Interapplication Communication with the Acrobat SDK

Contents

List of Examples

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 en3bles 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.

Object	Description	OLE automation class name	Apple event class name
`AVApp`	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.	`AcroExch.App`	`Application`
`AVDoc`	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.	`AcroExch.AVDoc`	`Document`
`AVPageView`	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.	`AcroExch.AVPageView`	`PDF Window`
`AVMenu`	Represents a menu in Acrobat DC. You can count or remove menus. Each menu has a language-independent name used to access it.	None	`Menu`
`AVMenuItem`	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.	None	`Menu item`
`AVConversion`	Represents the format in which to save the document.	None	`conversion`

Objects in the portable document layer

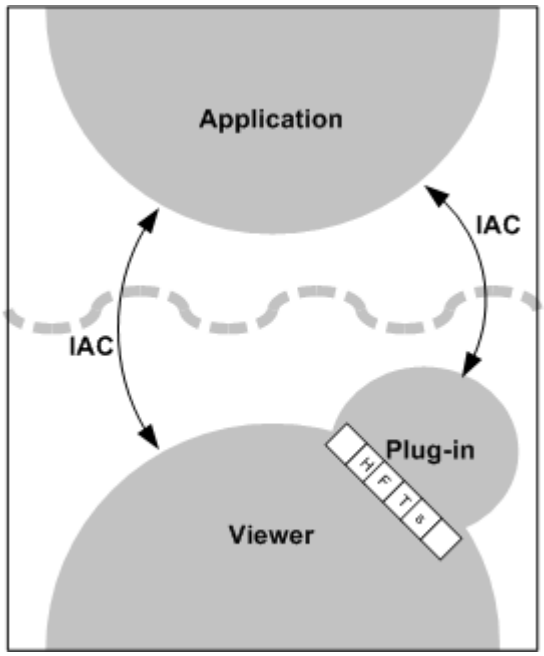
This table describes the IAC objects in the portable document (PD) layer.

Object	Description	OLE automation class name	Apple event class name
`PDDoc`	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.	`AcroExch.PDDoc`	`Document`
`PDPage`	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.	`AcroExch.PDPage`	`page`
`PDAnnot`	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.	`AcroExch.PDAnnot`	`annotation`
`PDBookmark`	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.	`AcroExch.PDBookmark`	`bookmark`
`PDTextSelect`	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.	`AcroExch.PDTextSelect`	None

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



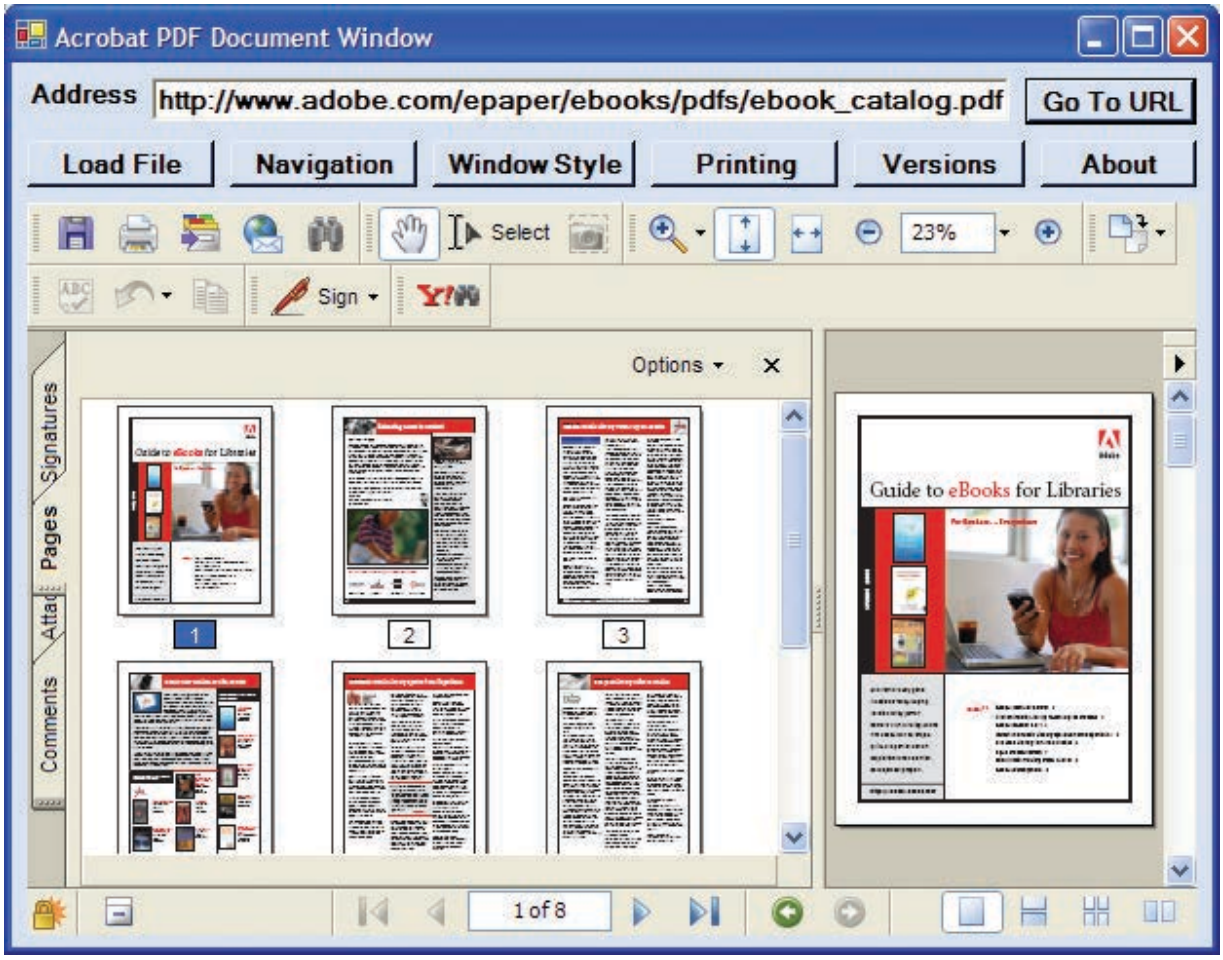
Similarly, the `JSObject` interface provides you with convenient access to the Acrobat DC features made available through JavaScript. Take advantage of this interface wherever possible. Its usage is explained in "Using the JSObject interface" on page 29.

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



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`

2 Using OLE

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*.

For more information on OLE 2.0 and OLE automation, see the *OLE Automation Programmer's Reference*, ISBN 1-55615-851-3, Microsoft Press. You can also find numerous articles at http://msdn.microsoft.com.

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++*

```
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*

```
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"),
     System.Windows.Forms.AxHost.State
)
```

```
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:

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

// Attributes
public:

// Operations
public:
   bool Add(short nOffset, short nLength);
};
```

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

```
   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*

```
// 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\n1st 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:

```
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:

```
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:

Class	String
`CAcroPoint`	`"AcroExch.Point"`
`CAcroRect`	`"AcroExch.Rect"`
`CAcroTime`	`"AcroExch.Time"`
`CAcroApp`	`"AcroExch.App"`
`CAcroPDDoc`	`"AcroExch.PDDoc"`
`CAcroAVDoc`	`"AcroExch.AVDoc"`
`CAcroHiliteList`	`"AcroExch.HiliteList"`
`CAcroPDBookmark`	`"AcroExch.PDBookmark"`
`CAcroMatrix`	`"AcroExch.Matrix"`
`AcroPDF`	`"AxAcroPDFLib.AxAcroPDF"`

Refer again to this code from the previous example:

```
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.

```
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.

```
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.

```
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*

```
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

```
jso.console.clear
```

with

```
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.

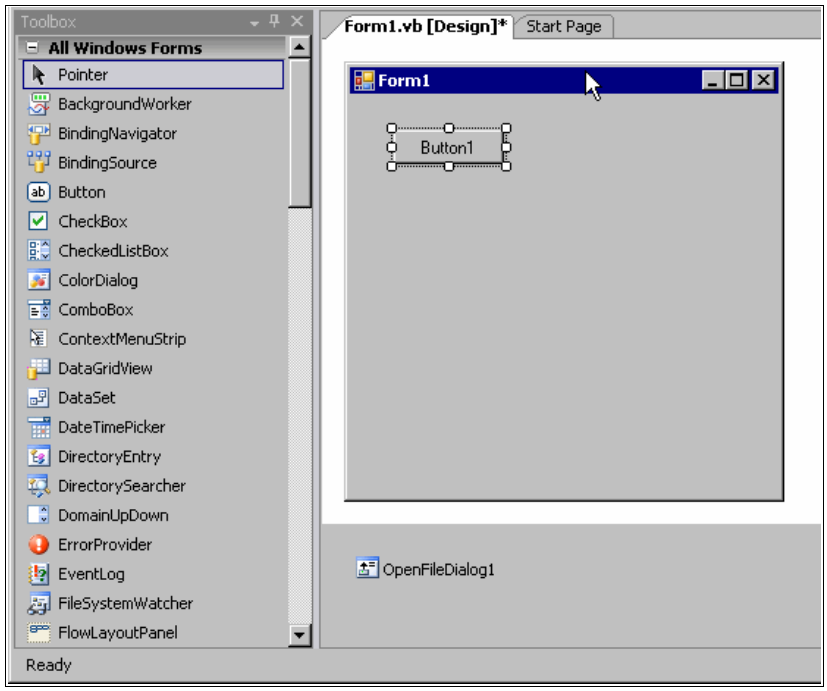
Working with annotations

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.



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

Example: ***Adding an annotation***

```
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:

```
annot = jso.AddAnnot
annot.Type = "Text"
annot.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.

```
dim color(0 to 3) as Variant
color(0) = "RGB"
color(1) = 1#
color(2) = 0#
color(3) = 0#
annot.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 Variants, 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
            End If
         Next i
         jso = Nothing
         pdDoc.Close

         If Not foundErr Then
            ListView1.Items.Add ("No spelling errors found in " & path)
         End If
      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*

```
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:

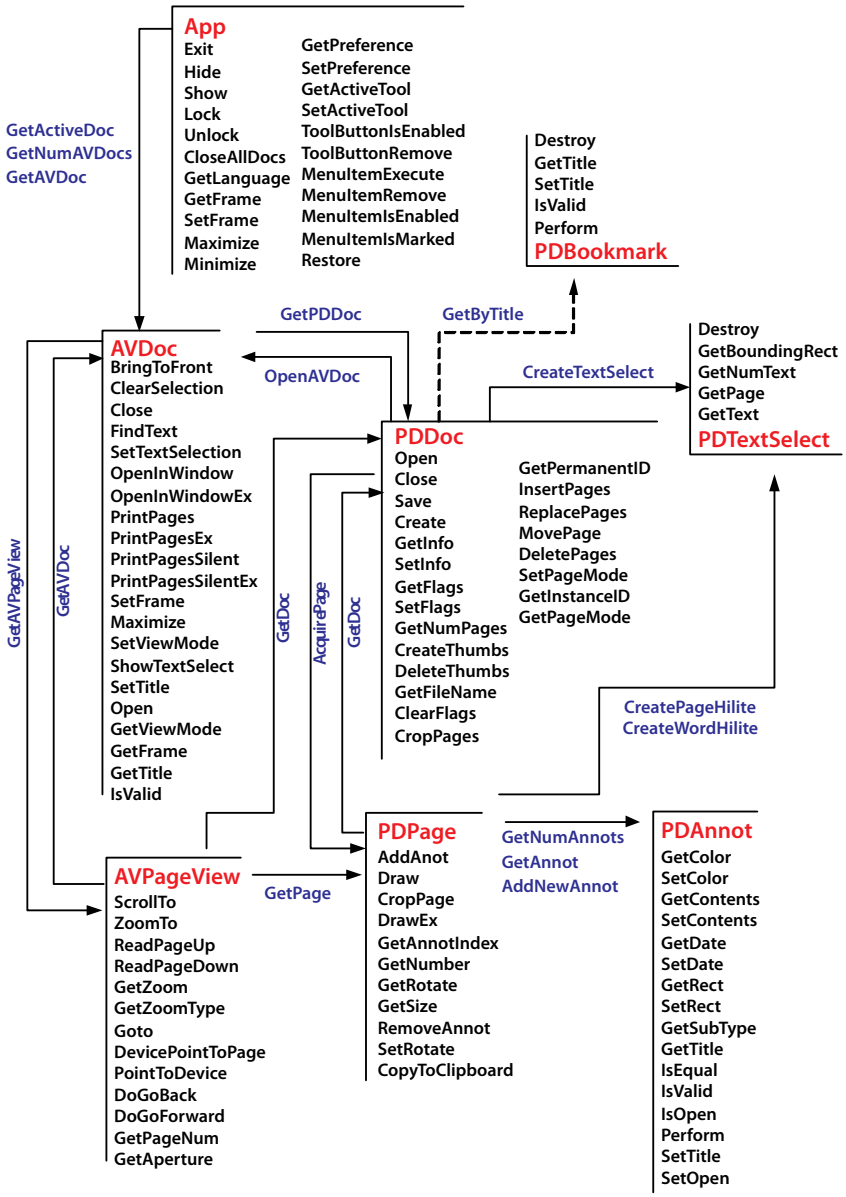
- If no PDF documents are open in Acrobat DC, the application quits.
- If a web browser is displaying a PDF document, the display goes blank. The user can refresh the page to redisplay it.

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.

OLE objects and methods



For complete descriptions, see the OLE automation sections of the *IAC API Reference*.

3 Using DDE

Acrobat DCAlthough 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*

```
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.

4 Using Apple Events

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:

Category	Description
Required events	Events that the Finder sends to all applications.
Core events	Events that are common to a wide variety of applications, though not universally applicable to all applications.
Acrobat DC-specific events	Events that are specific to Acrobat DC.
Miscellaneous Apple events	Events that are not in one of the preceding categories.

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*.

For more information on Apple events and scripting, see *Inside Macintosh: Interapplication Communication*, ISBN 0-201-62200-9, Addison-Wesley. The content of this document is currently available at http://developer.apple.com/documentation/mac/IAC/IAC-2.html.

For more information on the AppleScript language, see the *AppleScript Language Guide*, ISBN 0-201-40735-3, Addison-Wesley. The content of this document is currently available at http://developer.apple.com/documentation/AppleScript/Conceptual/AppleScriptLangGuide/.

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/.

5 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.PDAnnot	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.

Method	Description
CloseAllDocs	Closes all open documents.
Exit	Exits Acrobat DC.
GetActiveDoc	Gets the frontmost document.
GetActiveTool	Gets the name of the currently active tool.
GetAVDoc	Gets an `AcroExch.AVDoc` object via its index within the list of open `AVDoc` objects.
GetFrame	Gets the window's frame.
GetInterface	Gets an `IDispatch` interface for a named object, typically a third-party plug-in.
GetLanguage	Gets a code that specifies which language the Acrobat DC application's user interface is using.
GetNumAVDocs	Gets the number of open `AcroExch.AVDoc` objects.
GetPreference	Gets a value from the preferences file.
GetPreferenceEx	Gets the specified application preference, using the VARIANT type to pass values.
Hide	Hides the Acrobat DC application.
Lock	Locks the Acrobat DC application.
Minimize	Minimizes the Acrobat DC application.
Maximize	Maximizes the Acrobat DC application.
MenuItemExecute	Executes the menu item whose language-independent menu item name is specified.
MenuItemIsEnabled	Determines whether the specified menu item is enabled.
MenuItemIsMarked	Determines whether the specified menu item is marked.
MenuItemRemove	Removes the menu item whose language-independent menu item is specified.
Restore	Restores the main window of the Acrobat DC application.
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).
SetFrame	Sets the window's frame to the specified rectangle.
SetPreference	Sets a value in the preferences file.

Method	Description
SetPreferenceEx	Sets the application preference specified by nType to the value stored at pVal.
Show	Shows the Acrobat DC application.
ToolButtonIsEnabled	Determines whether the specified toolbar button is enabled.
ToolButtonRemove	Removes the specified button from the toolbar.
Unlock	Unlocks the Acrobat DC application if it was previously locked.
UnlockEx	Unlocks the Acrobat DC application if it was previously locked.

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 objects'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 `AcroExch.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 `AcroExch.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

```
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

```
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`

MenuItemIsEnabled

Determines whether the specified menu item is enabled.

Syntax

```
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`

MenuItemIsMarked

Determines whether the specified menu item is marked.

Syntax

```
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

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

Parameters

`szMenuItemName`	The language-independent name of the menu item to remove. See the *PDF Library documentation* 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

```
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

`szButtonName`	The name of the tool to set as the active tool. See the *PDF Library documentation* for a list of tool names.
`bPersistent`	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.

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

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

Parameters

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

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

```
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.

Syntax

```
VARIANT_BOOL Unlock();
```

Returns

`-1` if successful, `0` if not.

Related methods

`App.Lock`

`App.UnlockEx`

UnlockEx

Unlocks the Acrobat DC application if it was previously locked.

Syntax

```
VARIANT_BOOL UnlockEx (BSTR szLockedBy);
```

Parameters

`szLockedBy`	A string indicating the name of the application to be unlocked.

Returns

`-1` if successful, `0` if not.

Related methods

`App.Lock`

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.

Method	Description
`BringToFront`	Brings the window to the front.
`ClearSelection`	Clears the current selection.

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

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

`bNoSave`	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.

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

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

Parameters

`szText`	The text to be found.
`bCaseSensitive`	If a positive number, the search is case-sensitive. If `0`, it is case-insensitive.
`bWholeWordsOnly`	If a positive number, the search matches only whole words. If `0`, it matches partial words.
`bReset`	If a positive number, the search begins on the first page of the document. If `0`, it begins on the current page.

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

`bMaxSize`	Indicates whether the window should be maximized.

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

`szFullPath`	The full path of the file to open.
`szTempTitle`	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.

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

`fileName`	The full path of the file to open.
`hWnd`	Handle for the window in which the file is displayed.

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

`szFullPath`	The full path of the file to open.
`hWnd`	Handle for the window in which the file is displayed.

`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

```
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

`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`	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 may 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.
`bReverse`	(PostScript printing only) If a positive number, print the pages in reverse order. If false, print the pages in the regular order.
`bFarEastFontOpt`	(PostScript printing only) Set to a positive number if the destination printer has multibyte fonts; set to `0` otherwise.
`bEmitHalftones`	(PostScript printing only) If a positive number, emit the halftones specified in the document. If `0`, do not.
`iPageOption`	Pages in the range to print. Must be one of: `PDAllPages`, `PDEvenPagesOnly`, or `PDOddPagesOnly`.

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

`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`	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 may 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.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

`nFirstPage`	The first page to be printed.
`nLastPage`	The last page to be printed.
`nPSLevel`	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 may 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.
`bReverse`	(PostScript printing only) If a positive number, print the pages in reverse order. If false, print the pages in the regular order.
`bFarEastFontOpt`	(PostScript printing only) Set to a positive number if the destination printer has multibyte fonts; set to `0` otherwise.
`bEmitHalftones`	(PostScript printing only) If a positive number, emit the halftones specified in the document. If `0`, do not.
`iPageOption`	Pages in the range to print. Must be one of: `PDAllPages`, `PDEvenPagesOnly`, or `PDOddPagesOnly`.

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.GetFrame`

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.ShowTextSelect` to show the selection.

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.ClearSelection`

`AVDoc.ShowTextSelect`

`PDDoc.CreateTextSelect`

`PDPage.CreatePageHilite`

`PDPage.CreateWordHilite`

PDTextSelect.Destroy

PDTextSelect.GetBoundingRect

PDTextSelect.GetNumText

PDTextSelect.GetPage

PDTextSelect.GetText

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

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

Syntax

```
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.

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

DevicePointToPage

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

Syntax

```
LPDISPATCH DevicePointToPage(LPDISPATCH iAcroPoint);
```

Parameters

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

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

```
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

```
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

`nX`	The x–coordinate of the destination.
`nY`	The y–coordinate of the destination.

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

nType	Zoom type. Possible values are: AVZoomFitHeight — Fits the page's height into the window. AVZoomFitPage — Fits the page into 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%.
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

```
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.IsEqual

PDAnnot.IsValid

PDAnnot.SetOpen

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.IsEqual

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`

PDAnnot.SetOpen

PDAnnot.SetRect

PDAnnot.SetTitle

SetContents

Sets a text annotation's contents.

Syntax

```
VARIANT_BOOL SetContents(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.GetContents

PDAnnot.SetColor

PDAnnot.SetDate

PDAnnot.SetOpen

PDAnnot.SetRect

PDAnnot.SetTitle

SetDate

Sets an annotation's date.

Syntax

```
VARIANT_BOOL SetDate(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.GetTitle`

`PDAnnot.SetColor`

`PDAnnot.SetContents`

`PDAnnot.SetOpen`

`PDAnnot.SetRect`

`PDAnnot.SetTitle`

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`

`PDAnnot.SetTitle`

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.GetRect`

`PDAnnot.SetColor`

`PDAnnot.SetContents`

`PDAnnot.SetDate`

`PDAnnot.SetOpen`

`PDAnnot.SetTitle`

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`

`PDAnnot.SetColor`

`PDAnnot.SetContents`

`PDAnnot.SetDate`

`PDAnnot.SetOpen`

`PDAnnot.SetRect`

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.

Method	Description
`Destroy`	Destroys a bookmark.
`GetByTitle`	Gets the bookmark that has the specified title.
`GetTitle`	Gets a bookmark's title.
`IsValid`	Determines whether the bookmark is valid.
`Perform`	Performs a bookmark's action.
`SetTitle`	Sets a bookmark's title.

Destroy

Destroys a bookmark.

Syntax

```
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

```
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();
```

Returns

The title.

Related methods

`PDBookmark.GetByTitle`

`PDBookmark.SetTitle`

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.Destroy`

Syntax

Perform

Performs a bookmark's action.

Syntax

```
VARIANT_BOOL Perform(LPDISPATCH iAcroAVDoc);
```

Parameters

`iAcroAVDoc`	The `LPDISPATCH` for the `AcroExch.AVDoc` in which the bookmark 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

`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.

Method	Description
`AcquirePage`	Acquires the specified page.
`ClearFlags`	Clears a document's flags.
`Close`	Closes a file.

Method	Description
Create	Creates a new AcroExch.PDDoc.
CreateTextSelect	Creates a text selection from the specified rectangle on the specified page.
CreateThumbs	Creates thumbnail images for the specified page range in a document.
CropPages	Crops the pages in a specified range in a document.
DeletePages	Deletes pages from a file.
DeleteThumbs	Deletes thumbnail images from the specified pages in a document.
GetFileName	Gets the name of the file associated with this AcroExch.PDDoc.
GetFlags	Gets a document's flags.
GetInfo	Gets the value of a specified key in the document's Info dictionary.
GetInstanceID	Gets the instance ID (the second element) from the ID array in the document's trailer.
GetJSObject	Gets a dual interface to the JavaScript object associated with the PDDoc.
GetNumPages	Gets the number of pages in a file.
GetPageMode	Gets a value indicating whether the Acrobat DC application is currently displaying only pages, pages and thumbnails, or pages and bookmarks.
GetPermanentID	Gets the permanent ID (the first element) from the ID array in the document's trailer.
InsertPages	Inserts the specified pages from the source document after the indicated page within the current document.
MovePage	Moves a page to another location within the same document.
Open	Opens a file.
OpenAVDoc	Opens a window and displays the document in it.
ReplacePages	Replaces the indicated pages in the current document with those specified from the source document.
Save	Saves a document.
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.
SetInfo	Sets the value of a key in a document's Info dictionary.
SetPageMode	Sets the page mode in which a document is to be opened: display only pages, pages and thumbnails, or pages and bookmarks.

AcquirePage

Acquires the specified page.

Syntax

```
LPDISPATCH AcquirePage(long nPage);
```

Parameters

`nPage`	The number of the page to acquire. The first page is page 0.

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

`nPage`	The page on which the selection is created. The first page in a `PDDoc` object is page 0.
`iAcroRect`	The `LPDISPATCH` for the `AcroExch.Rect` enclosing the region to select. `iAcroRect` contains the instance variable `m_lpDispatch`, which contains the `LPDISPATCH`.

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

`nFirstPage`	First page for which thumbnail images are created. The first page in a `PDDoc` object is page `0`.
`nLastPage`	Last page for which thumbnail images are created.

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

`nStartPage`	First page that is cropped. The first page in a `PDDoc` object is page 0.
`nEndPage`	Last page that is cropped.
`nEvenOrOddPagesOnly`	Value indicating which pages in the range are cropped. Must be one of the following: `0` — crop all pages in the range `1` — crop only odd pages in the range `2` — crop only even pages in the range
`iAcroRect`	An `LPDISPATCH` for a `CAcroRect` specifying the cropping rectangle, which is specified in user space.

Returns

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

Related methods

`PDPage.CropPages`

DeletePages

Deletes pages from a file.

Syntax

```
VARIANT_BOOL DeletePages(long nStartPage, long nEndPage);
```

Parameters

`nStartPage`	The first page to be deleted. The first page in a `PDDoc` object is page `0`.
`nEndPage`	The last page to be deleted.

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);
```

Parameters

`nStartPage`	First page whose thumbnail image is deleted. The first page in a `PDDoc` object is page `0`.
`nEndPage`	Last page whose thumbnail image is deleted.

Returns

`-1` if the thumbnails were deleted, `0` if they were not deleted or if the Acrobat DC application does not support editing.

Related methods

`PDDoc.CreateThumbs`

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.Save`

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:

Flag	Description
`PDDocNeedsSave`	Document has been modified and needs to be saved.

`PDDocRequiresFullSave`	Document cannot be saved incrementally; it must be written using `PDSaveFull`.
`PDDocIsModified`	Document has been modified slightly (such as bookmarks or text annotations have been opened or closed), but not in a way that warrants saving.
`PDDocDeleteOnClose`	Document is based on a temporary file that must be deleted when the document is closed or saved.
`PDDocWasRepaired`	Document was repaired when it was opened.
`PDDocNewMajorVersion`	Document's major version is newer than current.
`PDDocNewMinorVersion`	Document's minor version is newer than current.
`PDDocOldVersion`	Document's version is older than current.
`PDDocSuppressErrors`	Don't display errors.

Related methods

`PDDoc.ClearFlags`

`PDDoc.SetFlags`

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

`nInsertPageAfter`	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.
`iPDDocSource`	The `LPDISPATCH` for the `AcroExch.PDDoc` containing the pages to insert. `iPDDocSource` contains the instance variable `m_lpDispatch`, which contains the `LPDISPATCH`.
`nStartPage`	The first page in `iPDDocSource` to be inserted into the current document.
`nNumPages`	The number of pages to be inserted.
`bBookmarks`	If a positive number, bookmarks are copied from the source document. If `0`, they are not.

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

`nMoveAfterThisPage`	The page being moved is placed after this page number. The first page in a `PDDoc` object is page `0`.
`nPageToMove`	Page number of the page to be moved.

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

`nStartPage`	The first page within the source file to be replaced. The first page in a `PDDoc` object is page `0`.
`iPDDocSource`	The `LPDISPATCH` for the `AcroExch.PDDoc` containing the new copies of pages that are replaced. `iPDDocSource` contains the instance variable `m_lpDispatch`, which contains the `LPDISPATCH`.
`nStartSourcePage`	The first page in `iPDDocSource` to use as a replacement page.

`nNumPages`	The number of pages to be replaced.
`bMergeTextAnnotations`	If a positive number, text annotations from `iPDDocSource` are copied. If `0`, they are not.

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

`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. 3. Call `PDDoc.Close`. 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

```
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.

Returns

Always returns `-1`.

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

`nPageMode`	The page 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

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.

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

Method	Description
DrawEx	Draws page contents into a specified window.
GetAnnot	Gets the specified annotation from the page's array of annotations.
GetAnnotIndex	Gets the index (within the page's annotation array) of the specified annotation.
GetDoc	Gets the AcroExch.PDDoc associated with the page.
GetNumAnnots	Gets the number of annotations on the page.
GetNumber	Gets the page number of the current page. The first page in a document is page zero.
GetRotate	Gets the rotation value, in degrees, for the current page.
GetSize	Gets a page's width and height in points.
RemoveAnnot	Removes the specified annotation from the page's annotation array.
SetRotate	Sets the rotation, in degrees, for the current page.

AddAnnot

Adds 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);
```

Parameters

`boundRect`	The `LPDISPATCH` for the `AcroExch.Rect` bounding rectangle in device space coordinates. `boundRect` contains the instance variable `m_lpDispatch`, which contains the `LPDISPATCH`.
`nXOrigin`	The x–coordinate of the portion of the page to be copied.
`nYOrigin`	The y–coordinate of the portion of the page to be copied.
`nZoom`	Zoom factor at which the page is copied, specified as a percent. For example, 100 corresponds to a magnification of 1.0.

Returns

`-1` if the page is successfully copied, `0` otherwise.

Related methods

`PDPage.DrawEx`

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

PDTextSelect.GetBoundingRect

PDTextSelect.GetNumText

PDTextSelect.GetPage

PDTextSelect.GetText

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.

`YOrigin`	The y–coordinate of the portion of the page to be drawn.
`zoom`	Zoom factor at which the page is to be drawn, specified as a percent. For example, 100 corresponds to a magnification of 1.0.

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

`window`	Handle for the window (`HWND`) into which the page is drawn.
`displayContext`	This parameter is invalid; do not use it. Assign it a `NULL` value. If it is not assigned `NULL`, an exception is thrown. **Note:** `displayContext` cannot be reliably used as the `hDC` for a printer device. In particular, Visual Basic applications cannot use `DrawEx` to print.

`updateRect`	`LPDISPATCH` for an `AcroExch.Rect` to be drawn with user space coordinates. `updateRect` contains the instance variable `m_lpDispatch`, which contains the `LPDISPATCH`.
	Any objects outside of `updateRect` are not drawn. All objects are drawn if `updateRect` is `NULL`.
	Use methods in the `CAcroRect` class to set the size of the rectangle. For example:
	`CAcroRect* rect = new CAcroRect;`
	
	`rect->CreateDispatch("AcroExch.Rect", &e);`
	`if (rect) {`
	`/* Set values for rect - increases from right to left and bottom to top */`
	`rect->SetLeft(100);`
	`rect->SetTop(400);`
	`rect->SetRight(400);`
	`rect->SetBottom(100);`
	`}`

`xOrigin`	The x–coordinate of the portion of the page to be drawn.
`yOrigin`	The y–coordinate of the portion of the page to be drawn.
`zoom`	Zoom factor at which the page is drawn, specified as a percent. For example, 100 corresponds to a magnification of 1.0.

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

PDPage.GetAnnotIndex

PDPage.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.PDAnnot whose index is obtained. iPDAnnot contains the instance variable m_lpDispatch, which contains the LPDISPATCH.

Returns

The annotation's index.

Related methods

PDPage.GetAnnot

PDPage.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

PDDoc.AcquirePage

PDDoc.GetNumPages

PDPage.GetNumber

PDPage.GetRotate

PDPage.GetSize

PDTextSelect.GetPage

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

```
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

```
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

PDPage.GetRotate

PDPage.GetSize

PDTextSelect.GetPage

GetRotate

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

Syntax

```
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

```
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

PDDoc.AcquirePage

PDPage.GetNumber

PDPage.GetRotate

PDTextSelect.GetPage

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.

Method	Description
`Destroy`	Destroys a text selection object.
`GetBoundingRect`	Gets a text selection's bounding rectangle.
`GetNumText`	Gets the number of text elements in a text selection.
`GetPage`	Gets the page number on which the text selection is located.
`GetText`	Gets the text from the specified element of a text selection.

Destroy

Destroys a text selection object.

Syntax

```
VARIANT_BOOL Destroy();
```

Returns

Always returns `-1`.

Related methods

`AVDoc.ClearSelection`

`AVDoc.SetTextSelection`

`AVDoc.ShowTextSelect`

`PDDoc.CreateTextSelect`

`PDPage.CreatePageHilite`

`PDPage.CreateWordHilite`

`PDTextSelect.GetBoundingRect`

`PDTextSelect.GetNumText`

PDTextSelect.GetPage

PDTextSelect.GetText

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

PDTextSelect.Destroy

PDTextSelect.GetBoundingRect

PDTextSelect.GetNumText

PDTextSelect.GetText

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.

Property	Description
X	Gets or sets the x-coordinate of an `AcroPoint`.
Y	Gets or sets the y-coordinate of an `AcroPoint`.

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
```

Returns

The y-coordinate of the `AcroPoint`.

AcroExch.Rect

Defines the location of an `AcroRect`.

The `Rect` object has the following properties.

Properties

Property	Description
Bottom	Gets or sets the bottom y-coordinate of an AcroRect.
Left	Gets or sets the left x-coordinate of an AcroRect.
Right	Gets or sets the right x-coordinate of an AcroRect.
Top	Gets or sets the top y-coordinate of an AcroRect.

Bottom

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

Syntax

```
[get/set] Short
```

Returns

The y-coordinate of the bottom of the AcroRect.

Left

Gets or sets left x-coordinate of an AcroRect.

Syntax

```
[get/set] Short
```

Returns

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

Right

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

Syntax

```
[get/set] 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.

Property	Description
`Date`	Gets or sets the date from an `AcroTime`.
`Hour`	Gets or sets the hour from an `AcroTime`.
`Millisecond`	Gets or sets the milliseconds from an `AcroTime`.
`Minute`	Gets or sets the minutes from an `AcroTime`.
`Month`	Gets or sets the month from an `AcroTime`.
`Second`	Gets or sets the seconds from an `AcroTime`.
`Year`	Gets or sets the year from an `AcroTime`.

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.

Method	Description
`GetVersions`	Deprecated
`GoBackwardStack`	Goes to the previous view on the view stack, if the previous view exists.
`GoForwardStack`	Goes to the next view on the view stack, if the next view exists.
`GotoFirstPage`	Goes to the first page in the document, maintaining the current location within the page and zoom level.
`GotoLastPage`	Goes to the last page in the document, maintaining the current location within the page and zoom level.
`GotoNextPage`	Goes to the next page in the document, if it exists. Maintains the current location within the page and zoom level.

Method	Description
GotoPreviousPage	Goes to the previous page in the document, if it exists. Maintains the current location within the page and zoom level.
LoadFile	Opens and displays the specified document within the browser.
Print	Prints the document according to the options selected in a user dialog box.
PrintAll	Prints the entire document without displaying a user dialog box.
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.
PrintPages	Prints the specified pages without displaying a user dialog box.
PrintPagesFit	Prints the specified pages without displaying a user dialog box.
PrintWithDialog	Prints the document according to the options selected in a user dialog box.
SetCurrentHighlight	Highlights the text selection within the specified bounding rectangle on the current page.
SetCurrentPage	Goes to the specified page in the document.
SetLayoutMode	Sets the layout mode for a page view according to the specified string.
SetNamedDest	Changes the page view to the named destination in the specified string.
SetPageMode	Sets the page mode according to the specified string.
SetShowScrollbars	Determines whether scrollbars will appear in the document view.
SetShowToolbar	Determines whether a toolbar will appear in the viewer.
SetView	Sets the view of a page according to the specified string.
SetViewRect	Sets the view rectangle according to the specified coordinates.
SetViewScroll	Sets the view of a page according to the specified string.
SetZoom	Sets the magnification according to the specified value.
SetZoomScroll	Sets the magnification according to the specified value, and scrolls the page view both horizontally and vertically according to the specified amounts.

Properties

The AxAcroPDF object has the following property.

Property	Description
Src	Gets or sets the URL for the document.

GetVersions

Note: Deprecated. This method is no longer available.

Syntax

```
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

```
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

```
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

```
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

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

Parameters

`fileName`	The path of the file to be opened.

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

```
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

```
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

```
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

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

Parameters

nFrom	The page number of the first page to be printed. The first page in a document is page 0.
nTo	The page number of the last page to be printed.

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

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

Parameters

nFrom	The page number of the first page to be printed. The first page in a document is page 0.
nTo	The page number of the last page to be printed.
bShrinkToFit	Specifies whether the pages will be shrunk, if necessary, to fit into the imageable area of a page in the printer.

Related methods

AcroPDF.Print

AcroPDF.PrintAll

AcroPDF.PrintAllFit

AcroPDF.PrintPages

AcroPDF.PrintWithDialog

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

```
void printWithDialog();
```

Related methods

AcroPDF.Print

AcroPDF.PrintAll

AcroPDF.PrintAllFit

AcroPDF.PrintPages

AcroPDF.PrintPagesFit

SetCurrentHighlight

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

Syntax

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

Parameters

nLeft	The distance in points from the left side of the page.
nTop	The distance in points from the top of the page.

nRight	The width of the bounding rectangle.
nBottom	The height of the bounding rectangle.

SetCurrentPage

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

Syntax

```
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

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

Parameters

szLayoutMode	Possible values: DontCare — use the current user preference SinglePage — use single page mode (as it would have appeared in pre-Acrobat DC 3.0 viewers) OneColumn — use one-column continuous mode TwoColumnLeft — use two-column continuous mode with the first page on the left TwoColumnRight — use two-column continuous mode with the first page on the right

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

szPageMode	Possible values: none — displays the document, but does not display bookmarks or thumbnails (default) bookmarks — displays the document and bookmarks thumbs — displays the document and thumbnails

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

`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. `FitB` — Fits the entire height of the bounding box within the window.

Related methods

`AcroPDF.SetLayoutMode`

`AcroPDF.SetNamedDest`

`AcroPDF.SetViewRect`

`AcroPDF.SetViewScroll`

SetViewRect

Sets the view rectangle according to the specified coordinates.

Syntax

```
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

```
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

```
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

`percent`	The desired zoom factor, expressed as a percentage. For example, 1.0 represents a magnification of 100%.
`left`	The horizontal coordinate positioned at the left edge.
`top`	The vertical coordinate positioned at the top edge.

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.

6 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

- DocPageRight
- DocPageUp
- DocScrollTo
- DocZoomTo

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

DocOpen

FileOpen

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

`fullPath`	The full path of the file to be closed.

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

`fullPath`	The full path of the document.

`fromPage`	The page number of the first page to be deleted.
`toPage`	The page number of the last page to be deleted.

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

`fullPath`	The full path of the file to be searched.
`string`	The string to be found.
`caseSensitive`	`true` if the search is case-sensitive, `false` otherwise.
`wholeWords`	`true` if the search will only match whole words, `false` otherwise.
`bReset`	`true` if the search begins on the first page of the document, `false` if the search begins on the current page.

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

`fullPath`	The full path of the file.
`pageNum`	The page number of the destination page.

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

`fullPath`	The full path of the file.
`nameDest`	The named destination.

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`

FileOpen

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

DocPageUp

DocPageUp

DocPageRight

Scrolls to the right by a small amount.

Syntax

```
[DocPageRight(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

DocPageLeft

DocPageUp

DocPageUp

DocPageUp

Scrolls backward through the document by one screen area.

Syntax

```
[DocPageUp(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

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

`fullPath`	The full path of document.
`startPage`	The page number of the first page to be printed.
`endPage`	The page number of the last page to be printed.

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

`fullPath`	The full path of the target document. This file must already be open in the Acrobat DC application.
`startDestPage`	The page number of the first page in the target document to be replaced.
`sourcePath`	The full path of the source document. This file does not have to be already open in the Acrobat DC application.
`startSourcePage`	The page number of the first page in the source document to use as a replacement page.
`endSourcePage`	The page number of the last page in the source document to use as a replacement page.

Returns

`true` if the pages are replaced successfully. Returns `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

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

Parameters

`fullPath`	The full path of the file to be saved.

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

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

Parameters

`fullPath`	The full path of the existing file.
`newPath`	The full path of the new file.

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

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

Parameters

`fullPath`	The full path of the document.
`x`	The destination's x–coordinate.
`y`	The destination's y–coordinate.

Returns

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

Related methods

`DocPageDown`

`DocPageLeft`

`DocPageRight`

DocPageUp

DocSetViewMode

Determines whether bookmarks, thumbnail images, or neither are shown in addition to the document.

Syntax

```
[DocSetViewMode(char* fullPath, char* viewType)]
```

Parameters

`fullPath`	The full path of the document.
`viewType`	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.

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

`fullPath`	The full path of the file whose zoom to set.

`zoomType`	The zoom strategy to use. Must be one of the following: `AVZoomNoVary` — A fixed zoom, such as 100%. `AVZoomFitPage` — Fits the page in the window. `AVZoomFitWidth` — Fits the page's width into the window. `AVZoomFitVisibleWidth` — Fits the page's visible content into the window.
`scale`	The magnification specified as a percent (for example, 100 corresponds to a magnification of 1.0). `scale` is used only when `zoomType` is `AVZoomNoVary`.

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

`true` 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

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

Parameters

`fullPath`	The full path of the file to be opened.

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

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

Parameters

`fullPath`	The full path of the file to be printed.

Returns

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

Related methods

DocPrint

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

`true` 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

`fullPath`	The full path of the file to be printed.

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

`fullPath`	The full path of the file to be printed.

Returns

`true` is always returned.

Related methods

`DocPrint`

`FileOpenEx`

`FilePrintEx`

`FilePrintSilent`

`FilePrintTo`

`FilePrintToEx`

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

`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

`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

```
[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

```
[FullMenus()]
```

Returns

`true` if full menus are set successfully, `false` otherwise.

Related methods

`DocSetViewMode`

`ShortMenus`

HideToolbar

Hides the toolbar.

Syntax

```
[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

```
[MenuitemExecute(char* menuItemName)]
```

Parameters

`menuItemName`	The language-independent name of the menu item to execute. See the Acrobat and PDF Library API Reference 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`

7 Apple Event Objects and Apple Events

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

Property	Class	Description
best type	type class [r/o]	The best descriptor type.
bounds	a list of small real	The boundary rectangle for the annotation in PDF space (left, top, right, bottom).
class	type class [r/o]	The class.

Property	Class	Description
`color`	'RGB'	The color of the border around the annotation.
`contents`	international text	Text annotations only. The textual contents of the note.
`default type`	type class [r/o]	The default descriptor type.
`destination page number`	integer	Link annotations only. The page number to appear in the PDF window when the annotation link is activated.
`destination rectangle`	a list of small real	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.
`fit type`	constant	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*.
`index`	integer [r/o]	The annotation's index within the `page` object.
`modification date`	date	The date and time the annotation was last modified.
`name`	string	Text annotations only. The annotation's label.
`open state`	Boolean	Text annotations only. Whether the annotation is open.
`subtype`	international text [r/o]	The subtype of the annotation.
`zoom factor`	small real	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`.

Related methods

`delete`

`perform`

application

The Acrobat DC or Acrobat DC Reader application itself.

Elements

Element	Accessed by
document	name, numeric index
PDF Window	name, numeric index
menu	name, numeric index
menu item	name

Properties

Property	Class	Description
active doc	reference	The active document.
active tool	international text	The type of the currently active tool. See the *Acrobat and PDF Library API Reference* for a list of tool names.
anti_alias text	Boolean	Determines whether to anti-alias text and monochrome images.
best type	type class [r/o]	The best descriptor type.
case sensitivity	Boolean	Determines whether searches are case-sensitive.
class	type class [r/o]	The class.
default type	type class [r/o]	The default descriptor type.
default zoom factor	small real	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%).
default zoom type	constant	The default zoom type when opening a new document. Valid values are `no vary`, `fit page`, `fit width`, `fit height`, and `fit visible width`.
download entire file	Boolean	Determines whether to download the entire file.
frontmost	Boolean	Determines whether Acrobat DC is the frontmost application. Value can be set to true only.
fullscreen click advances	Boolean	Determines whether mouse click advances in fullscreen mode.

Property	Class	Description
`fullscreen cursor`	Boolean	Determines whether to hide the cursor in fullscreen mode.
`fullscreen escape`	Boolean	Determines whether the Esc key can be used to exit fullscreen mode.
`fullscreen loop`	Boolean [r/o]	Determines whether the document's pages are displayed in a loop while in fullscreen mode.
`fullscreen timer delay`	integer	The number of seconds to advance to the next page in fullscreen mode.
`fullscreen transition`	international text [r/o]	Default fullscreen transition.
`highlight color`	'RGB '	Color used to highlight selections.
`maximum documents`	integer [r/o]	Maximum number of open documents.
`name`	string [r/o]	The application's name.
`note color`	'RGB '	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}`.
`note font name`	international text	Deprecated.
`note font size`	integer	Deprecated.
`open in place`	Boolean	Determines whether to open cross-document links in the same window.
`page layout`	international text	Default page layout. Values are: `Single Page`, `Continuous`, `Facing`, and `Continuous - Facing`.
`page units`	international text	Default page display units: `Points`, `Inches` or `Millimeters`.
`PS level`	integer	Deprecated. Set the PostScript level when using `save` or `print pages` commands.
`save as linearize`	Boolean	Determines whether to save the document as optimized for the web.
`show splash at startup`	Boolean	Determines whether the splash screen is shown at startup.
`skip warnings`	Boolean	Determines whether to skip warning dialog boxes during program execution.

Property	Class	Description
`shrink to fit`	Boolean	Deprecated.
`text note label`	international text	The text that will appear in the title bar of all newly created text notes.
`toolbar visibility`	Boolean	Determines whether the toolbar is visible.
`UI language`	international text [r/o]	A three-character language code identifying which language is used in the Acrobat DC user interface. Example: `ENU` represents English.
`use fullscreen timer`	Boolean	Determines whether to use a timer to advance pages in fullscreen mode
`version`	string [r/o]	The version number of the application.
`whole word searching`	Boolean	Determines whether searches are applied to whole words only.

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 `PDBookmark` object.

Note: This object was formerly known as `PDBookmark`.

Plural form

Bookmarks

Properties

Property	Class	Description
`best type`	type class [r/o]	The best descriptor type.
`class`	type class [r/o]	The class.
`default type`	type class [r/o]	The default descriptor type.
`destination page number`	integer	The page number to which the `PDF Window` goes when the bookmark's action is performed.
`destination rectangle`	list of small real	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). **Note:** Set this only after setting `fit type`.
`fit type`	constant	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.
`index`	integer [r/o]	The bookmark's index within the `document`.
`name`	international text	The bookmark's title.
`zoom factor`	small real	The zoom factor used when `fit type` is `Left Top Zoom`; ignored otherwise. Setting this property automatically sets `fit type` to `Left Top Zoom`.

Related methods

insert pages

perform

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

Property	Class	Description
best type	type class [r/o]	The best descriptor type.
class	type class [r/o]	The class.
default type	type class [r/o]	The default descriptor type.
index	integer [r/o]	The index number of the converter.
name	international text	The conversion's description.

Related methods

save

document

Represents a single open document in Acrobat DC or Acrobat DC Reader.

Elements

Element	Accessed by
page	Numeric index. The first page in a document is page 1.
bookmark	Name or numeric index.
PDF Window	An index of 1 or with the some keyword in AppleScript. No document has more than one PDF Window.

Plural form

documents

Properties

Property	Class	Description
`best type`	type class [r/o]	The best descriptor type.
`bounds`	bounding rectangle [r/o]	The boundary rectangle for the document's window, in screen coordinates (left, top, right, bottom).
`class`	type class [r/o]	The class.
`default type`	type class [r/o]	The default descriptor type.
`file alias`	alias [r/o]	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.
`modified`	Boolean [r/o]	Determines whether the document has been modified and should be saved.
`name`	international text [r/o]	The document's name as it appears in the window's titlebar.
`view mode`	constant	The viewing mode of the document. Possible values: `just pages`, `pages and thumbs`, or `pages and bookmarks`.

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

Element	Accessed by
menu item	name, numeric index.

Properties

Property	Class	Description
`best type`	type class [r/o]	The best descriptor type.
`class`	type class [r/o]	The class.
`default type`	type class [r/o]	The default descriptor type.
`name`	international text [r/o]	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.
`title`	string [r/o]	The menu's title as it would appear in the user interface.

Related methods

`execute`

menu item

A menu item contained within a menu in Acrobat DC or Acrobat DC Reader.

Properties

Property	Class	Description
`best type`	type class [r/o]	The best descriptor type.
`class`	type class [r/o]	The class.
`default type`	type class [r/o]	The default descriptor type.
`enabled`	Boolean [r/o]	Determines whether the menu item is enabled.
`has submenu`	Boolean [r/o]	Determines whether the menu item has a hierarchical sub-menu.
`marked`	Boolean [r/o]	Determines whether the menu item is checked.
`name`	international text [r/o]	The menu item's language-independent name. See the *Acrobat and PDF Library API Reference* for a list of menu item names.
`title`	string [r/o]	The menu's title as it would appear in the user interface.

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

Element	Accessed by
annotation	numeric index.

Plural form

Pages

Properties

Property	Class	Description
best type	type class [r/o]	The best descriptor type.
bounds	list of small real	The boundary rectangle for the page in user space (left, top, right, bottom).
class	type class [r/o]	The class.
default type	type class [r/o]	The default descriptor type.
page number	integer [r/o]	The page's number. The first page in a document is page 1.
rotation	integer	The rotation angle of the page in degrees (0, 90, 180, or 270).

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

Element	Accessed by
page	numeric index. The first page in a document is page 1.

Properties

Property	Class	Description
best type	type class [r/o]	The best descriptor type.
bounds	bounding rectangle	The boundary rectangle for the window.
class	type class [r/o]	The class.
default type	type class [r/o]	The default descriptor type.
document	document [r/o]	The document that owns this window.
index	integer	The number of the window.

Property	Class	Description
`name`	international text [r/o]	The document's name as shown in the window's titlebar.
`page number`	integer	The number of the currently displayed page.
`position`	point [r/o]	The upper left coordinates of the window.
`visible`	Boolean [r/o]	Whether the window is visible.
`zoomed`	Boolean	Whether the window is zoomed.
`zoom factor`	small real	The current zoom factor specified as a percentage. For example, a value of 100 corresponds to a zoom factor of 1.0 (100%).
`zoom type`	constant	The zooming and content fitting algorithm currently employed. Possible values: `no vary`, `fit page`, `fit width`, `fit height`, and `fit visible width`.

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`.

Property	Class	Description
annotations	Boolean [r/o]	Determines whether to include annotations.
binary	Boolean [r/o]	Determines whether the output file should be in binary or ASCII text format.
embedded fonts	Boolean [r/o]	Determines whether to include fonts.
halftones	Boolean [r/o]	Determines whether to use halftone screens.
images	Boolean [r/o]	Determines whether to include RGB and LAB images.
postScript level	integer [r/o]	The PostScript Language level. Only levels 2 and 3 are supported.
preview	Boolean [r/o]	Determines whether to include preview in output.
TrueType	Boolean [r/o]	Determines whether to convert TrueType fonts to Type 1.

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

close	The document to close.
saving	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.
linearize	Determines whether the document should be optimized for the web when saving before closing.

Related events

open

count

Counts the number of instances of a particular class.

Syntax

```
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:

```
count [reference] each [type class]
```

Returns

An integer specifying the number of elements.

AppleScript example

```
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

```
delete [reference]
```

Parameters

`delete`	The object to be deleted.

Related events

`make`

`exists`

AppleScript example

```
delete first bookmark of document "test.pdf"
```

exists

Tests whether a specified object exists.

Syntax

```
[reference] exists
exists [reference]
```

Parameters

`exists`	Object whose existence is checked.

Returns

`true` if the object exists, `false` otherwise.

AppleScript example

```
exists second document
second document exists
```

get

Retrieves the value of an object or property.

Syntax

```
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

```
get the name of last bookmark
get the index of last bookmark as string
```

make

Creates a new object.

Syntax

```
make new [type class] at [location reference] with data [anything] with
properties [record]
```

Parameters

`make [new]`	The class of the new object.
`at`	The location at which to insert the new object.
`with data`	The initial data for the new object.
`with properties`	The initial values for the properties of the new object.

Returns

A reference to the newly created object.

Related events

`delete`

`exists`

AppleScript example

```
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

```
move [reference] to [location reference]
```

Parameters

`move`	The page object to move. The first page in a document is page 1.
`to`	The new location for the page.

Returns

A reference to the page that is moved.

AppleScript example

```
move page 3 to before page 1
```

open

Opens a document or documents.

Syntax

```
open [list of alias] invisible [boolean] options [string]
```

Parameters

`open`	The document or documents to open.
`invisible`	Whether the opened document should be hidden. Default is `false`.
`options`	Optional parameter string of open actions.

Related events

`close`

quit

Causes the Acrobat DC application to quit.

Syntax

```
quit saving [constant]
```

Parameters

`saving`	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`.

AppleScript example

```
quit saving yes
```

save

Saves a document.

Syntax

```
save [reference] to [file specification] using [reference] linearize[ boolean]
```

Parameters

save	The document to be saved.
to	The file into which the document is to be saved. This parameter is optional in Acrobat DC 6.0 and higher. Specifying the to parameter is equivalent to doing a Save As. You can save a document in one of the supported formats with the using parameter.
linearize	Determines whether the document should be optimized for the web.
using	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.

AppleScript example

```
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

```
set [reference] to [anything]
```

Parameters

set	The object or property whose value is set.
to	The new value.

Related events

get

AppleScript example

```
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`*`]`

Parameters

`bring to front`	The document to be displayed as the active document in the front window.

AppleScript example

```
bring to front document "AppleEvt.pdf"
```

Apple event ID

```
kAEBringToFront ('bfrt')
```

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]
```

Parameters

`saving`	Determines whether to save modified documents before closing. Possible values: `yes` — Save the document. `no` — Do not save the document. `ask` — If the document has been modified, ask the user whether to save it. The default value is `ask`.

Related events

`open` (Required suite)

`open` (Core suite)

AppleScript example

```
close all docs
```

Apple event ID

```
kAECloseAllDocs ('cldc')
```

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

```
delete pages [reference] first [integer] last [integer]
```

Parameters

`delete pages`	The document containing the pages to be deleted.
`first`	The first page to be deleted. The first page in a document is page 1.
`last`	The last page to be deleted.

Related events

`insert pages`

`replace pages`

AppleScript example

```
delete pages document "AppleEvt.pdf" first 1 last 3
```

Apple event ID

```
kAEDeletePages ('dlpg')
```

Apple event parameters

```
keyAEFirstPage ('frpg')
keyAELastPage ('lapg'')
```

delete thumbs

Deletes all thumbnails from the document.

Syntax

```
delete thumbs [reference]
```

Parameters

`delete thumbs`	The document from which thumbnails are deleted.

Related events

`create thumbs`

AppleScript example

```
delete thumbs document "AppleEvt.pdf"
```

Apple event ID

```
kAEDeleteThumbs ('dltb')
```

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.

Related events

find text

AppleScript example

```
find next note document "dev_acro.pdf"
```

Apple event ID

```
kAEFindNextNote ('fnnt')
```

Apple event parameters

```
keyAEWrapAround ('wrar')
```

find text

Finds text in a document.

Syntax

```
find text [reference] string [international text] case sensitive [boolean]
whole words [boolean] wrap around [boolean]
```

Parameters

find text	The document to be searched.
string	The string to be found.
case sensitive	Determines whether searching is case-sensitive. The default value is false.
whole words	Determines whether to search only for whole words. The default value is false.
wrap around	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.

Related events

find next note

AppleScript example

```
find text document "PLUGINS.PDF" string "Develop" whole words true
```

Apple event ID

```
kAEFindText ('ftxt')
```

Apple event parameters

```
keyAESearchString ('sstr')
keyAECaseSensitive ('case')
keyAEWholeWordsOnly ('whwd')
keyAEWrapAround ('wrar')
```

get info

Gets the value of the specified key in the document's `Info` dictionary.

Syntax

```
get info [reference] key [international text]
```

Parameters

`get info`	The document from which to obtain the `Info` dictionary entry.
`key`	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.

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

`go backward`	A `PDF Window` object

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

goto	The PDF Window object in which to change the page.
page	The page number of the page to be displayed. The first page in a document is page 1.

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

goto next	The PDF Window object in which to change the page.

Related events

go backward

go forward

goto

goto previous

AppleScript example

```
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

```
goto previous [reference]
```

Parameters

goto previous	The PDF Window object in which to change the page.

Related events

go backward

go forward

goto

goto next

AppleScript example

```
goto previous first PDF Window
```

Apple event ID

```
kAEGotoPrevPage ('pvpg')
```

insert pages

Inserts one or more pages from one document into another.

Syntax

```
insert pages [reference] after [integer] from [reference] starting with
[integer] number of pages [integer] insert bookmarks [boolean]
```

Parameters

`insert pages`	The target document in which to insert the page or pages.
`after`	The number of the page after which the pages will be inserted. The first page in a document is page 1.
`from`	The source document containing the page or pages to be inserted.
`starting with`	The first page to be inserted.
`number of pages`	The number of pages to be inserted.
`insert bookmarks`	Determines whether to copy bookmarks that point to the inserted pages. Default is `true`.

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

`named`	Button name. See the *PDF Library documentation* for a list of toolbar button names.

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

`maximize`	The document whose window is to be resized.
`max size`	If `true`, the document's window is set to full size. If `false`, the window is returned to its original size.

AppleScript example

```
maximize document "AppleEvt.pdf" max size false
```

Apple event ID

```
kAEMaximize ('maxi')
```

Apple event parameters

```
keyAEMaxSize ('mxsz')
```

perform

Executes a bookmark's or link annotation's action.

Syntax

```
perform [reference]
```

Parameters

object	The bookmark or page object whose action is to be performed.

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

```
print pages [reference] first [integer] last [integer] PS Level [integer]
binary output [boolean] shrink to fit [boolean]
```

Parameters

print pages	The document containing the page or pages to be printed. This keyword and the actual filename must be specified.
first	The first page to be printed. The default value is 1.
last	The last page to print. The default value is the number of the last page in the document.
PS Level	The PostScript language level (1 or 2) to use when printing to a PostScript printer. The default value is 1.
binary output	Determines whether binary output is permitted (used for PostScript printing only). The default value is false.
shrink to fit	Determines whether pages should be shrunk to fit paper in printer. The default value is false.

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

```
replace pages [reference] over [integer] from [reference] starting with
[integer] number of pages [integer] merge notes [boolean]
```

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.

`from`	The source document from which the replacement page or pages are obtained.
`starting with`	The first page in the source document to be copied.
`number of pages`	The number of pages to be replaced.
`merge notes`	Determines whether to copy notes from the source document. The default value is `true`.

Related events

`delete pages`

`insert pages`

AppleScript example

```
replace pages document "AppleEvt.pdf" over 2 from document "dev_acro.pdf"
starting with 1 number of pages 4 merge notes false
```

Apple event ID

```
kAEReplacePages ('rppg')
```

Apple event parameters

```
keyAEDestStartPage ('dtpg')
keyAESourceDoc ('srdc')
keyAESourceStartPage ('stpg')
keyAENumPages ('nmpg')
keyAEMergeNotes ('mgnt')
```

scroll

Scrolls the view of a page by the specified amount.

Syntax

`scroll [`*reference*`] X Amount [`*integer*`] Y Amount [`*integer*`]`

Parameters

`scroll`	The `PDF Window` object in which to scroll the view.
`X Amount`	The amount to scroll in the horizontal direction, in pixels. Positive values move the view to the right.
`Y Amount`	The amount to scroll in the vertical direction, in pixels. Positive values move the view down.

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')
keyAEYDelta ('ydlt')
```

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

select text	The PDF Window object in which to select text.
from words	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).
from chars	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).

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

`zoom`	The `PDF Window` object to be zoomed.
`to`	The zoom factor specified as a percentage. For example, a value of 100 (100%) displays the document with a magnification of 1.0.

AppleScript example

```
zoom first PDFWindow to 150
```

Apple event ID

```
kAEZoomTo ('zmto')
```

Apple event parameters

```
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

```
do script [international text] file [alias]
```

Parameters

`do script`	The JavaScript script to be executed.
`file`	File holding the JavaScript script to be executed.

Returns

Result of JavaScript execution as text.

AppleScript example

```
do script MyJavaScriptFile.js
```

8 Acrobat DC Catalog Plug-In

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

```
[AppExit()]
```

Returns

If `true`, Catalog exited successfully, otherwise `false`.

AppFront

Brings Catalog to the front.

Syntax

```
[AppExit()]
```

FileBuild

Builds an index based on the specified index definition file.

Syntax

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

Parameters

`fullPath`	The full path of the file to be opened, including the .pdx extension.

Returns

If `true`, the file opened successfully, otherwise `false`.

FileOpen

Opens an index definition file and displays the Edit Index Definition dialog box.

Syntax

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

Parameters

`fullPath`	The full path of the file to be opened, including the .pdx extension.

Returns

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

FilePurge

Purges an index definition file.

Syntax

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

Parameters

`fullPath`	The full path of the file to be purged, including the .pdx extension.

Returns

`true` if the file was successfully purged, otherwise `false`.

9 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:

   ```
   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:

   ```
   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.

Exception name	Numeric value	Description
`AutErcNoDoc`	1	No document is currently open in the Acrobat DC application.
`AutErcNotTerminal`	2	This property or method applies to terminal fields or their annotations.
`AutErcNotToThisFieldType`	3	This property or method is not applicable to this type of field.

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

```
void PopulateListOrComboBox ( const VARIANT& arrItems,
                  const VARIANT& arrExportVal);
```

Parameters

`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

```
void SetBackgroundColor (LPCTSTR bstrColorSpace, float GorRorC, float GorM,
float BorY, float K);
```

Parameters

`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.

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

SetBorderColor

SetForegroundColor

Example

```
Field.SetBackgroundColor "RGB", 0.7, 0.3, 0.6, 0
```

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

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.
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.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

`bstrFace`	A string that specifies the face for which the caption will be used. Valid strings include: `N` — Normal appearance `D` — Down appearance `R` — Appearance for rollover
`bstrCaption`	The caption for the button. 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

```
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

```
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

```
void SetForegroundColor (LPCTSTR bstrColorSpace, float GorRorC, float GorM,
float BorY, float K);
```

Parameters

`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.
`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

SetBorderColor

Example

```
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

```
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.

`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:

`AFDate_KeystrokeEx(`*`cFormat`*`)` `AFDate_Format(`*`cFormat`*`)`	*`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"
`AFTime_Keystroke(`*`ptf`*`)` `AFTime_Format(`*`ptf`*`)`	*`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]

`AFPercent_Keystroke(nDec, sepStyle)` `AFPercent_Format(nDec, sepStyle)`	*nDec* is the number of places after the decimal point. *sepStyle* is an integer denoting whether to use a separator. If *sepStyle* is `0`, use commas. If *sepStyle* is `1`, do not separate.
`AFSpecial_Keystroke(psf)` `AFSpecial_Format(psf)`	*psf* is the type of formatting to use: 0 = zip code 1 = zip + 4 2 = phone 3 = SSN
`AFNumber_Format(nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend)` `AFNumber_Keystroke(nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend)`	*nDec* is the number of places after the decimal point. *sepStyle* is an integer denoting whether to use a separator. If *sepStyle* is `0`, use commas. If *sepStyle* is `1`, do not separate. *sepStyle* is the formatting used for negative numbers: 0 = MinusBlack 1 = Red 2 = ParensBlack 3 = ParensRed *currStyle* is the currency style - not used. *strCurrency* is the currency symbol. *bCurrencyPrepend* is `true` to prepend the currency symbol; `false` to display on the end of the number.

SetResetFormAction

Sets the action of the field to be of type `ResetForm`.

Syntax

```
void SetResetFormAction (LPCTSTR bstrTrigger, 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
`theFlags`	When `0` (`Include`), `arrFields` specifies which fields to include in the reset operation. When non-zero (`Exclude`), `arrFields` specifies which fields to exclude from the reset operation.

`arrFields`	Optional. An array of strings for the fully-qualified names of the fields. Depending on the value of `theFlags`, these fields are included in or excluded from the reset operation. When the fields are included, the set can include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the action. When not supplied, all fields are reset.

SetSubmitFormAction

Sets the action of the field to be of type `SubmitForm`.

Syntax

```
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`

- DefaultValue
- Editable
- Highlight
- IsHidden
- IsMultiline
- IsPassword
- IsReadOnly
- IsRequired
- IsTerminal
- Name
- NoViewFlag
- PrintFlag
- Style
- TextFont
- TextSize
- Type
- Value

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 `AutErcNotToThisFieldType` 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 `AutErcNotTerminal` 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 `AutErcNotToThisFieldType` 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

group should have a different export value; see `SetExportValues`). For fields of type listbox or combobox, if an export value is defined, then that represents the value, otherwise the item name is used.

These remarks apply also to `DefaultValue`.

Syntax

```
[get/set] String
```

Example

```
Dim arrExp(1) As String
arrExp(0) = "CreditCardV"
arrExp(1) = "CreditCardM"
Field.SetExportValues arrExp
Field.Value = arrExp(0)
```

Fields

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`
- `ExportAsFDF`
- `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

`bstrFieldName`	The fully-qualified name of the field.
`bstrFieldType`	Field type for the newly created field. Valid types are: • `text` • `button` • `combobox` • `listbox` • `checkbox` • `radio button` • `signature` 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.
`pageNum`	The page number (zero-based).
`left, top, right, bottom`	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.

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. See the *Acrobat SDK JavaScript API Reference* for information on event level values.

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:

```
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

```
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

```
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

```
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

```
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

```
void Remove (LPCTSTR bstrFieldName);
```

Parameters

`bstrFieldName`	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.

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*
```

10 Acrobat DC Search Plug-in

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;
```

```
    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

`qlt`	The query language type. Must be one of the values shown in "Query language type constants" on page 240.
`bOverrideWordOptions`	Indicates that the client wishes to use different word options than those currently set by the user.
`nWordOptions`	The word options. Must be an `OR` of the values shown in "Word option bit-flag constants" on page 241.
`nMaxDocs`	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.
`nSortOffsets`	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.
`nQueryOffset`	An offset into the `cData` chunk that points to a `NULL`-terminated string containing the query to execute.
`nNumSorts`	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.
`bSortWays`	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.

Query language type constants

`QLangType_Simple`	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.
`QLangType_CQL`	Allows Boolean searches using `AND`, `OR`, and `NOT`, as described in the Acrobat DC Search plug-in's online help file.
`QLangType_Passthrough`	The Verity BooleanPlus query language. Contact Verity for further information on this language.

Word option bit-flag constants

QPON_Case	The search is case-sensitive.
QPON_Stemming	Find not only the specified word, but other words that have the same stem. For example, run and ran have the same stem.
QPON_SoundsLike	Find not only the specified word, but other words that sound like it.
QPON_Thesaurus	Find not only the specified word, but other words that have the same meaning.
QPON_Proximity	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.
QPON_Refine	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.

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:

```
nSize = sizeof(QueryData) + sls + lq;
qd = (QueryData *)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:

```
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" .

```
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

`eAction`	The operation to be performed on the index. Must be one of values listed in "Index operation selectors" on page 242.
`nIndexOffset`	An offset into the `cData` chunk that points to a `NULL`-terminated string containing the PDX file representing the index.
`nTempNameOffset`	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.

Index operation selectors

`IndexAction_Add`	Adds an index to the shelf.
`IndexAction_Remove`	Removes an index from the shelf.
`IndexAction_Enable`	Enables an index on the shelf.
`IndexAction_Disable`	Disables an index on the shelf.

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

`kIndexListTag ('SilP'), typeLongInteger`	An opaque `void*` representing the shelf, obtained from `SearchGetIndexList`.
`kPathTag ('Path'), typeChar`	Mac OS full path representing an index, of the form: `MyDisk:TopFolder:BottomFolder:Strange.pdx`
`kFlagTag ('Flag'), typeLongInteger`	Index flags. See `SearchGetIndexFlags` on page 245 for a description. The `kIndexAvailable` flag should always be set.

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

`kIndexListTag ('SilP'), typeLongInteger`	An opaque `void*` representing the shelf, obtained from `SearchGetIndexList`.

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

`kQueryStringTag ('Quryv'), typeChar`	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`.
`kParserTag ('Prsr'), typeShortInteger`	The query parser to use; may be one of (see SrchType.h): `kParserSimple 0` — Allows only simple phrase searches; does not allow Boolean searching. `kParserCQL 1` — Allows Boolean searches using `AND`, `OR`, and `NOT`, as described in the Acrobat DC Search plug-in's online help file. `kParserBPlus 2` — The Verity BooleanPlus query language. Contact Verity for further information on this language.
`kSortSpecTag ('Sort'), typeAEList`	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.
`kWordOptionsTag ('WOpt'), typeLongInteger`	A bit field of word options. Must be a logical OR of the values listed below in "Word options for Apple events" on page 244. The manner in which the options are used depends on the value associated with `kOptionsOverrideTag`.
`kOptionsOverrideTag ('WOer'), typeShortInteger`	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.
`kMaxDocsTag ('MaxD'), typeShortInteger`	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.

Word options for Apple events

`kWordOptionCase`	The search is case-sensitive.
`kWordOptionStemming`	Find not only the specified word, but other words that have the same stem (for example, run and ran have the same stem).
`kWordOptionSoundsLike`	Find not only the specified word, but other words that sound like it.
`kWordOptionThesaurus`	Find not only the specified word, but other words that have the same meaning.

`kWordOptionProximity`	Consider the proximity of results when using the `AND` operator to look for more than one word in a document. Without `kWordOptionProximity`, `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 `kWordOptionProximity`, however, `AND` terms must be within two or three pages of each other to be found. Also, with `kWordOptionProximity`, the closer `AND` terms appear together, the higher the relevance ranking of the document that contains them.
`kWordOptionRefine`	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.

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

`kIndexListTag ('SilP'), typeLongInteger`	An opaque `void*` representing the shelf, obtained from `SearchGetIndexList`.
`kPathTag ('Path'), typeChar`	Mac OS full path representing an index, of the form: `MyDisk:TopFolder:BottomFolder:Strange.pdx`

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')`

Parameters

`kIndexTag ('SixP'), typeLongInteger`	An opaque `void*` representing an index.

Returns

`kFlagTag ('Flag'), typeLongInteger`

A logical `OR` of the following:

`kIndexAvailableFlag (1L << 0)` — Set if the index is available for searching.

`kIndexSelectedFlag (1L << 1)` — Set if the index appears with a check mark in the Search plug-in's user interface.

`kIndexPtrInvalidFlag (1L << 31)` — Set if the index is not valid or is no longer valid.

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 `SearchGetIndexByPath`, `SearchGetNthIndex`, or `SearchAddIndex`.

Returns

`kPathTag ('Path'), typeChar`

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 nth 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 nth index is gone.

SearchRemoveIndex

Removes the specified index from the shelf.

Apple event ID

`kSearchRemoveIndex ('rmdx')`

Parameters

`kIndexListTag ('SilP'), typeLongInteger`	An opaque `void*` representing the shelf, obtained from `SearchGetIndexList`.
`kIndexTag ('SixP'), typeLongInteger`	An opaque `void*` representing the index to be removed. The index may be obtained using `SearchGetIndexByPath`, `SearchGetNthIndex`, or `SearchAddIndex`.

SearchSetIndexFlags

Sets the flags for an index.

Apple event ID

`kSearchSetIndexFlags ('sfdx')`

Parameters

`kIndexTag ('SixP'), typeLongInteger`	An opaque `void*` representing an index.
`kFlagTag ('Flag'), typeLongInteger`	Index flags. See the description in `SearchGetIndexFlags`. In practice, `kIndexAvailableFlag` should always be set.

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.

Menu name	Description
`AcroSrch:ToolsSubMenu`	Acrobat DC Search submenu of Edit menu

Menu item names

The Search plug-in adds the following menu items to Acrobat DC.

Menu item name	Description
`AcroSrch:Query`	Displays the Search dialog box.
`AcroSrch:Indexes`	Displays the Index dialog box.
`AcroSrch:Results`	Displays the Results dialog box.
`AcroSrch:Assist`	Displays the Word Assistant dialog box.
`AcroSrch:Separator`	A separator item in the Search tools menu.
`AcroSrch:PrevDoc`	Goes to the previous document in the hit list.
`AcroSrch:PrevHit`	Goes to the previous hit in the hit list.
`AcroSrch:NextHit`	Goes to the next hit in the hit list.
`AcroSrch:NextDoc`	Goes to the next document in the hit list.

Toolbar button names

The Search plug-in adds the following buttons to the Acrobat DC toolbar.

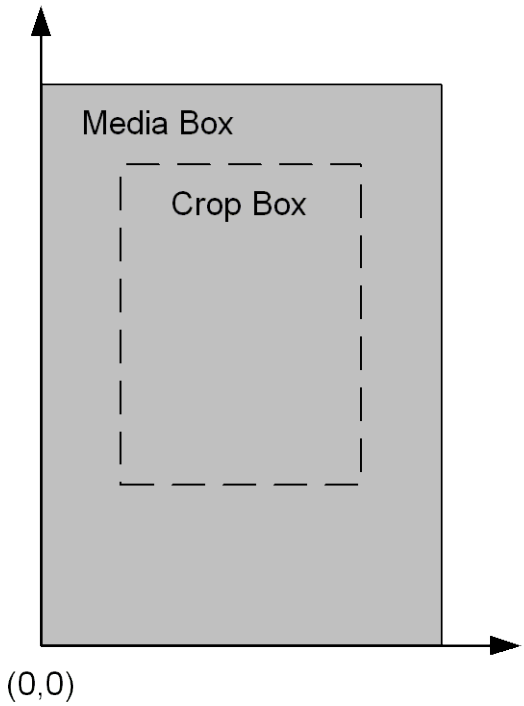
Button name	Description
`AcroSrch:Separator`	Separator (not visible).
`AcroSrch:Query`	Displays the Acrobat DC Search plug-in's query dialog box.
`AcroSrch:Results`	Displays the Acrobat DC Search plug-in's search results dialog box.
`AcroSrch:Prev`	Goes to the previous hit in the Acrobat DC Search plug-in's results list.
`AcroSrch:Next`	Goes to the next hit in the Acrobat DC Search plug-in's results list.

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

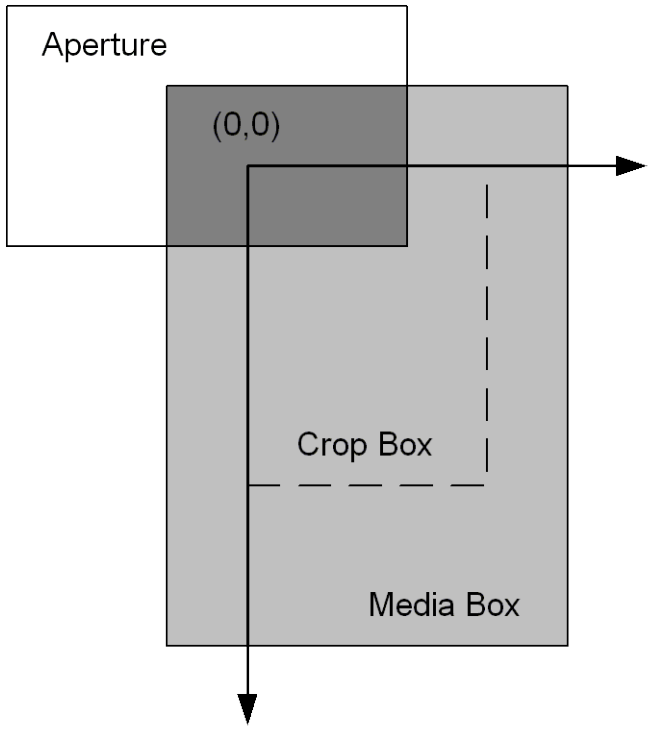


The default user space is the user space coordinate system in effect immediately before each page begins drawing. The origin of this coordinate system is the lower left corner of a page's media box. The x-coordinate increases to the right, and the y-coordinate increases upward. One unit in the default 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").

Device space coordinate system



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

D

E

F

S

T

U

V

W