syntax
[get/set] String

Exceptions
If the field is not of type button, an exception AutErcNotToThisFieldType is returned.

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned.

Example
Field.Highlight = "invert"

IsHidden
Determines whether the field is hidden or visible to the user. If the value is true the field is invisible, and false indicates that the field is visible.

During get operations, if the field is non-terminal, an exception AutErcNotTerminal is returned. If it is terminal, and has multiple child annotations, a get returns the value of the hidden flag for the first child, whichever annotation that happens to be.

During set operations, the property is propagated to any child annotations underneath, therefore a field can be non-terminal.

Syntax
[get/set] Boolean

Example
'Hide "name.last"
Set Field = Fields("name.last")
Field.IsHidden = True

IsMultiline
Determines whether the text field is multi-line or single-line. On a set, the property is propagated to any child annotations underneath, if any.

Syntax
[get/set] Boolean

Exceptions
If the field is not of type text, an exception AutErcNotToThisFieldType is returned.

Example
Field.IsMultiline = True
IsPassword

Determines whether the field will display asterisks for the data entered. Upon submission, the actual data entered is sent. Fields that have the password attribute set will not have the data in the field saved when the document is saved to disk.

On a set, the property is propagated to any child annotations underneath, if any.

Syntax

[get/set] Boolean

Exceptions

If the field is not of type text, an exception AutErcNotToThisFieldType is returned.

Example

Field.IsPassword = True

IsReadOnly

The read-only characteristic of a field. When a field is read-only, the user can see the field but cannot change it. If a button is read-only, the user cannot click it to execute an action.

Because this is a field flag and not an annotation flag, both a get and a set of this property are allowed regardless of whether the field is terminal or non-terminal.

- A get on a non-terminal field retrieves that field’s flag.
- A set changes the flag on all its terminal children.

Syntax

[get/set] Boolean

IsRequired

The required characteristic of a field. When a field is required, its value must be non-NULL when the user clicks a submit button that causes the value of the field to be sent to the web. If the field value is NULL, the user receives a warning message and the submit does not occur.

Since this is a field flag and not an annotation flag, both a get and a set of this property are allowed, regardless of whether the field is terminal or non-terminal.

A get on a non-terminal field retrieves that field’s flag. A set changes the flag on all its terminal children.

Syntax

[get/set] Boolean

IsTerminal

true if the field is terminal, otherwise false.
Syntax
[read-only] Boolean

Example
Dim Field As AFORMAUTLib.Field
Dim bTerminal As Boolean

'bTerminal should be True
bTerminal = Field.IsTerminal

Name
The fully qualified name of the field. It is the default member of the Field interface.

Syntax
[read-only] String

NoViewFlag
Determines whether a given field prints but does not display on the screen.

Set the NoViewFlag property to true to allow the field to appear when the user prints the document but not when it displays on the screen; set it to false to allow both printing and displaying.

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned. If it is terminal, and has multiple child annotations, a get returns the value of the no-view flag for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Syntax
[get/set] Boolean

PrintFlag
Determines whether a field prints. Set the PrintFlag property to true to allow the field to appear when the user prints the document, set it to false to prevent printing.

On a get, if the field is non-terminal, an exception AutErcNotTerminal is returned. If it is terminal, and has multiple child annotations, a get returns the value of the print flag for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Syntax
[get/set] Boolean
Style

The style of a checkbox or a radio button (the glyph used to indicate that the check box or radio button has been selected).

Valid styles include:

- check
- cross
- diamond
- circle
- star
- square

If it is terminal and has multiple child annotations, a get returns the style for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, therefore a field can be non-terminal.

Syntax

[get/set] String

Exceptions

During set, if the field is not of type checkbox or radio button, an exception `AutErcNotToThisFieldType` is returned.

On a get, if the field is non-terminal, an exception `AutErcNotTerminal` is returned.

Example

`Field.Style = "star"`

TextFont

The text font used when laying out the field. Valid fonts include:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Symbol
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- ZapfDingbats

On a set, the property is propagated to any child annotations underneath, if any.
Syntax
[get/set] String

Example
Field.TextFont = "Times-BoldItalic"

TextSize

The text points size used in the field. In combobox and radio button fields, the text size determines the size of the check. Valid text sizes include zero and the range from 4 to 144 inclusive.

A text size of zero means that the largest point size that can still fit in the field’s rectangle should be used. In multi-line text fields and buttons this is always 12 points.

On a set, the property is propagated to any child annotations underneath, if any.

Syntax
[get/set] short

Example
Field.TextSize = 18

Type

The type of the field as a string. Valid types that are returned:

- text
- button
- combobox
- listbox
- checkbox
- radiobutton
- signature

Syntax
[read-only] String

Example
Set Field = Fields("name.last")
' Should print "name.last"
print Field
' Should print the type of field. Example, "text"
print Field.Type

Value

A string that represents the value of the field. Returns the empty string if the field has no value. If the field is non-terminal, an exception `AutErcNotTerminal` is returned.

For fields of type checkbox, the value `Off` represents the unchecked state. The checked state is represented using the export value. This is also true for radio buttons (where each individual button in a
A collection of all the fields in the document that are currently active in Acrobat DC at the time Fields is instantiated.

The Fields collection includes both terminal and non-terminal fields. A terminal field is one that either does not have children, or if it does, they are simply multiple appearances (that is, child annotations) of the field in question.

Note: If you instantiate a Fields object, and subsequently fields are manually added or removed using the Forms tool in Acrobat DC, the Fields object will no longer be in sync with the document. You must re-instantiate the Fields object.

Methods

The Fields object has the following methods.

- Add
- AddDocJavascript
- ExecuteThisJavascript
- ExportAsPDF
- ExportAsHtml
- ImportAnFDF
- Remove

Add

Dynamically adds a new field to the Acrobat DC form and to the Fields collection.

Returns the newly-created Field object. You can pass the name of an existing field as a parameter, as long as that field is of the same type as the one being created.

This is useful in the following circumstances:
For radio buttons to use the `SetExportValues` method to make the radio buttons mutually exclusive.

For fields that should have multiple appearances (that is, child annotations) in the document.

**Syntax**

```
LPDISPATCH Add (LPCTSTR bstrFieldName, LPCTSTR bstrFieldType, short pageNum,
float left, float top, float right, float bottom);
```

**Parameters**

<table>
<thead>
<tr>
<th>bstrFieldName</th>
<th>The fully-qualified name of the field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bstrFieldType</td>
<td>Field type for the newly created field. Valid types are:</td>
</tr>
<tr>
<td></td>
<td>• text</td>
</tr>
<tr>
<td></td>
<td>• button</td>
</tr>
<tr>
<td></td>
<td>• combobox</td>
</tr>
<tr>
<td></td>
<td>• listbox</td>
</tr>
<tr>
<td></td>
<td>• checkbox</td>
</tr>
<tr>
<td></td>
<td>• radio button</td>
</tr>
<tr>
<td></td>
<td>• signature</td>
</tr>
</tbody>
</table>

You must use the quotation marks. See the sample code below.

When creating list or combo boxes, there is a limit of 64K for string data on Windows platforms. Mac OS systems have a limit of 200 entries for the list or combo boxes. Using more than the limit degrades performance. You populate the fields of the list and combo boxes using the `PopulateListOrComboBox` method.

<table>
<thead>
<tr>
<th>pageNum</th>
<th>The page number (zero-based).</th>
</tr>
</thead>
<tbody>
<tr>
<td>left, top, right, bottom</td>
<td>These parameters are floats representing the left, top, right, and bottom coordinates of the field rectangle, measured in rotated page space; that is, [0,0] is always at the left bottom corner regardless of page rotation.</td>
</tr>
</tbody>
</table>

**Returns**

The newly-created `Field` object.

**Related methods**

- `PopulateListOrComboBox`
- `Remove`

**Example**

```
Set Field = Fields.Add("payment", _ "radiobutton", 0, 100, 600, 130, 570)
```
AddDocJavascript

Adds a document-level JavaScript function to the PDF file. When using AddDocJavascript, within Visual Basic, you can use Chr(13) to add a <CR>, and Chr(9) for tabs, so that the function is well formatted.

**Syntax**
void AddDocJavascript (LPCTSTR bstrScriptName, LPCTSTR bstrTheScript);

**Parameters**
- **bstrScriptName**: The name of the function to be added to the document.
- **bstrTheScript**: The definition to be added to the document.

**Related methods**
ExecuteThisJavascript

**Example**
'Adding a document-level JavaScript function, to compute factorials:
Fields.AddDocJavaScript "Fact", _
"function Fact(n)" & Chr(13) & _
"{" & Chr(13) & _
Chr(9) & "if (n <= 0)" & Chr(13) & _
Chr(9) & Chr(9) & "return 1;" & Chr(13) & _
Chr(9) & "else" & Chr(13) & _
Chr(9) & Chr(9) & "return n * Fact(n - 1);" & Chr(13) & _
""

ExecuteThisJavascript

Executes the specified JavaScript script.

**Syntax**
CString ExecuteThisJavascript (LPCTSTR bstrTheScript);

**Parameters**
- **bstrTheScript**: A string containing a JavaScript script, which is executed by Acrobat DC in the context of the currently active document.

**Returns**
Returns a result by assigning it to event value.

**Related methods**
AddDocJavascript
Example

`Fields.ExecuteThisJavaScript "var f = _ this.getField("myButton"); f.delay = _ false;"

To get the returns in Visual Basic:

```vbnet
Dim cSubmitName As String
cSubmitName = Fields.ExecuteThisJavaScript
    "event.value = this.getField("myField").submitName;"
```

ExportAsFDF

Exports the data as FDF from an Acrobat DC form.

**Syntax**

```cpp
void ExportAsFDF (LPCTSTR bstrFullPath, LPCTSTR bstrSubmitButton,
    BOOL bEmptyFields, const VARIANT& arrFields);
```

**Parameters**

- **bstrFullPath**
  A full path of the file to which the produced FDF file will be saved.

- **bstrSubmitButton**
  The name of an existing form field of type `button` (in case you want to include it in the FDF file, as if it had been used to trigger a `SubmitForm` action). You can specify an empty string.

- **bEmptyFields**
  A Boolean value to indicate whether fields with no value should be included in the produced FDF file.

- **arrFields**
  Optional. An array of strings representing the fully-qualified names of the fields to include in the FDF file. This array may include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file.

**Related methods**

- **ImportAnFDF**
- **ExportAsHtml**

Example

```vbnet
Dim arrFields(1) As String
arrFields(0) = "name"
arrFields(1) = "address"
' This will create an FDF that includes
' name.last, name.first, address.street, etc., but only if they have a value
' (since we are passing False for the
' "bEmptyFields" parameter.
Fields.ExportAsFDF "C:\Temp\out.fdf", ",", False, arrFields
```
ExportAsHtml

Exports the data as HTML from an Acrobat DC form. This method is similar to `ExportAsFDF`. The only difference is that the form data is exported in URL-encoded format.

**Syntax**

```c
void ExportAsHtml (LPCTSTR bstrFullPath, LPCTSTR bstrSubmitButton, BOOL bEmptyFields, const VARIANT& arrFields);
```

**Parameters**

- `bstrFullPath` A full path of the file to which the produced FDF file will be saved.
- `bstrSubmitButton` The name of an existing form field of type `button` (in case you want to include it in the FDF file, as if it had been used to trigger a `SubmitForm` action). You may pass an empty string.
- `bEmptyFields` A Boolean to indicate whether fields with no value should be included in the produced FDF file.
- `arrFields` Optional. An array of strings representing the fully-qualified names of the fields to include in the FDF file. This array may include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file.

**Related methods**

- `ExportAsFDF`

ImportAnFDF

Imports the FDF file into an Acrobat DC form.

**Syntax**

```c
void ImportAnFDF (LPCTSTR bstrFullPath);
```

**Parameters**

- `bstrFullPath` The full path of the file containing the FDF file to be imported.

**Related methods**

- `ExportAsFDF`

Remove

Removes a field from the Acrobat DC Form and from the `Fields` collection.

**Syntax**

```c
void Remove (LPCTSTR bstrFieldName);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bstrFieldName</td>
<td>The fully-qualified name of the field to be removed from the Acrobat DC form. If the field has multiple child annotations, all of them are removed. If multiple fields have the same name, all are removed.</td>
</tr>
</tbody>
</table>

Related methods

Add

Example

'Remove fields you no longer used.
Fields.Remove("MyOldField")

Properties

The Fields object has the following properties.

- **Count**
- **Item**
- **_NewEnum**

Count

The number of items in the collection.

Syntax

[read-only] long

Example

Dim Field As AFORMAUTLib.Field
Dim nFields As Long

nFields = Fields.Count

For Each Field In Fields
    If Field.IsTerminal Then
        print Field.Value
    End If
Next Field

Item

Takes the fully qualified name of the field (for example, "name.last") as a parameter, and returns the Field object for it. It is the default member of the Fields interface. That is, item is the property invoked if the object name is specified by itself without a property or a method in the controller script.

Syntax

[read-only] IDispatch*
Example

Dim Field As AFORMAUTLib.Field
Dim nFields As Long

Set Field = Fields.Item("name.last")
'Since Item is the default property:
Set Field = Fields("name.last")

_NewEnum

The IEnumVariant enumerator for the collection.

You do not need to call this property directly. Visual Basic calls it in the background whenever the code contains a For Each Field In Fields loop. For example:

    For Each Field in Fields
    If Field.IsTerminal
        print Field.Value
    End If
    Next Field

Syntax

[read-only] IUnknown*
This chapter describes IAC support for the Acrobat DC Search plug-in, which allows users to perform text searches in PDF documents. It adds menus, menu items, toolbar buttons, and a Search panel to the Acrobat DC application. The Search plug-in exports a host function table (HFT) containing several methods that can be used by other Plugins.

Search supports interapplication communication in the form of DDE messages in Windows and Apple events in Mac OS. These messages and events allow remote clients to submit search queries and manipulate a list of indexes (the list of indexes is referred to as the shelf).

For more information, see the PDF Library documentation.

Search plug-in using DDE

A client can connect to the Search plug-in with DDE using the service name "Acrobat Search" and the topic name "Acrobat Search".

DdeInitialize(&id, _DDE_ProcessMessage, APPCMD_CLIENTONLY, 0);
hszServerName = DdeCreateStringHandle(id, "Acrobat Search", 0);
hszTopicName = DdeCreateStringHandle(id, "Acrobat Search", 0);
hConv = DdeConnect(id, hszServerName, hszTopicName, NULL);

After a connection has been made, a single poke transaction will submit a search query. Two types of queries are supported: simple query and query.

Simple query item

A simple query has the item name "SimpleQuery". When using a simple query, pass only a string that contains the query, using the ASQL query parser’s format (see QLangType_CQL in the table “Query language type constants” on page 240). It is not possible to choose another parser or to set word options using the simple query item.

Query item

A query has the item name “Query”. When using query, a QueryData structure is used. This structure contains the query, as well as specifying the query parser to use and additional options.

hszItemName = DdeCreateStringHandle(id, "Query", 0);
DdeClientTransaction(qd, nLen, hConv, hszItemName, CF_TEXT, XTYP_POKE, 1000, &dwResult);
DdeDisconnect(hConv)

The global data handle (qd) passed to the server must be in the following format:

typedef struct _QueryData {
    eQLangType qlt;
    boolean bOverrideWordOptions;
    uns32 nWordOptions;
    uns16 nMaxDocs;
} QueryData;
uns16 nQueryOffset;
uns16 nNumSorts; //deprecated in Acrobat 6.0
uns16 nSortOffset[QP_MAX_SORT_FIELDS]; //deprecated in Acrobat 6.0
boolean bSortWays[QP_MAX_SORT_FIELDS]; //deprecated in Acrobat 6.0
unsigned char cData[1];
} QueryData;

## Query options

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qlt</td>
<td>The query language type. Must be one of the values shown in “Query language type constants” on page 240.</td>
</tr>
<tr>
<td>bOverrideWordOptions</td>
<td>Indicates that the client wishes to use different word options than those currently set by the user.</td>
</tr>
<tr>
<td>nWordOptions</td>
<td>The word options. Must be an OR of the values shown in “Word option bit-flag constants” on page 241.</td>
</tr>
<tr>
<td>nMaxDocs</td>
<td>If non-zero, the client wishes to use a different limit for the maximum number of documents than the limit currently set by the user.</td>
</tr>
<tr>
<td>nSortOffsets</td>
<td>A list of offsets into the cData chunk. Each offset points to a NULL-terminated string containing the field name. This value has no effect in Acrobat DC 6.0 or later, because sort options are not valid.</td>
</tr>
<tr>
<td>nQueryOffset</td>
<td>An offset into the cData chunk that points to a NULL-terminated string containing the query to execute.</td>
</tr>
<tr>
<td>nNumSorts</td>
<td>The number of fields in the sort spec. If this number is 0, the plug-in uses the current sort spec set by the user. This value has no effect in Acrobat DC 6.0 or later, because sort options are not valid.</td>
</tr>
<tr>
<td>bSortWays</td>
<td>A list of sort order flags, one for each sort field. true indicates an ascending sort, and false indicates a descending sort. This value has no effect in Acrobat DC 6.0 or later, because sort options are not valid.</td>
</tr>
</tbody>
</table>

## Query language type constants

<table>
<thead>
<tr>
<th>Language Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLangType_Simple</td>
<td>Allows only simple phrase searches; does not allow Boolean searching. This query type does not work in the DDE interface of the Search plug-in shipped with version 2.0 of Acrobat DC.</td>
</tr>
<tr>
<td>QLangType_CQL</td>
<td>Allows Boolean searches using AND, OR, and NOT, as described in the Acrobat DC Search plug-in’s online help file.</td>
</tr>
<tr>
<td>QLangType_Passthrough</td>
<td>The Verity BooleanPlus query language. Contact Verity for further information on this language.</td>
</tr>
</tbody>
</table>
Word option bit-flag constants

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QPON_Case</td>
<td>The search is case-sensitive.</td>
</tr>
<tr>
<td>QPON_Stemming</td>
<td>Find not only the specified word, but other words that have the same stem.</td>
</tr>
<tr>
<td>QPON_SoundsLike</td>
<td>Find not only the specified word, but other words that sound like it.</td>
</tr>
<tr>
<td>QPON_Thesaurus</td>
<td>Find not only the specified word, but other words that have the same meaning.</td>
</tr>
<tr>
<td>QPON_Proximity</td>
<td>Consider the proximity of results when using the AND operator to look for more than one word in a document. Without this option, AND terms can be anywhere in a document. Searching for “red” and “blue,” for example, finds a document where “red” is the first word on the first page and where “blue” is the last word on the last page. With this option, however, AND terms must be within two or three pages of each other to be found. Also, the closer AND terms appear together, the higher the relevance ranking of the document that contains them.</td>
</tr>
<tr>
<td>QPON_Refine</td>
<td>Do not search the entire list of indexes, but only the documents that matched the previous search. This is used to refine the results of the previous search.</td>
</tr>
</tbody>
</table>

To create and populate this structure correctly, the client must know the sum of the lengths of each sort field (sls), the length of the query (lq), and the size of the QueryData structure. The client then allocates memory as follows:

\[
\text{nSize} = \text{sizeof(QueryData)} + \text{sls} + \text{lq}; \\
\text{qd} = (\text{QueryData} \times) \text{malloc(nSize)};
\]

For example, if the query was “Adobe” and the sort spec was “Title” ascending and “Score” descending then the structure would be packed as follows:

```c
memset(qd, 0, nSize);
qd->nQueryOffset = 0;
strcpy(&cData[0], "Adobe");
qd->nNumSort = 2;
qd->nSortOffset[0] = strlen("Adobe") + 1;
qd->bSortWays[0] = TRUE;
strcpy(&cData[qd->nSortOffset[0]], "Title");
qd->bSortWays[1] = FALSE;
qd->nSortOffset[1] = qd->nSortOffset[0] + strlen("Title") + 1;
strcpy(&cData[qd->nSortOffset[1]], "Score");
```

Manipulating indexes through DDE

After a connection has been made, a single poke transaction can add, delete, add, or remove indexes. The item name to use is “Index”.

```c
hszItemName = DdeCreateStringHandle(id, "Index", 0);
DdeClientTransaction(qd, nLen, hConv, hszItemName, CF_TEXT, XTYP_POKE, 1000, &dwResult);
DdeDisconnect(hConv);
```

The global data handle (gd) passed to the server must be in the following format:
typedef struct __IndexData {
    IndexActionType eAction;
    int16 nIndexOffset;
    int16 nTempNameOffset;
    unsigned char cData[1];
} IndexData;

Options

<table>
<thead>
<tr>
<th>eAction</th>
<th>The operation to be performed on the index. Must be one of values listed in “Index operation selectors” on page 242.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nIndexOffset</td>
<td>An offset into the cData chunk that points to a NULL-terminated string containing the PDX file representing the index.</td>
</tr>
<tr>
<td>nTempNameOffset</td>
<td>An offset into cData. It points to a temporary name that is displayed by the Search plug-in when the index is unavailable. This field must specify an offset either to an empty string (\0) or to a non-empty C string.</td>
</tr>
</tbody>
</table>

Index operation selectors

<table>
<thead>
<tr>
<th>IndexAction_Add</th>
<th>Adds an index to the shelf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexAction_Remove</td>
<td>Removes an index from the shelf.</td>
</tr>
<tr>
<td>IndexAction_Enable</td>
<td>Enables an index on the shelf.</td>
</tr>
<tr>
<td>IndexAction_Disable</td>
<td>Disables an index on the shelf.</td>
</tr>
</tbody>
</table>

To create and populate this structure correctly, the client must know the sum of the lengths of the Index (li) and Temp names (lt) (including NULL-terminating characters), and the size of the IndexData structure.

The client then allocates memory as follows:

```
nSize = sizeof(IndexData) + li + lt;
id = (IndexData *)malloc(nSize);
```

For example, to add the index C:\FOO.PDX to the Search plug-in’s shelf:

```
memset(id, 0, nSize);
id->eAction = IndexAction_Add;
id->nIndexOffset = 0;
strcpy(&id->cData[0], "C:\FOO.PDX");
id->nTempNameOffset = strlen("C:\FOO.PDX") + 1;
strcpy(&id->cData[id->nTempNameOffset], "My Favorite Index");
```

Search plug-in using Apple events

The Search plug-in supports the Apple events described in this section.

SearchAddIndex

Adds a specified index to the shelf.
Apple event ID
kSearchAddIndex ('addx')

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIndexListTag ('SilP'), typeLongInteger</td>
<td>An opaque void* representing the shelf, obtained from SearchGetIndexList.</td>
</tr>
<tr>
<td>kPathTag ('Path'), typeChar</td>
<td>Mac OS full path representing an index, of the form: MyDisk:TopFolder:BottomFolder:Strange.pdx</td>
</tr>
<tr>
<td>kFlagTag ('Flag'), typeLongInteger</td>
<td>Index flags. See SearchGetIndexFlags on page 245 for a description. The kIndexAvailable flag should always be set.</td>
</tr>
</tbody>
</table>

Returns

kIndexTag ('SixP'), typeLongInteger

An opaque void* representing an index. Returns NULL if failure.

Returns

#define kIndexExists ((SearchIndexPtr)-1)

if the index already exists in the index list. If the index already exists, you can retrieve it using SearchGetIndexByPath on page 245.

SearchCountIndexList

Gets the number of indexes currently on the shelf.

Apple event ID
kSearchCountIndexList ('cidx')

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIndexListTag ('SilP'), typeLongInteger</td>
<td>An opaque void* representing the shelf, obtained from SearchGetIndexList.</td>
</tr>
</tbody>
</table>

Returns

kIndexListTag ('SilP'), typeLongInteger

Number of indexes on the shelf (kIndexListTag here is not semantically correct, but works).

SearchDoQuery

Executes a specified query, using the set of indexes currently on the shelf. The search results are displayed in the Acrobat DC Search plug-in’s Results window.
Apple event ID

kSearchDoQuery ("kwry")

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kQueryStringTag (&quot;Quryv&quot;), typeChar</td>
<td>The query string, a NULL-terminated block of text. Its format is the same as what a user would type into the search Query window, and depends on the search language specified by kParserTag.</td>
</tr>
<tr>
<td>kParserTag (&quot;Prsr&quot;), typeShortInteger</td>
<td>The query parser to use; may be one of (see SrchType.h):</td>
</tr>
<tr>
<td>kParserSimple 0</td>
<td>Allows only simple phrase searches; does not allow Boolean searching.</td>
</tr>
<tr>
<td>kParserCQL 1</td>
<td>Allows Boolean searches using AND, OR, and NOT, as described in the Acrobat DC Search plug-in's online help file.</td>
</tr>
<tr>
<td>kParserBPlus 2</td>
<td>The Verity BooleanPlus query language. Contact Verity for further information on this language.</td>
</tr>
<tr>
<td>kSortSpecTag (&quot;Sort&quot;), typeAEList</td>
<td>A list of C strings representing fields to sort by. The first element is the first level sort, the second is the second level sort, and so forth. Each string may be any field that appears in the index, plus Score (which sorts results by relevance ranking). Some common fields are Title, ModificationDate, CreationDate, and Keywords.</td>
</tr>
<tr>
<td>kWordOptionsTag (&quot;WOpt&quot;), typeLongInteger</td>
<td>A bit field of word options. Must be a logical OR of the values listed below in &quot;Word options for Apple events&quot; on page 244. The manner in which the options are used depends on the value associated with kOptionsOverrideTag.</td>
</tr>
<tr>
<td>kOptionsOverrideTag (&quot;WOer&quot;), typeShortInteger</td>
<td>Flag that indicates whether the word options are OR'ed with the search options set in the user interface, or used instead of them. If 0, the word options are OR'ed with the user interface search options, and the resulting value is used. If non-zero, the word options are used instead of the user interface search options.</td>
</tr>
<tr>
<td>kMaxDocsTag (&quot;MaxD&quot;), typeShortInteger</td>
<td>The maximum number of documents to display in the Results window. If more documents than this have hits, only the first maxDocs are displayed. maxDocs cannot be greater than 999.</td>
</tr>
</tbody>
</table>

Word options for Apple events

<table>
<thead>
<tr>
<th>Word option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kWordOptionCase</td>
<td>The search is case-sensitive.</td>
</tr>
<tr>
<td>kWordOptionStemming</td>
<td>Find not only the specified word, but other words that have the same stem (for example, run and ran have the same stem).</td>
</tr>
<tr>
<td>kWordOptionSoundsLike</td>
<td>Find not only the specified word, but other words that sound like it.</td>
</tr>
<tr>
<td>kWordOptionThesaurus</td>
<td>Find not only the specified word, but other words that have the same meaning.</td>
</tr>
</tbody>
</table>
SearchGetIndexByPath

Gets the index that has the specified path. The index must already be on the shelf. The index can be passed to other Search Apple events to remove it from the shelf, obtain its title, and so forth.

Apple event ID
kSearchGetIndexByPath ('fpdx')

Parameters

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIndexListTag</td>
<td>'SilP', typeLongInteger An opaque void* representing the shelf, obtained from SearchGetIndexList.</td>
</tr>
<tr>
<td>kPathTag ('Path'), typeChar</td>
<td>Mac OS full path representing an index, of the form: MyDisk:TopFolder:BottomFolder:Strange.pdx</td>
</tr>
</tbody>
</table>

Returns

kIndexTag ('SixP'), typeLongInteger

An opaque void* representing an index. Returns NULL if the specified index is gone.

SearchGetIndexFlags

Get the flags for an index.

Apple event ID
kSearchGetIndexFlags ('gfdx')
Overview

SearchGetIndexList

Gets a list of the indexes currently on the shelf.

Apple event ID

kSearchGetIndexList ('gidx')

Returns

kIndexListTag ('SilP'), typeLongInteger

An opaque void* representing the list of indexes currently on the shelf. This value can subsequently be used by other search Apple events to obtain information about a specific index, the number of indexes on the shelf, and so forth.

SearchGetIndexPath

Gets the full path to an index.

Apple event ID

kSearchGetIndexPath ('gpdx')

Parameters

kIndexTag ('SixP'), typeLongInteger

An opaque void* representing the index whose path is to be obtained. The index may be obtained using SearchGetIndexPath, SearchGetNthIndex, or SearchAddIndex.

Returns

kPathTag ('Path'), typeChar

An opaque void* representing the index's path.
A NULL-terminated character string representing the full path of the index. Returns an empty string if the requested index is not valid.

**SearchGetIndexTitle**

Gets the title of an index.

**Apple event ID**

kSearchGetIndexTitle ('gtdx')

**Parameters**

| kIndexTag ('SixP'), typeLongInteger | An opaque void* representing the index whose title is to be obtained. The index may be obtained using SearchGetIndexByPath, SearchGetNthIndex, or SearchAddIndex. |

**Returns**

kTitleTag ('Title'), typeChar

A NULL-terminated character string representing the title of the index. If there is no title, it returns the index's path. Returns an empty string if the requested index is not valid.

**SearchGetNthIndex**

Gets the n<sup>th</sup> index on the shelf. The index can be passed to other Search Apple events to remove it from the shelf, obtain its title, and so forth.

**Apple event ID**

kSearchGetNthIndex ('fndx')

**Parameters**

| kIndexListTag ('SilP'), typeLongInteger | An opaque void* representing the shelf, obtained from SearchGetIndexList. |
| kNthIndexTag ('Enth'), typeLongInteger | The index to get. The first index on the shelf is index zero. |

**Returns**

kIndexTag ('SixP'), typeLongInteger

An opaque void* representing an index. Returns NULL if the n<sup>th</sup> index is gone.

**SearchRemoveIndex**

Removes the specified index from the shelf.
Apple event ID

kSearchRemoveIndex ('rmdx')

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIndexListTag ('SilP'),</td>
<td>An opaque void* representing the shelf, obtained from SearchGetIndexList.</td>
</tr>
<tr>
<td>typeLongInteger</td>
<td></td>
</tr>
<tr>
<td>kIndexTag ('SixP'),</td>
<td>An opaque void* representing the index to be removed. The index may be</td>
</tr>
<tr>
<td>typeLongInteger</td>
<td>obtained using SearchGetIndexByPath, SearchGetNthIndex, or SearchAddIndex.</td>
</tr>
<tr>
<td>kIndexTag ('SixP'),</td>
<td>An opaque void* representing an index.</td>
</tr>
<tr>
<td>typeLongInteger</td>
<td></td>
</tr>
<tr>
<td>kFlagTag ('Flag'),</td>
<td>Index flags. See the description in SearchGetIndexFlags. In practice, kIndex</td>
</tr>
<tr>
<td>typeLongInteger</td>
<td>AvailableFlag should always be set.</td>
</tr>
</tbody>
</table>

Returns

kFlagTag ('Flag'), typeLongInteger

Index flags. See the description in “SearchGetIndexFlags” on page 245. This value is returned because it is possible for a request to set a flag to fail.

Search lists

The Search plug-in adds a new menu, menu items, and toolbar buttons to the Acrobat DC application.

Menu names

The Search plug-in adds the following menu to Acrobat DC.

<table>
<thead>
<tr>
<th>Menu name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcroSrch:ToolsSubMenu</td>
<td>Acrobat DC Search submenu of Edit menu</td>
</tr>
</tbody>
</table>
### Menu item names

The Search plug-in adds the following menu items to Acrobat DC.

<table>
<thead>
<tr>
<th>Menu item name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcroSrch:Query</td>
<td>Displays the Search dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Indexes</td>
<td>Displays the Index dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Results</td>
<td>Displays the Results dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Assist</td>
<td>Displays the Word Assistant dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Separator</td>
<td>A separator item in the Search tools menu.</td>
</tr>
<tr>
<td>AcroSrch:PrevDoc</td>
<td>Goes to the previous document in the hit list.</td>
</tr>
<tr>
<td>AcroSrch:PrevHit</td>
<td>Goes to the previous hit in the hit list.</td>
</tr>
<tr>
<td>AcroSrch:NextHit</td>
<td>Goes to the next hit in the hit list.</td>
</tr>
<tr>
<td>AcroSrch:NextDoc</td>
<td>Goes to the next document in the hit list.</td>
</tr>
</tbody>
</table>

### Toolbar button names

The Search plug-in adds the following buttons to the Acrobat DC toolbar.

<table>
<thead>
<tr>
<th>Button name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcroSrch:Separator</td>
<td>Separator (not visible).</td>
</tr>
<tr>
<td>AcroSrch:Query</td>
<td>Displays the Acrobat DC Search plug-in’s query dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Results</td>
<td>Displays the Acrobat DC Search plug-in’s search results dialog box.</td>
</tr>
<tr>
<td>AcroSrch:Prev</td>
<td>Goes to the previous hit in the Acrobat DC Search plug-in’s results list.</td>
</tr>
<tr>
<td>AcroSrch:Next</td>
<td>Goes to the next hit in the Acrobat DC Search plug-in’s results list.</td>
</tr>
</tbody>
</table>
11 Coordinate Systems

User space

The user space is the coordinate system used within PDF files. In the IAC interface, it is used for most PD layer objects (that is, objects such as PDBookmark whose names begin with “PD”). The following graphic shows the user space coordinate system. The orientation, origin, and scale of the user space coordinate system can be changed by operators in the page description in a PDF file.

![User space coordinate system](image)

The default user space is the user space coordinate system in effect immediately before each page begins drawing. The origin of this coordinate system is the lower left corner of a page’s media box. The x-coordinate increases to the right, and the y-coordinate increases upward. One unit in the default user space is 1/72 of an inch.

Device space

The device space specifies coordinates in screen pixels, as shown in the following graphic. It is used in the AV layer of the IAC interface (that is, objects such as AVDoc whose names begin with “AV”).
The origin of the device space coordinate system is at the upper left corner of the visible page on the screen (that is, the upper left corner of the white part of the page). The x-coordinate increases to the right, and the y-coordinate increases downward.

The upper left corner of the visible page is determined by the intersection of a page’s PDF crop box and media box. As a result, the device space coordinate system changes if the cropping on a page changes.
Index

A
accessing
  annotations 15
text 15
AcquirePage method 100
Acrobat application events 191
Acrobat type library 23, 25, 29
acrobat.cpp 25
acrobat.h 25
acrobat.tlb 25
AcroExch.App 42
AcroExch.AVDoc 58
AcroExch.AVPageView 76
AcroExch.HiliteList 85
AcroExch.PDAnnot 85
AcroExch.PDBookmark 95
AcroExch.PDDoc 98
AcroExch.PDPage 115
AcroExch.PDTextSelect 128
AcroExch.Point 132
AcroExch.Rect 132
AcroExch.Time 134
AcroPDF object 21
ActiveX document 16
ActiveX documents 13, 21
Add method 85, 232
AddAnnot method 116
AddDocJavascript method 234
adding references 29
AddNewAnnot method 117
Adobe Reader
  Apple events 185
  browser control 21
  DDE support 151
  OLE support 20
AFORMApp object 214
Alignment property 223
annotation object 171
annotations
  accessing 15
  creating 15
  example 31
  manipulating 15
API layers 13
App object 42
appearance of Acrobat, controlling 14
AppExit message 151
AppExit method 211
AppFront method 211
AppHide message 152
Apple events 41
Apple events and objects 171
AppleScript 19
application layer objects 14
application object 172
AppShow message 152
AV layer
  description 13
  objects 14
AVApp object 14
AVConversion object 14
AVDoc object 13, 14, 21, 58
AVMenu object 14
AVMenuItem object 14
AVPageView object 14, 76, 175
AxacroPDF object 136
AxacroPDFLib.AxacroPDF 136
B
bookmark object 15, 175
BorderStyle property 224
BorderWidth property 224
Bottom property 133
bridge methods 14, 39
bring to front event 192
BringToFront method 59
browser controls 21
ButtonLayout property 225
C
C 24
CAcro classes 25
CalcOrderIndex property 225
case sensitivity 36, 40
Catalog plug-in 211
CharLimit property 225
checking spelling 34
child windows 37
clear selection event 193
ClearFlags method 100
ClearSelection method 60
client side implementation 25
close all docs event 193
close event 186
Close method 60, 101
CloseAllDocs message 152
CloseAllDocs method 44
COleDispatchDriver
  class 25
  objects and methods 25
controlling, Acrobat appearance 14
convenience functions 25
conversion object 177
converting documents 14
diplimate systems 250
CopyToClipboard method 117
Core suite events 186
count event 186
Count property 237
counting
  menus 14
  pages 15
Create method 101
create thumbs event 194
CreateDispatch statement 13
CreateObject statement 13
CreateObjSpecifier statement 13
CreatePageHilite method 118
CreateTextSelect method 102
CreateThumbs method 103
CreateWordHilite method 119
creating
  annotations 15
  Plugins 16
  simple application 30
  thumbnails 15
CropPage method 120
CropPages method 103

D
Date property 134
DDE
  Adobe Reader support 151
  messages 150
  messages, setting up 40
  overview 40
DDE messages 19
DefaultValue property 226
delete event 187
delete pages event 195
delete thumbs event 195
DeletePages method 104
DeleteThumbs method 104
deleting
  bookmarks 15
  pages 15
  thumbnails 15
Destroy method 95, 128
development environment
  choosing 22
  configuration 23
device space 250
DevicePointToPage method 76
displaying documents 21
do script event 210
DocClose message 153
DocDeletePages message 153
DocFind message 154
DocGoTo message 154
DocGoToNameDest message 155
DocInsertPages message 155
DocOpen message 156
DocPageDown message 157
DocPageLeft message 157
DocPageRight message 158
DocPageUp message 158
DocPrint message 159
DocReplacePages message 159
DocSave message 160
DocSaveAs message 161
DocScrollTo message 161
DocSetViewMode message 162
document object 177
documents
  displaying 21
  information fields 15
  loading 21
  opening 21
  pages 15
  printing 14
DocZoomTo message 162
DoGoBack method 77
DoGoForward method 77
Draw method 120
DrawEx method 21, 121
dual interfaces 42

E
Editable property 226
EPS Conversion object 179
events
  Acrobat application 191
  Core suite 186
  miscellaneous 210
  Required suite 184
events and child windows 37
exceptions, Forms plug-in 213
execute event 196
ExecuteThisJavascript method 234
exists event 187
Exit method 44
executing an OLE application 38
ExportAsFDF method 235
ExportAsHtml method 236
extending with Plugins 16

F
Field object 214
Fields collection 232
file format object 14
FileBuild method 212
FileOpen message 163
FileOpen method 212
FileOpenEx message 163
FilePrint message 164
FilePrintEx message 165
FilePrintSilent message 166
FilePrintSilentEx message 166
FilePrintTo message 167
FilePrintToEx message 168
FilePurge method 212
find next note event 196
find text event 197
finding text 14
FindText method 61  
FindWindow method 38  
Forms plug-in 213  
FullMenus message 168  

G  
get event 188  
get info event 198  
GetActiveDoc method 45  
GetActiveTool method 45  
GetAnnot method 122  
GetAnnotIndex method 123  
GetAperture method 78  
GetAVDoc method 46, 78  
GetAVPageView method 62  
GetBoundingRect method 129  
GetByTitle method 96  
GetColor method 86  
GetContents method 87  
GetDate method 87  
GetDoc method 78, 123  
GetFileName method 105  
GetFlags method 105  
GetFrame method 46, 62  
GetInfo method 106  
GetInstanceID method 106  
GetInterface method 47  
GetJSObject method 107  
GetLanguage method 47  
GetNumAnnots method 124  
GetNumAVDocs method 48  
GetNumber method 124  
GetNumPages method 107  
GetNumText method 129  
GetPage method 79, 130  
GetPageMode method 108  
GetPageNum method 79  
GetPDDoc method 62  
GetPermanentID method 108  
GetPreference method 48  
GetPreferenceEx method 49  
GetRect method 87  
GetRotate method 125  
GetSize method 125  
GetSubtype method 88  
GetText method 131  
getting  
annotations 15  
document fields 15  
page information 15  
GetTitle method 63, 88, 96  
GetVersions method 138  
GetViewMode method 63  
GetZoom method 80  
GetZoomType method 80  
go backward event 198  
go forward event 199  
GoBackwardStack method 138  
GoForwardStack method 138  
goto event 199  
Goto method 81  
goto next event 200  
goto previous event 201  
GotoFirstPage method 138  
GotoLastPage method 139  
GotoNextPage method 139  
GotoPreviousPage method 139  

H  
handling events in child windows 37  
header files 23  
HFT 239  
Hide method 49  
HideToolbar message 169  
Highlight property 226  
HiliteList object 85  
history object 14  
host function table 239  
Hour property 135  

I  
ImportAnFDF method 236  
index, Catalog plug-in 211  
insert pages event 201  
InsertPages method 108  
interfaces 25  
is toolbutton enabled event 202  
IsEqual method 89  
IsHidden property 227  
IsMultiline property 227  
IsOpen method 89  
IsPassword property 228  
IsReadOnly property 228  
IsRequired property 228  
IsTerminal property 228  
IsValid method 64, 90, 97  
Item property 237  

J  
JavaScript  
interface 29  
translating to JSObject 36  
JSObject  
example code 30, 31, 34  
JavaScript tips 36  
overview 29  
type library reference 29  

L  
layers 13  
Left property 133  
Link Annotation object 179  
link annotations 15  
link object 15  
LoadFile method 21, 140  
loading a document 21  
Lock method 49
LPDISPATCH pointer 25

M
magnifying 14
make event 188
manipulating
  bookmarks 15
  link annotations 15
  text annotations 15
maximize event 203
Maximize method 51, 64
MDI applications 37
menu item object 14, 180
menu object 14, 179
MenuItemExecute message 169
MenuItemExecute method 51
MenuItemsEnabled method 52
MenuItemsMarked method 52
MenuItemRemove method 53
messaging, synchronous 37
methods 39
Milliseconds property 135
Minimize method 50
Minute property 135
Month property 135
move event 189
MovePage method 109
multiple document interfaces 37

N
Name property 229
navigating pages 14
_NewEnum property 238
NoViewFlag property 229

O
object layers 13
object reference syntax 13
objects 39
OLE
  Adobe Reader support 20
    exiting 38
    on-screen rendering 21
    PDF browser controls 21
    remote control 21
OLE automation 42
on-screen rendering 21
open event 185, 190
Open method 64, 110
OpenAVDoc method 110
opening documents 21
OpenInWindow method 65
OpenInWindowEx method 21, 66

P
page navigation 14
Page object 21
page object 181
page representation 15
pages
  counting 15
  deleting 15
  getting information 15
PD layer
  description 13
  objects 15
PDAannot object 15, 85, 182
PDBookMark object 182
PDBookmark object 15, 95
PDDoc object 15, 29, 31, 98
PDF browser controls 16, 21
PDF document object 15
PDF file object 14
PDF Window object 182
PDLinkAnnot object 182
PDFPage object 13, 15, 115, 182
PDTextAnnot object 182
PDTextSelect object 15, 128
perform event 203
Perform method 91, 97
Plugins 16
  Catalog 211
  Forms 213
  Search 239
Point object 132
PointToDevice method 82
PopulateListOrComboBox method 214
portable document layer objects 15
PostScript Conversion object 183
print event 185
Print method 140
print pages event 204
PrintAll method 141
PrintAllFit method 141
PrintFlag property 229
printing 14
PrintPages method 68, 142
PrintPagesEx method 69
PrintPagesFit method 142
PrintPagesSilent method 70
PrintPagesSilentEx method 70
PrintWithDialog method 143

Q
queries 239
quit event 185, 190

R
read page down event 205
read page up event 205
ReadPageDown method 82
ReadPageUp method 83
Rect object 132
references, adding 29
remote control 21
Remove method 236
remove toolbutton event 206
RemoveAnnot method 126
removing
   menu items 14
   menus 14
rendering Acrobat 15
replace pages event 206
ReplacePages method 111
replacing pages 15
Required suite events 184
Restore method 53
Right property 133
run event 185
running applications 38

S
save event 190
Save method 112
saving, conversion 14
scroll event 207
scrolling 14
ScrollTo method 83
search lists 248
Search plug-in 239
SearchAddIndex event 242
SearchCountIndexList event 243
SearchDoQuery event 243
SearchGetIndexByPath event 245
SearchGetIndexFlags event 245
SearchGetIndexList event 246
SearchGetIndexPath event 246
SearchGetIndexTitle event 247
SearchGetNthIndex event 247
SearchRemoveIndex event 247
SearchSetIndexFlags event 248
Second property 136
select text event 208
select text object 15
selecting text 14, 15
set event 191
set info event 209
set...to statement 13
SetActiveTool method 54
SetBackgroundColor method 215
SetBorderColor method 216
SetButtonCaption method 217
SetButtonIcon method 217
SetColor method 91
SetContents method 92
SetCurrentHighlight method 143
SetCurrentPage method 144
SetDate method 92
SetExportValues method 218
SetFlags method 113
SetForegroundColor method 219
SetFrame method 54, 71
SetInfo method 114
SetJavaScriptAction method 220
SetLayoutMode method 144
SetNameDest method 145
SetOpen method 93
SetPageMode method 114, 145
SetPreference method 55
SetPreferenceEx method 55
SetRect method 93
SetResetFormAction method 221
SetRotate method 126
SetShowScrollbars method 146
SetShowToolbar method 146
SetSubmitFormAction method 222
SetTextSelection method 72
setting
   document fields 15
text regions 15
SetText method 73, 94, 98
SetView method 146
SetViewMode method 73
SetViewRect method 147
SetViewScroll method 148
SetZoom method 148
SetZoomScroll method 149
ShortMenus message 169
Show method 56
ShowTextSelect method 74
ShowToolbar message 170
spell-checking 34
Src property 149
Style property 230
synchronous messaging 37
text, object references 13

T
Text Annotation object 184
text annotation object 15
text regions 15
text searches 239
text selection object 15
TextFont property 230
TextSize property 231
Time object 134
ToolButtonIsEnabled method 56
ToolButtonRemove method 57
top property 134
top-level object 14
translating JavaScript to JSObject 36
type library file 23, 25, 29
Type property 231

U
Unlock method 57
UnlockEx method 58
user space 250

V
Value property 231

W
window content object 14
window object 14
wrapper functions 25

X
X property 132

Y
Y property 132
Year property 136

Z
zoom event 209
ZoomTo method 84