



# **FDF Toolkit Overview and Reference**

**Technical Note #5194**

**Version :Acrobat 6.0**



**ADOBE SYSTEMS INCORPORATED**

**Corporate Headquarters**


345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

<http://partners.adobe.com>

*October 2003*



Copyright 2003 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the Adobe Systems Incorporated.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a "PostScript printing device," "PostScript display device," or similar item refers to a printing device, display device or item (respectively) that contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items that purport to be merely compatible with the PostScript language.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Distiller, PostScript, the PostScript logo and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third party rights.

# Contents

<b>Preface</b>	<b>7</b>
About This Document	7
Who Should Read This Document	7
Prerequisites	7
System Requirements	8
What's New In This Version Of The FDF Toolkit	8
Acrobat FDF Toolkit Download	9
Other Useful Documentation	9
Conventions Used in This Book	9
 <b>Chapter 1 Introduction</b>	 <b>11</b>
Acrobat Forms	11
Forms Data Format (FDF)	11
The FDF Toolkit	11
Submitting Data to a Server	12
Encoding	12
 <b>Chapter 2 FDF and Web Server Connectivity</b>	 <b>13</b>
Parsing FDF Data with the FDF Toolkit	13
Step One: Initializing the FDF Toolkit	14
Step Two: Opening the FDF File	14
Step Three: Getting Values from the Field	14
Step Four: Cleaning Up and Finalizing the Library	15
Sample Application	15
Creating the Client Side PDF File	17
Generating FDF Data with the FDF Toolkit	17
Sample Application to Generate FDF Data	20
Handling Errors	22
 <b>Chapter 3 FDF Toolkit for C/C++</b>	 <b>23</b>
Building Applications with the FDF Toolkit	23

Building Applications In UNIX . . . . .	23
Microsoft Windows Support . . . . .	23
Using Strings . . . . .	23
C/C++ FDF Toolkit Methods . . . . .	24
General Methods . . . . .	24
Methods That Parse FDF Data . . . . .	25
Methods That Generate FDF Data . . . . .	25
UNIX-Specific Methods . . . . .	27
<b>Chapter 4 FDF Toolkit for ActiveX . . . . .</b>	<b>29</b>
Introduction. . . . .	29
Installation . . . . .	29
Using the FDF Toolkit in Visual Basic . . . . .	29
Using the FDF Toolkit with Active Server Pages. . . . .	30
Setting Up the Server to Handle FDF Data . . . . .	30
Handling Errors in ActiveX . . . . .	31
FDF Toolkit Methods . . . . .	31
Methods Exposed By The FDFApp Object . . . . .	31
General Methods . . . . .	32
Methods That Parse FDF Data . . . . .	32
Methods That Generate FDF Data . . . . .	33
<b>Chapter 5 FDF Toolkit for Perl . . . . .</b>	<b>35</b>
Using the Perl FDF Toolkit . . . . .	35
Using Perl in Unix . . . . .	35
Using Perl in Windows. . . . .	36
Writing cgi-bin Applications Under IIS . . . . .	36
Perl FDF Toolkit Methods . . . . .	36
Functions Only Available in the Perl FDF Toolkit . . . . .	36
Functions Unavailable in the Perl FDF Toolkit . . . . .	36
General Methods . . . . .	37
Methods that Parse FDF Data . . . . .	37
Methods That Generate FDF Data . . . . .	38
<b>Chapter 6 FDF Toolkit For Java . . . . .</b>	<b>41</b>
Setup and Sample Usage of the Java FDF Toolkit. . . . .	41
Changes to the API . . . . .	41

Other Useful Documentation . . . . .	42
<b>Chapter 7 FDF Toolkit Reference . . . . .</b>	<b>43</b>
Language Differences . . . . .	43
Method Names. . . . .	43
Parameters . . . . .	43
Errors and Return Values . . . . .	44
Encoding. . . . .	44
Data Types . . . . .	44
Error Codes . . . . .	44
FDFItem . . . . .	46
FDFAppFace. . . . .	48
FDFScaleWhen . . . . .	48
FDFActionTrigger . . . . .	48
pdfFileSpec . . . . .	49
Methods . . . . .	51
FDFAddDocJavaScript . . . . .	51
FDFAddTemplate . . . . .	52
FDFClose . . . . .	55
FDFCreate. . . . .	56
FDFEmbedAndClose . . . . .	57
FDFEnumValues . . . . .	59
FDFExtractAppendSaves . . . . .	61
FDFExtractAttachment . . . . .	62
FDFFinalize . . . . .	64
FDFGetAP. . . . .	65
FDFGetEncoding . . . . .	66
FDFGetFDFVersion . . . . .	68
FDFGetFile . . . . .	69
FDFGetFlags . . . . .	71
FDFGetID . . . . .	73
FDFGetNthValue . . . . .	75
FDFGetOpt . . . . .	77
FDFGetOptNumElem . . . . .	80
FDFGetRichValue . . . . .	81
FDFGetStatus . . . . .	83
FDFGetValue . . . . .	85
FDFGetVersion . . . . .	87
FDFInitialize . . . . .	88
FDFNextFieldName . . . . .	89
FDFOpen . . . . .	91
FDFOpenFromBuf. . . . .	92
FDFOpenFromEmbedded. . . . .	93

FDFOpenFromFile . . . . .	95
FDFOpenFromStr . . . . .	96
FDFRegisterThreadsafeCallbacks . . . . .	97
FDFRemoveItem . . . . .	99
FDFSave . . . . .	100
FDFSavetoBuf . . . . .	101
FDFSavetoFile . . . . .	102
FDFSaveToStr . . . . .	103
FDFSetAP . . . . .	104
FDFSetAPRef . . . . .	106
FDFSetAS . . . . .	108
FDFSetEncoding . . . . .	109
FDFSetFDFVersion . . . . .	110
FDFSetFile . . . . .	111
FDFSetFileEx . . . . .	112
FDFSetFlags . . . . .	113
FDFSetGoToAction . . . . .	115
FDFSetGoToRAction . . . . .	117
FDFSetHideAction . . . . .	119
FDFSetID . . . . .	121
FDFSetIF . . . . .	122
FDFSetImportDataAction . . . . .	124
FDFSetJavaScriptAction . . . . .	126
FDFSetNamedAction . . . . .	128
FDFSetOnImportJavaScript . . . . .	130
FDFSetOpt . . . . .	131
FDFSetResetByNameAction . . . . .	133
FDFSetResetFormAction . . . . .	135
FDFSetRichValue . . . . .	137
FDFSetStatus . . . . .	138
FDFSetSubmitByNameAction . . . . .	139
FDFSetSubmitFormAction . . . . .	141
FDFSetTargetFrame . . . . .	143
FDFSetURIAction . . . . .	144
FDFSetValue . . . . .	146
FDFSetValues . . . . .	148
new . . . . .	150
newFromBuf . . . . .	151
newFromEmbedded . . . . .	152
EmbedAndClose . . . . .	153
Callbacks . . . . .	154
FDFEnumValuesProc . . . . .	154
ThreadsafeCallback . . . . .	155



# Preface

---

## About This Document

The Forms Data Format (FDF) Toolkit gives any server running UNIX or Microsoft Windows NT, 2000, or XP (web server software required) the capability to generate or parse FDF data for or from a form created by the Acrobat Forms plug-in. For Java, the Java Development Kit (JDK) version 1.2 or later is required.

This document provides an introduction to development using the FDF Toolkit. It describes FDF, the FDF Toolkit Software Development Kit (*SDK*), and support options. This document is divided into the following chapters:

- [Chapter 1, "Introduction,"](#) provides an introduction to the FDF toolkit.
- [Chapter 2, "FDF and Web Server Connectivity,"](#) discusses FDF and Web Server Connectivity. It provides instructions for building server-side applications with FDF (including parsing FDF and generating FDF).
- [Chapter 3, "FDF Toolkit for C/C++,"](#) describes the FDF Toolkit for C/C++.
- [Chapter 4, "FDF Toolkit for ActiveX,"](#) describes the FDF Toolkit for ActiveX.
- [Chapter 5, "FDF Toolkit for Perl,"](#) describes the FDF Toolkit for Perl.
- [Chapter 6, "FDF Toolkit For Java,"](#) describes the FDF Toolkit for Java.
- [Chapter 7, "FDF Toolkit Reference,"](#) provides a complete description of all functions, data structures and callbacks used in the FDF Toolkit.

Together these chapters provide background information on FDF, examples of use, and platform-specific information.

---

## Who Should Read This Document

You should read this document if you are a developer who is:

- looking for basic information on the FDF Toolkit and its capabilities
- looking to use Adobe's FDF Toolkit to produce FDF directly, either in C, Active X, Java, or Perl.

---

## Prerequisites

You are assumed to have knowledge of the following:

- The PDF file format.
- The process of creating form fields and modifying their properties. See the Acrobat Help document, accessible via the menu item **Help -> Acrobat Help**.
- The process flow for HTML forms, on both the client and server. Technical bookstores generally stock a number of excellent books that cover this topic.
- At least one of the following languages: C/C++, Visual Basic, Java, or Perl.

## System Requirements

You need to install the FDF Toolkit on a computer with the following requirements:

- Windows NT, 2000, or XP (web server software required) or UNIX.
- For Java, the Java Development Kit (JDK) version 1.2 or later is required.
- Administrator privileges.
- A Web Server, such as IIS or Apache.
- A development environment for at least one of the following languages: C/C++, Java, Perl, or any scripting language supported by Active Server Pages (ASP).
- An e-mail account (optional, but strongly suggested).

---

## What's New In This Version Of The FDF Toolkit

This version of the FDF Toolkit contains several new features, including:

- New **FDFGetRichValue** and **FDFSetRichValue** APIs for C/C++, Perl, ActiveX, and Java implementations.
- Support for greater than 32K characters per field.
- New samples.






---

## Acrobat FDF Toolkit Download

To download all or part of the FDF Toolkit, which includes support for writing Web applications in C/C++, Perl and ActiveX, go to:

<http://partners.adobe.com/asn/developer/acrosdk/forms.html>

---

## Other Useful Documentation

The Acrobat SDK includes other books that you might find useful. It is available at the Adobe Solutions Network web site at <http://partners.adobe.com/asn/tech/pdf/index.jsp>.

In particular, you should be familiar with the Portable Document Format (PDF). The *PDF Reference* provides a complete description of the PDF file format.

Also the document *Forms System Implementation Notes* (formsys.pdf), available on the <http://partners.adobe.com/asn/tech/pdf/index.jsp>, contains more information on using Acrobat forms.

---

## Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions.

Font	Used for	Examples
monospaced	Paths and filenames	C:\templates\mytmpl.fm
	Code examples set off from plain text	These are variable declarations: AVMenu commandMenu,helpMenu;
monospaced bold	Code items within plain text	The <b>GetExtensionID</b> method ...
	Parameter names and literal values in reference documents	The enumeration terminates if <b>proc</b> returns <b>false</b> .
monospaced italic	Pseudocode	ACCB1 void ACCB2 ExeProc(void) { <i>do something</i> }
	Placeholders in code examples	AFSimple_Calculate( <i>cFunction</i> , <i>cFields</i> )

---

Font	Used for	Examples
blue	Live links to Web pages	The Acrobat Solutions Network URL is: <a href="http://partners/adobe.com/asn/">http://partners/adobe.com/asn/</a>
	Live links to sections within this document	See <a href="#">Using the SDK</a> .
	Live links to other Acrobat SDK documents	See the <a href="#">Acrobat Core API Overview</a> .
	Live links to code items within this document	Test whether an <a href="#">ASAtom</a> exists.
bold	PostScript language and PDF operators, keywords, dictionary key names	The <b>setpagedevice</b> operator
	User interface names	The <b>File</b> menu
italic	Document titles that are not live links	<i>Acrobat Core API Overview</i>
	New terms	<i>User space</i> specifies coordinates for...
	PostScript variables	<i>filename</i> <b>deletefile</b>

# 1

## Introduction

This chapter describes the FDF Toolkit that creates and/or parses FDF-formatted text for Acrobat forms.

---

### Acrobat Forms

Acrobat Forms provides the ability to electronically fill out PDF-based forms. Forms can contain annotations that represent text fields, action buttons, radio buttons, check boxes, list boxes and combo boxes.

Acrobat Forms are a group of PDF extensions. You can consider these extensions a layer on top of a PDF file. You can create the underlying PDF file with any PDF producer such as PDFMaker, Acrobat Distiller, or Acrobat Capture. Then you can manually add the fields using Acrobat.

---

### Forms Data Format (FDF)

The Forms Data Format (FDF) is described in detail in the *PDF Reference* (available at <http://partners.adobe.com/asn/tech/pdf/index.jsp>).

FDF is used:

- when submitting form data to a server, receiving the response, and incorporating it into the form.
- to generate ( “export”) stand-alone files containing form data that can be stored, transmitted electronically (for example, via e-mail), and imported back into the corresponding form.
- to control the document structure. Constructs within FDF allow it to specify Acrobat forms to be used in the creation of new PDF documents. You can use this functionality to create complex documents dynamically.
- to define a container for annotations that are separate from the PDF document to which the annotations apply.

---

### The FDF Toolkit

The FDF Toolkit is a thread-safe API (application programming interface) for writing a server application that generates FDF data or parses FDF data from a form created by the Acrobat

Forms plug-in. The FDF Toolkit supports C/C++, Perl, Java, and any scripting language supported by ASP (such as Visual Basic).

---

## Submitting Data to a Server

After filling in an Acrobat form, the user must click on a button to submit the data to a server. Submitted data may be in one of the following formats:

- HTML-compatible (MIME type **application/x-www-form-urlencoded**): The format submitted is identical to HTML form submissions. You can use existing CGI scripts for HTML forms to parse data in this format.
- FDF (an Acrobat-specific Forms Data Format: MIME type **application/vnd.fdf**). The FDF Toolkit library helps parse and generate FDF files.
- XFDF (for XML-based FDF files): MIME type **application/vnd.adobe.xfdf**.

The creator of the form decides which format to use.

The URL submission target is not restricted to the http scheme. For example, it can also be the mailto scheme, for example, **mailto:someuser@somecompany.com**.

**NOTE:** To see the format of FDF data that is being sent to the server, create a form and enter data into one or more of its fields. Instead of submitting it to the server, select the **Export Form Data...** menu item from Acrobat's **File** menu.

For detailed information on posting form data to a web server using HTML or FDF, see the document *Form System Implementation Notes* (`formsys.pdf`), available from <http://partners.adobe.com/asn/tech/pdf/index.jsp>. For detailed information on the XFDF specification and the XFDF schema, see <http://partners.adobe.com/asn/tech/pdf/xmlformspec.jsp>.

---

## Encoding

When using HTML forms and submitting data to the Web server, that data may be transmitted using some encoding. For example, in Japan, that encoding is typically *Shift-JIS*.

FDF has the ability to transmit/receive Forms data in FDF using an encoding that is different from the one used internally in PDF. This scheme indicates inside FDF which encoding is being used for transmission.

The **/Encoding** key defines which encoding is being used inside the FDF data for any string that is the value of the **/V** key, and any string that is an element of the **/Opt** array. The **/V** key may also have a name as its value (for the case of fields of type checkbox or radio button),

For Acrobat 6.0, acceptable values of the **/Encoding** key are *Shift-JIS*, *UHC*, *GBK*, and *BigFive*.

# 2

## FDF and Web Server Connectivity

You can parse an FDF file to extract data from the individual fields. You can then use this data to populate a database or provide input to some other application. You can also generate FDF data and save it to a file or buffer to populate a PDF file.

Exporting the data in FDF format allows you to do a number of things:

- populate the same PDF file on the client with new data
- send graphical information back to a PDF file
- alter a PDF file on the client
- construct a PDF file with templates

Generally you export data to HTML format only when you have some existing web application that parses a web page and you want to retain your current code.

---

### Parsing FDF Data with the FDF Toolkit

To parse FDF data from an FDF file using the FDF Toolkit, the Web server application needs to open the FDF file or buffer and use the various methods in the Toolkit to extract the data. Your application can generate a response (acknowledging some action has taken place) or otherwise process this data. For example, it could populate a database with the incoming form data or process the field data to be used in an order entry system. The final step in either case involves cleaning up any open network connections and closing any open data buffers.

As an illustration, let's say we have a company that sells office supplies. We have an online catalog with an order form that users can fill out to submit the items they wish to purchase. You can write an application to parse the FDF data coming in from a PDF file (from a Web client, for example). To parse an FDF file you need to:

1. Initialize the library.
2. Open the FDF data from a file or a buffer.
3. Get the values of the data from the individual fields (usually contained in the **/V** key in the FDF data).
4. Close the FDF data buffers and free any resources used.

We will now detail these steps further.

## Step One: Initializing the FDF Toolkit

On UNIX systems, if you are using the C/C++ version of the FDF Toolkit library, you must first call **FDFInitialize** to initialize the Toolkit. No other calls to API methods will work until after you initialize the Toolkit. This is only true for the C/C++ versions of the library; you do not need to initialize the Toolkit with the Perl, ActiveX or Java implementations.

When developing on Microsoft Windows systems, you do not need to initialize and finalize the library. This automatically occurs when your application opens the FDF data. Once the FDF data is opened, you can make calls to the rest of the API and use **FDFClose** to close any open FDF files.

## Step Two: Opening the FDF File

The first section of your code should open the FDF file either from a source file or from a standard http port from the Web client. To open the FDF file, use the **FDFOpen** command and pass the following parameters:

- The pathname to the FDF file or, to read from **stdin**, "-".
- The number of bytes to read. If you are opening a file, pass "0" for this parameter.
- A pointer to an **FDFDoc** object. The API can then reference this pointer.

The following example shows how you call **FDFOpen**:

```
ASInt32 howMany = atoi(getenv("CONTENT_LENGTH"));
errorCode = FDFOpen("-", howMany, &FdfInput);
```

## Step Three: Getting Values from the Field

Once the FDF file or data buffer is opened, the field values can be parsed by several **get** methods. The most common method used is **FDFGetValue**.

The **FDFGetValue** method gets field values of the specified field. If you open any FDF file in a text editor, you will find the **/V** key, which points to the value of the field. The following example illustrates this:

```
%FDF-1.2
%,,,
1 0 obj
<<
  /FDF << /Fields [ << /V (John Peppermint)/T (Name)>> ] /F (simple.pdf)>>
```

The **/V** key points to the value of "John Peppermint". The name of the field is specified by the **/T** key. In this case, the name of the field is "Name".

To get the values of the key, reference the **FDFDoc** object passed by the **FDFOpen** method, specify the field you are interested in, and the buffers that will store the data. Your application uses the data returned in this buffer (for example, the address or name of a customer). The following example shows how to use **FDFGetValue** to get the field values of a field named **Customer.Name**:

```
errorCode = FDFGetValue (FdfInput, "Customer.Name", cNameBuffer,
sizeof(cNameBuffer), &howMany);
```

## Step Four: Cleaning Up and Finalizing the Library

Once data processing has been completed, you must free the resources the toolkit used. Use **FDFClose** to close any open **FDFDoc** objects. If you are writing a C application for a UNIX system, you must call **FDFFinalize** to finalize the library and free resources used by the Toolkit.

## Sample Application

[Example 2.1](#) is a sample C application that parses a simple PDF file coming in from a Web client. You could also write this application in Perl or ActiveX (for use with Microsoft's Active Server Pages) by using the other flavors of the FDF Toolkit. (See the subsequent chapters of this technical note for more information.) This particular example runs on a Windows NT system using Microsoft's IIS version 3.0, and extracts data from a PDF document displayed on the Web client.

### EXAMPLE 2.1 Parsing A PDF File From A Web Client

```
/*
** Copyright 1999 Adobe Systems, Inc.
**
** parseFDF - A sample Windows NT application to parse FDF data
** from "stdin".
**/

#include <stdio.h>
#include <stdlib.h>
#include "fdftk.h"

void main()

{
    /*
    ** Definitions
    **/

    FDFDoc FdfInput;
    FDFErc errorCode;
    char cNameBuffer [50];
    char cAddrBuffer [50];
    char cComboBuffer [50];

    ASInt32 howMany = atoi(getenv("CONTENT_LENGTH"));

    /*
```

```

** FDFOpen:
*/

errorCode = FDFOpen ("-", howMany, &FdfInput);

/*
** FDFGetValue:
*/

errorCode = FDFGetValue (FdfInput, "Customer.Name",
                          cNameBuffer, sizeof(cNameBuffer), &howMany);

errorCode = FDFGetValue (FdfInput, "Customer.Address",
                          cAddrBuffer, sizeof(cNameBuffer), &howMany);

errorCode = FDFGetValue (FdfInput, "My Combo Box",
                          cComboBuffer, sizeof(cComboBuffer), &howMany);

/*
** Presumably after we parsed this data, we would populate a
database
** or generate another FDF file with some sort of response ...
*/

    // Your code goes here

/*
** This next line of code is important if you are returning FDF
** You must emit the correct MIME type to "stdout" or you'll get a
** CGI error simular to this:
**
**          "The specified CGI application misbehaved by not
returning
**          a complete set of HTTP headers."
**
** At this point you would generate FDF for the return and
** then do this:
**
**          printf ("Content-type: application/vnd.fdf\n\n");
**          fflush (stdout);
**
** For this example we will only send back acknowledgement that
the
** code worked
**/

    printf ("Content-type: text/plain\n\n");
    printf ("Parsing of the submitted FDF completed\n");

```



```
/*
** FDFClose:
**
**      Use FDFClose to close any open FDFDocs and free resources
**/

errorCode = FDFClose(FdfInput);

}
```

---

## Creating the Client Side PDF File

When building form fields that will be sent to a Web server, it is necessary to create and define a button that can be used to submit all or some of the data. The submit button on the PDF file must have a button defined with the "Submit Form" action pointing to the Web server and the complete path to the Web application program. If you are submitting from an Acrobat Form and the server returns FDF data, then your URL must end in "#FDF". For example:

```
http://localhost/cgi-bin/parseFDF.exe#FDF
```

In this case, the Submit Form action in the PDF file points to the Web server called "localhost" and the application called `parseFDF.exe` to extract FDF data.

**NOTE:** You must use upper-case "FDF" in the pathname.

On the other hand, if you are submitting from an HTML form, then the URL for submitting Form data to the server does not need to end with "#FDF". However, the returned FDF data must include an **/F** key giving the full URL of the targeted PDF file. Acrobat automatically loads the PDF file upon opening the FDF file. You set the **/F** key with the **FDFSetFile** method (see [Step Three: Setting Field Values](#).)

**NOTE:** The FDF file does not require an **/F** key if the FDF data is for the same form that was originally submitted. It does require an **/F** key if the FDF data is for a different form than the one originally submitted.

## Generating FDF Data with the FDF Toolkit

The Web server application creates FDF data using methods in the library to initialize and to create a pointer to the **FDFDoc** object. From this point other methods in the Toolkit can reference the **FDFDoc** object to set the values and actions in the form fields. After you have set the values and actions, you can save the FDF data to a file or buffer, clean up and close any open buffers.

The steps to create an FDF file are:

1. Initialize the library.

2. Call methods to create the FDF data.
3. Set the values of the individual fields (the **/N** key in the FDF data).
4. Set actions for fields.
5. Target FDF data at PDF files.
6. Close the FDF data buffers and free any resources used.

In addition, if the FDF data you generate requires a different PDF file than the one data was received from, you must set the value of this new PDF file with the **FDFSetFile** method.

### Step One: Initializing the FDF Library

On UNIX systems, if you are using the C/C++ version of the FDF Toolkit library, you must first call **FDFInitialize** to initialize the Toolkit. You can only call other API methods after you initialize the Toolkit. There is no need to initialize the Toolkit for the Perl, ActiveX or Java versions of the library.

When developing on Microsoft Windows systems, you do not need to initialize and finalize the library. This automatically occurs when your application opens the FDF data. Once the FDF data is opened, you can make calls to the rest of the API and use **FDFClose** to close any open FDF files.

### Step Two: Creating FDF Data with the Toolkit

To build FDF data to be used in a file or by a buffer, use the **FDFCreate** method. The parameter to pass is a pointer to an **FDFDoc** object, which represents the FDF data used in a form field. Other methods in the library can then reference this object.

The following example shows a C example of this call:

```
FDFDoc FdfOutput = NULL;
errorCode = FDFCreate (&FdfOutput);
```

### Step Three: Setting Field Values

You use **FDFSetValue** (or its language-specific equivalent) to set values for any particular field. When adding field data to a new **FDFDoc** object, if the field does not exist in the FDF file— such as when creating FDF data from scratch — use a placeholder with the name of the field to create the field. The FDF field is set with the specified value.

**FDFSetValue** takes as arguments:

- A pointer to the **FDFDoc** object returned from **FDFCreate**.
- A string representing the fully-qualified name of the field (for example "customer.name.last").
- A string to use as the new value of the field.

The following example shows a C example of using this call. The field value of the text field "Customer.Address" is set to "12 Saratoga Ave".

```
errorCode = FDFSetValue (FdfOutput, "Customer.Address",
                          "12 Saratoga Ave", false);
```

### Step Four: Set Actions with FDF Data

In PDF 1.1, the presence of an **/A** key or **/Dest** key in an annotation or outline entry denotes an action that is to be performed when the mouse button is released after clicking inside the annotation or outline entry. PDF 1.2 provides a more general mechanism by defining other “trigger points” (events) and associating actions with each one by means of an “additional actions” dictionary, that is included in an annotation or outline entry as the value of the **/AA** key.

With the FDF Toolkit, you may use the **/A** and the **/AA** keys to set or change actions for a form field. The PDF version 1.3 specification defines several subtypes of actions. The FDF Toolkit takes advantage of the following actions:

- **GoTo** — Changes the current page view to a specified page and zoom factor.
- **GoToR** — Opens another PDF file (as specified by the F key) at a specified page and zoom factor (“for example, GoTo Remote”).
- **URI** — Resolves the specified Uniform Resource Identifier (URI).
- **Hide** — Sets or clears the Hidden flag for an annotation.
- **SubmitForm** — Send data to a URL.
- **ResetForm** — Set field values to their defaults.
- **ImportData** — Import field values from a file.

The following example shows how to change the value of a submit button whose action was to submit the FDF data from a Web server named “Alpha” to a Web server named “Beta”. You use the FDF Toolkit method **FDFSetSubmitFormAction**:

```
errCode = FDFSetSubmitFormAction (FdfOutput, "Submit Button", FDFUp,
    "http://beta/cgi-bin/myscript.exe#FDF", 4);
```

The data is now submitted to the Web server named “Beta”.

Other actions that can be performed include changing the current page view to a different page with the **GoTo** action. You can also use these methods to change an action of a field to point to other PDF files. For this you would use the **GoToR** action. You can even use the **ResetForm** action to reset all or specific form fields.

### Step Five: Targeting FDF Data to a PDF File

Because you can use FDF data to populate a PDF file, you can also use the FDF Toolkit to set the file specification of the target PDF. This data could point to the same PDF file that the submission came from, another PDF file on your Web server, or to the local client.

To do this, use the **FDFSetFile** method that takes the relative pathname of the PDF file. If you plan to have the PDF file on a remote server, you need to use the full path of the PDF file.

The following example assumes the user filled in a PDF file named `order-form.pdf` and you want to return results of the order into a different form called `completed-order.pdf`. Use **FDFSetFile**:

```
errCode = FDFSetFile (FdfOutput, "http://localhost/PDF/completed-
order.pdf");
```

```

/* This takes the newly-generated FDF data and points it to a new file
called "completed-order.pdf". To complete the task, you must emit the
correct MIME type and flush the FDF data back to the server with the
FDFSave method*/
printf("Content-type: application/vnd.fdf\n\n");
fflush(stdout);
FDFSave (FdfOutput, "-");
/* In this example, FDFSave sends the data back to the Web client. You
can also save the FDF data to a file by specifying the full pathname to
the file in place of the "-". */

```

## Step Six: Cleaning Up and Finalizing the Library

Use **FDFClose** to close any open **FDFDocs** and free resources used by the library. If you are writing a C application for a UNIX system, you must call **FDFFinalize** to finalize the library.

## Sample Application to Generate FDF Data

**Example 2.2** contains a sample application that shows how you can use the FDF Toolkit to generate FDF data. This code was written on a Windows NT system with Visual C++ v5.0:

### EXAMPLE 2.2 Full Sample of FSF Data Generation

```

/******
*
Copyright 1999 Adobe Systems, Inc.

    generateFDF - A sample Windows NT Server application to generate
    FDF data and send it to "stdout".

*****
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
    The line below includes the standard FDF Toolkit Header file.
*/

#include "Fdfstk.h"

/*
    Main application
*/

void main()

{

```

```

FDFErc retCode;
FDFDoc FdfOutput = NULL;

/*
    Create a new FDF. Parameter is the pointer to FDFDoc every call
    uses.
*/
retCode = FDFCreate (&FdfOutput);

/*
    FDFSetValue.
*/

retCode = FDFSetValue (FdfOutput, "Date", "December 31 1999",
false);
retCode = FDFSetValue (FdfOutput, "Name", "James Clay", false);
retCode = FDFSetValue (FdfOutput, "Address", "12 Saratoga Road",
false);
retCode = FDFSetValue (FdfOutput, "City", "Monte Sereno", false);

/*
    Set the target PDF inside the outgoing FDF.
*/
retCode = FDFSetFile(FdfOutput,
    "http://localhost/PDF/generateFDF.pdf");

/*
    Next we'll do three things:

    1) Set everything up to emit the correct HTTP header for the
        MIME type. In the case with FDF, the MIME needs to be set to
        "application/vnd.fdf".
    2) Emit the HTTP header
    3) Write the FDF data to stdout
*/
printf("Content-type: application/vnd.fdf\n\n");
fflush(stdout);
FDFSave (FdfOutput, "-");

/*
    You don't have to make this next call. It just shows that after the
    data is flushed to standard out you may want to save it locally.
    you can then open the FDF by clicking on it or opening it with
    Acrobat. It's a lot easier than saving the whole PDF with the data.

    We want a record of the transaction, so save the FDF data to a
    specific place on the hard drive.
*/
FDFSave (FdfOutput,

```

```
        "C:\\InetPub\\wwwroot\\Pdf\\generated-data.fdf");  
    */  
  
    /*  
    Close the FDF File used to generate the output.  
    */  
    retCode = FDFClose (FdfOutput);  
  
}
```

### Debugging Tip

The following example shows how you can emit the MIME type plain/text (instead of application/vnd.fdf) when sending the FDF data:

```
printf ("Content-type: text/plain\n\n");  
printf ("Generated FDF and sent it to the client. \n");
```

This displays the text “Generated FDF and sent it to the client” in the browser windows to aid the debugging process.

---

## Handling Errors

In C/C++ and Perl most functions return error codes. In ActiveX and Java, functions raise exceptions. See [“Error Codes” on page 44](#) for detailed information on errors.

# 3

## FDF Toolkit for C/C++

This chapter describes how to use the C/C++ version of the FDF Toolkit. It lists the methods that can be used, with links to detailed descriptions in the reference chapter.

---

### Building Applications with the FDF Toolkit

To prepare your development environment for FDF Toolkit use, copy the C header file `fdftk.h` into your project's directory and include it your source code:

```
#include "Fdftk.h"
```

---

### Building Applications In UNIX

On UNIX systems, you must first call **FDFInitialize** to initialize the library. You cannot call other methods in the Toolkit until you have initialized the library. When your application finishes making calls to the library, call **FDFFinalize**.

**NOTE:** Remember to place the shared libraries into the proper location so that the operating system can find them.

---

### Microsoft Windows Support

`FDFTK.LIB` is a Win32-specific import library. To link this library, add it to the project.

When running your application on the target Windows system, copy the `FDFTK.DLL` supplied with this API into the same directory as your executable, or into the Windows system directory. It can also be put in other directories as long as that directory is included in the **PATH** statement in the `AUTOEXEC.BAT`.

---

### Using Strings

All strings in the FDF Toolkit API must be null-terminated. Every call specifies which encodings are acceptable for every string. Strings that are in Host- or PDFDocEncoding end with a single null byte. Strings in Unicode have `'\xfe'` as byte 0 and `'\xff'` as byte 1. Additionally, they must be terminated with 2 null bytes. Field names are always in PDFDocEncoding, and they always represent the fully qualified name of the field (e.g.

**employee.name.last**). PDFDocEncoding is described in Appendix D of the *PDF Reference, Fourth Edition*.

---

## C/C++ FDF Toolkit Methods

This section lists the FDF Toolkit methods for C/C++. The name of each method links to a complete description of the method.

- [General Methods](#)
- [Methods That Parse FDF Data](#)
- [Methods That Generate FDF Data](#)
- [UNIX-Specific Methods](#)

### General Methods

The following methods open, close and save FDF files and perform other general functions.

Method	Description
<a href="#">FDFGetVersion</a>	Gets the current version of the FDF Toolkit library.
<a href="#">FDFOpen</a>	Opens existing FDF file or reads stdin in C.
<a href="#">FDFOpenFromEmbedded</a>	Opens an FDF that is embedded in a container.
<a href="#">FDFRemoveItem</a>	Removes key-value pairs from the FDF file.
<a href="#">FDFClose</a>	Frees resources used by the FDF file.
<a href="#">FDFEmbedAndClose</a>	Embeds FDF in another document and then closes it.
<a href="#">FDFSave</a>	Writes out an FDF file.



## Methods That Parse FDF Data

The following methods parse FDF data.

TABLE 3.1

Method	Description
<a href="#">FDFEnumValues</a>	Enumerates field names and values in the FDF file.
<a href="#">FDFExtractAppendSaves</a>	Extracts incremental changes submitted within an FDF, and makes a new file out of it.
<a href="#">FDFExtractAttachment</a>	Extracts an uploaded file and creates a new file out of it.
<a href="#">FDFGetAP</a>	Gets the appearance of a field ( <b>/AP</b> ) and creates a PDF document out of it.
<a href="#">FDFGetEncoding</a>	Gets the value of the FDF <b>/Encoding</b> key as a string.
<a href="#">FDFGetFDFVersion</a>	Gets the current version of the FDF.
<a href="#">FDFGetFile</a>	Gets the value of the <b>/F</b> key.
<a href="#">FDFGetFlags</a>	Gets the flags of a field ( <b>/Ff</b> or <b>/F</b> keys).
<a href="#">FDFGetID</a>	Gets the value of one element in the FDF's <b>/ID</b> key.
<a href="#">FDFGetNthValue</a>	Gets an element from a field's value if it is an array.
<a href="#">FDFGetOpt</a>	Gets the value of one element in a field's <b>/Opt</b> array.
<a href="#">FDFGetRichValue</a>	Gets the value of the <b>/RV</b> key.
<a href="#">FDFGetStatus</a>	Gets the value of the <b>/Status</b> key.
<a href="#">FDFGetValue</a>	Gets the value of a field ( <b>/V</b> key).
<a href="#">FDFNextFieldName</a>	Gets the next field name.

## Methods That Generate FDF Data

The following methods generate FDF data.

TABLE 3.2

Method	Description
<a href="#">FDFAddDocJavaScript</a>	Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.

TABLE 3.2

Method	Description
<a href="#">FDFAddTemplate</a>	Adds a template to an FDF file.
<a href="#">FDFCreate</a>	Creates a new FDF file.
<a href="#">FDFSetAP</a>	Sets the appearance of a button field ( <b>/AP</b> key) from a PDF document.
<a href="#">FDFSetAPRef</a>	Sets a reference to a PDF document to use for the appearance of a field (one of the faces within the <b>/APRef</b> key).
<a href="#">FDFSetAS</a>	Sets the <b>/AS</b> key.
<a href="#">FDFSetEncoding</a>	Sets the value of the <b>/Encoding</b> key.
<a href="#">FDFSetFDFVersion</a>	Sets the FDF version.
<a href="#">FDFSetFile</a>	Sets the value of the <b>/F</b> key (string specification).
<a href="#">FDFSetFileEx</a>	Sets the value of the <b>/F</b> key (complex file specification).
<a href="#">FDFSetFlags</a>	Sets the value of one of the <b>/Ff</b> , <b>/F</b> , <b>/SetFf</b> , <b>/ClrFf</b> , <b>/SetF</b> , or <b>/ClrF</b> flags of a field.
<a href="#">FDFSetGoToAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoTo</b> .
<a href="#">FDFSetGoToRAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoToR</b> .
<a href="#">FDFSetHideAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>Hide</b> .
<a href="#">FDFSetID</a>	Sets the value of one element in the <b>/ID</b> key.
<a href="#">FDFSetIF</a>	Sets the Icon Fit attribute for the appearance of a button field ( <b>/IF</b> key).
<a href="#">FDFSetImportDataAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>ImportData</b> .
<a href="#">FDFSetJavaScriptAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>JavaScript</b> .
<a href="#">FDFSetNamedAction</a>	Sets <b>/A</b> or <b>/AA</b> key to a named action.
<a href="#">FDFSetOnImportJavaScript</a>	Adds a script to the FDF that will execute when it is imported.
<a href="#">FDFSetOpt</a>	Sets the value of one element in a field's <b>/Opt</b> key.
<a href="#">FDFSetResetByNameAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>ResetForm</b> using field names.

TABLE 3.2

Method	Description
<a href="#">FDFSetResetFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>ResetForm</b> .
<a href="#">FDFSetRichValue</a>	Sets the value of the <b>/RV</b> key.
<a href="#">FDFSetStatus</a>	Sets the value of the <b>/Status</b> key.
<a href="#">FDFSetSubmitByNameAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>SubmitForm</b> using field names.
<a href="#">FDFSetSubmitFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>SubmitForm</b> .
<a href="#">FDFSetTargetFrame</a>	Sets the value of the <b>/Target</b> key.
<a href="#">FDFSetURIAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>URI</b> .
<a href="#">FDFSetValue</a>	Sets the value of a field ( <b>/V</b> key).
<a href="#">FDFSetValues</a>	Sets the value of a field to an array.

## UNIX-Specific Methods

The following methods are specific to the Unix platform.

Method	Description
<a href="#">FDFInitialize</a>	Initializes the FDF Toolkit library.
<a href="#">FDFFinalize</a>	Frees resources in the FDF Toolkit library.
<a href="#">FDFRegisterThreadsafeCallbacks</a>	Registers callbacks for thread-safe operations for lock, unlock and destroy.



# 4

## FDF Toolkit for ActiveX

---

### Introduction

FdfAcX.dll is a Win32 “in-process” ActiveX server component (formerly known as OLE automation server) that you can also use in conjunction with Internet Information Server (IIS) and Active Server Pages (ASP).

The examples in this document are Microsoft Visual Basic or VBScript. They assume you are using VBScript in an ASP environment.

---

### Installation

Install FdfAcX.dll in a directory that has “execute” permissions. With Microsoft’s Internet Information Server version 3 using ASP, a good location (using Windows NT for the example) is \WINNT\system32\inetsrv\ASP\Cmpnts. With IIS version 4 or later, use \WINNT\system32.

If you are running on an NTFS file system, the DLLs themselves need the proper execute permissions.

FdfAcX.dll uses FdfTk.dll. Place both in the same directory, or in the C:\WINNT\System32 directory.

FdfAcX.dll needs to be registered with Windows NT. Use the Windows NT “regsvr32” Program. To register FdfAcX.dll, copy the file to \WINNT\System32\Inetsrv\ASP\Cmpnts directory or a location you have chosen and at the Windows NT Command Prompt window type:

```
C:\WINNT> cd \WINNT\System32\Inetsrv\ASP\Cmpnts
C:\WINNT\System32\Inetsrv\ASP\Cmpnts> regsvr32 FdfAcX.dll
```

---

### Using the FDF Toolkit in Visual Basic

FdfAcX.dll exposes one main object: **FdfApp.FdfApp**. [Example 4.1](#) shows a possible use of the Visual Basic 5.0 main object:

**EXAMPLE 4.1 Use of VB Main Object**

```
Dim FdfAcX As FDFACXLib.FdfApp
Set FdfAcX = CreateObject("FdfApp.FdfApp")
```

You can now make other calls to the FDF Toolkit library to parse or generate FDF data.

## Using the FDF Toolkit with Active Server Pages

This version of the FDF Toolkit also supports using the Toolkit with Active Server Pages (ASP). From an ASP document using VBScript, you would use the object like this:

```
<% Set FdfAcX = Server.CreateObject("FdfApp.FdfApp") %>
```

Use the methods the same as you would in VB.

## Setting Up the Server to Handle FDF Data

When using the FDF Toolkit with ASP, you must make sure that when returning FDF to the client browser you have *application/vnd.fdf* defined as a valid MIME type in the registry. If your Web site includes files that are in multiple formats, your computer must have a MIME mapping for each file type. If MIME mapping on the server is not set up for a specific file type, browsers may not be able to retrieve the file.

If the MIME type is not defined, you will see a **Save file as ...** dialog box on Internet Explorer Web browsers when opening FDF files. This indicates the MIME type is unknown by the server (in this case, the unknown MIME type is "FDF"). When this happens, the MIME type is specified by Windows as an asterisk (\*). This is the default MIME type used in Windows when a MIME mapping does not exist.

For example, to handle a request for the file *myfile.foo* when the file-name extension ".foo" is not mapped to a MIME type, your computer will use the MIME type specified for the asterisk extension, which is the type used for binary data. When this happens, this will cause the Internet Explorer browser to save the file to disk.

To configure additional MIME mappings use a Registry Editor and open:

```
HKEY_LOCAL_MACHINE
```

Search for the following key:

```
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\MimeMap
```

Once found, add a REG\_SZ value for the MIME mapping required.

Using Regedt32.exe, the syntax is:

```
application/fdf,fdf,,5
```

Using Regedit.exe:

1. From the menu select **Edit-> New-> String Value**
2. Enter the value:  

```
application/fdf,fdf,,5
```
3. Reboot your system.

**NOTE:** Windows 2000, Windows XP, or Windows NT Server 4.0 with Service Pack 3 or later must be installed.

---

## Handling Errors in ActiveX

All functions may raise an exception. Some of the possible ones are standard OLE exceptions, such as **E\_OUTOFMEMORY (0x8007000E)**. Others are FdfAcX-specific. For these, the numeric value, as well as the description, are derived from **FDFErrc**. See [“Error Codes” on page 44](#) for a complete list of FDF Toolkit-specific errors.

For example, one possible exception has a numeric value of **3** and a description of **FDFErrcFileSysErr**. 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. Therefore, the numeric value of the exception is hex 0x80040202. The important part is the rightmost hex 202, which is the third error (200 is the first).

**NOTE:** In the ActiveX implementation, there is no corresponding string for the case where there is no error, or error number 0. In this case, be sure to write your application comparing the number property to 0 rather than the string value to **FDFErrcOK**.

---

## FDF Toolkit Methods

This section lists the FDF Toolkit methods for ActiveX. The name of each method links to a complete description of the method.

- [Methods Exposed By The FDFApp Object](#)
- [General Methods](#)
- [Methods That Parse FDF Data](#)
- [Methods That Generate FDF Data](#)

The examples used in this document are written in Visual Basic or VBScript (the latter assumes you are using an Active Server Pages (ASP) environment).

### Methods Exposed By The FDFApp Object

The following methods are available for the **FdfApp.FdfApp** object:

Method	Description
<a href="#">FDFCreate</a>	Creates a new FDF file.
<a href="#">FDFGetVersion</a>	Returns the version of the ActiveX component.

Method	Description
<a href="#">FDFOpenFromFile</a>	Opens an existing FDF file.
<a href="#">FDFOpenFromBuf</a>	Opens an FDF from a buffer.
<a href="#">FDFOpenFromStr</a>	Opens an FDF from a string.

These methods (with the exception of **FDFGetVersion**) return an object of type **FDFACXLib.FdfDoc** which has the remaining methods in the API.

## General Methods

The following are general utility functions.

Method	Description
<a href="#">FDFClose</a>	Frees resources used by the FDF file.
<a href="#">FDFEmbedAndClose</a>	Embeds FDF in another document and then closes it
<a href="#">FDFOpenFromEmbedded</a>	Opens FDF data that is held in a container.
<a href="#">FDFRemoveItem</a>	Removes a key-value pair from the FDF data.
<a href="#">FDFSavetoBuf</a>	Writes FDF data to a buffer.
<a href="#">FDFSavetoFile</a>	Saves FDF data to a file.
<a href="#">FDFSavetoStr</a>	Writes FDF data to a string.

## Methods That Parse FDF Data

The following functions parse FDF data.

Method	Description
<a href="#">FDFExtractAppendSaves</a>	Extracts incremental changes submitted within an FDF, and makes a new file out of it.
<a href="#">FDFExtractAttachment</a>	Extracts an uploaded file and creates a new file out of it.
<a href="#">FDFGetAP</a>	Gets the appearance of a field ( <b>/AP</b> ) and creates a PDF document out of it.
<a href="#">FDFGetFile</a>	Gets the value of the <b>/F</b> key.
<a href="#">FDFGetFlags</a>	Gets the flags of a field ( <b>/Ff</b> or <b>/F</b> keys).



Method	Description
<a href="#">FDFGetID</a>	Gets the value of one element in the FDF's <b>/ID</b> key.
<a href="#">FDFGetNthValue</a>	Gets an element from a field's value if it is an array.
<a href="#">FDFGetOpt</a>	Gets the value of one element in a field's <b>/Opt</b> array.
<a href="#">FDFGetOptNumElem</a>	Gets the number of elements in the <b>/Opt</b> key.
<a href="#">FDFGetRichValue</a>	Gets the value of the <b>/RV</b> key.
<a href="#">FDFGetStatus</a>	Gets the value of the <b>/Status</b> key.
<a href="#">FDFGetValue</a>	Gets the value of a field ( <b>/V</b> key).
<a href="#">FDFGetFDFVersion</a>	Gets the current version of the FDF.
<a href="#">FDFNextFieldName</a>	Gets the next field name.

## Methods That Generate FDF Data

The following functions generate FDF data.

Method	Description
<a href="#">FDFAddDocJavaScript</a>	Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.
<a href="#">FDFAddTemplate</a>	Adds a template to an FDF file.
<a href="#">FDFSetAP</a>	Sets the appearance of a button field ( <b>/AP</b> key) from a PDF document.
<a href="#">FDFSetAPRef</a>	Sets a reference to a PDF document to use for the appearance of a field (one of the faces within the <b>/APRef</b> key).
<a href="#">FDFSetAS</a>	Sets the <b>/AS</b> key.
<a href="#">FDFSetFDFVersion</a>	Sets the FDF version.
<a href="#">FDFSetFile</a>	Sets the value of the <b>/F</b> key (string specification).
<a href="#">FDFSetFlags</a>	Sets the value of one of the <b>/Ff</b> , <b>/F</b> , <b>/SetFf</b> , <b>/ClrFf</b> , <b>/SetF</b> , or <b>/ClrF</b> flags of a field.
<a href="#">FDFSetGoToAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoTo</b> .
<a href="#">FDFSetGoToRAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoToR</b> .

Method	Description
<a href="#">FDFSetHideAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>Hide</b> .
<a href="#">FDFSetID</a>	Sets the value of one element in the <b>/ID</b> key.
<a href="#">FDFSetIF</a>	Sets the Icon Fit attribute for the appearance of a button field ( <b>/IF</b> key).
<a href="#">FDFSetImportDataAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>ImportData</b> .
<a href="#">FDFSetJavaScriptAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type JavaScript.
<a href="#">FDFSetNamedAction</a>	Sets <b>/A</b> or <b>/AA</b> key to a named action.
<a href="#">FDFSetOnImportJavaScript</a>	Adds a script to the FDF that will execute when it is imported.
<a href="#">FDFSetOpt</a>	Sets the value of one element in a field's <b>/Opt</b> key.
<a href="#">FDFSetResetFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>ResetForm</b> .
<a href="#">FDFSetRichValue</a>	Sets the value of the <b>/RV</b> key.
<a href="#">FDFSetStatus</a>	Sets the value of the <b>/Status</b> key.
<a href="#">FDFSetSubmitFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>SubmitForm</b> .
<a href="#">FDFSetTargetFrame</a>	Sets the value of the <b>/Target</b> key.
<a href="#">FDFSetURIAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>URI</b> .
<a href="#">FDFSetValue</a>	Sets the value of a field ( <b>/V</b> key).
<a href="#">FDFSetValues</a>	Sets the value of a field to an array.

# 5

## FDF Toolkit for Perl

The Perl FDF Toolkit is based on, and requires, the C component of the FDF Toolkit. Most of the calls work as in the C/C++ FDF Toolkit, except that they don't have the "FDF" prefix (for example, use **GetValue** instead of **FDFGetValue**).

The Unix and Windows versions of the Perl FDF Toolkit are nearly identical. This chapter explains the functions and the few differences that exist between the two implementations.

For sample usage of the Perl FDF Toolkit, please see the sample "EmployeeInfoDemo" (Windows or UNIX) at:

<http://partners.adobe.com/asn/developer/acrosdk/forms.html>

The Perl libraries for Windows were tested under Perl 5.6 (see <http://www.ActiveState.com>). The libraries are not upward compatible with later versions of Perl.

---

### Using the Perl FDF Toolkit

You must import the library into Perl's namespace via the **use** operator:

```
use Acrobat::FDF;
```

To install the Perl toolkit on Unix, execute the script `InstallAdvice.pl` through Perl. It gives further installation instructions based on the configuration of your system.

To install the Perl FDF Toolkit on Windows, create a directory called `Acrobat` in the `lib` folder of the Perl installation. Copy `FDF.pm` into the `Acrobat` directory.

In order to load the libraries, you will also need to install the `FDF.dll` and `FDFTK.dll` in the root of your `lib` folder.

### Using Perl in Unix

You must place these three things in a directory named `Acrobat`, which should be where Perl can reach them while running your script:

- **FDF.so**: the one shared object
- your script
- **FDF.pm**: the Perl module definition

The Perl version of the FDF Toolkit is supported for HP, Linux and Solaris versions of UNIX.

## Using Perl in Windows

You need to place two libraries, `FdfTk.dll` and `FDF.dll`, somewhere where Perl can reach them when running your script.

## Writing cgi-bin Applications Under IIS

To write cgi-bin applications under Internet Information Server (IIS):

1. Use `Regedt32.exe` to open  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ScriptMap`.
2. From the Edit menu, choose "Add Value".
3. Enter the value ".pl" for the "REG\_SZ" data type.
4. Use the String Editor to enter `d:\Perl\bin\Perl.exe %s %s`.
5. Restart the Web browser.

---

## Perl FDF Toolkit Methods

This section lists the FDF Toolkit methods for Perl. The name of each method links to a complete description of the method.

- [General Methods](#)
- [Methods that Parse FDF Data](#)
- [Methods That Generate FDF Data](#)

## Functions Only Available in the Perl FDF Toolkit

**newFromBuf** — For example,

```
$inFdf = Acrobat::FDF::newFromBuf ($buf);
```

## Functions Unavailable in the Perl FDF Toolkit

**FDFCreate** — use **new** instead. For example,

```
$inFdf = new Acrobat::FDF;
```

**FDFOpen** — use **new** instead. For example,

```
$inFDF = new Acrobat::FDF('in.fdf');
```

or

```
$inFdf=new Acrobat::FDF('-', $ENV{'CONTENT_LENGTH'});
```

**FDFSaveToStr** and **FDFSaveToBuf**— use **Save** instead. For example,

```
$outFdf=Save(' - ' );
```

**FDFClose** — there is no **Close** function. FDF objects are destroyed when they are no longer reachable by Perl.

## General Methods

TABLE 5.1

Method	Description
<a href="#">EmbedAndClose</a>	Embeds FDF in another document and then closes it.
<a href="#">GetVersion</a>	Gets the current version of the FDF Toolkit library.
<a href="#">RemoveItem</a>	Removes a key-value pair from the FDF data.
<a href="#">Save</a>	Writes out an FDF file.

## Methods that Parse FDF Data

The following methods parse FDF data.

TABLE 5.2

Method	Description
<a href="#">ExtractAppendSaves</a>	Extracts incremental changes submitted within an FDF, and makes a new file out of it.
<a href="#">ExtractAttachment</a>	Extracts an uploaded file and creates a new file out of it.
<a href="#">GetAP</a>	Gets the appearance of a field ( <b>/AP</b> ) and creates a PDF document out of it.
<a href="#">GetEncoding</a>	Gets the value of the FDF <b>/Encoding</b> key as a string.
<a href="#">GetFDFVersion</a>	Gets the current version of the FDF.
<a href="#">GetFile</a>	Gets the value of the <b>/F</b> key.
<a href="#">GetFlags</a>	Gets the flags of a field ( <b>/Ff</b> or <b>/F</b> keys).
<a href="#">GetID</a>	Gets the value of one element in the FDF's <b>/ID</b> key.
<a href="#">GetNthValue</a>	Gets an element from a field's value if it is an array.
<a href="#">GetOpt</a>	Gets the value of one element in a field's <b>/Opt</b> array.

TABLE 5.2

Method	Description
<code>GetRichValue</code>	Gets the value of the <b>/RV</b> key.
<code>GetStatus</code>	Gets the value of the <b>/Status</b> key.
<code>GetValue</code>	Gets the value of a field ( <b>/V</b> key).
<code>NextFieldName</code>	Gets the next field name.

## Methods That Generate FDF Data

The following methods generate FDF data.

TABLE 5.3

Method	Description
<code>AddDocJavaScript</code>	Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.
<code>AddTemplate</code>	Adds a template to an FDF file.
<code>new</code>	Creates a new FDF object.
<code>newFromBuf</code>	Creates a new FDF object from a buffer.
<code>newFromEmbedded</code>	Creates a new FDF object from FDF data embedded in a container.
<code>SetAP</code>	Sets the appearance of a button field ( <b>/AP</b> key) from a PDF document.
<code>SetAPRef</code>	Sets a reference to a PDF document to use for the appearance of a field (one of the faces within the <b>/APRef</b> key).
<code>SetAS</code>	Sets the <b>/AS</b> key.
<code>SetEncoding</code>	Sets the value of the <b>/Encoding</b> key.
<code>SetFDFVersion</code>	Sets the FDF version.
<code>SetFile</code>	Sets the value of the <b>/F</b> key (string specification).
<code>SetFileEx</code>	Sets the value of the <b>/F</b> key (complex file specification).

TABLE 5.3

Method	Description
<a href="#">SetFlags</a>	Sets the value of one of the <b>/Ff</b> , <b>/F</b> , <b>/SetFf</b> , <b>/ClrFf</b> , <b>/SetF</b> , or <b>/ClrF</b> flags of a field.
<a href="#">SetGoToAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoTo</b> .
<a href="#">SetGoToRAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>GoToR</b> .
<a href="#">SetHideAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>Hide</b> .
<a href="#">SetID</a>	Sets the value of one element in the <b>/ID</b> key.
<a href="#">SetIF</a>	Sets the Icon Fit attribute for the appearance of a button field ( <b>/IF</b> key).
<a href="#">SetImportDataAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>ImportData</b> .
<a href="#">SetJavaScriptAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>JavaScript</b> .
<a href="#">SetNamedAction</a>	Sets <b>/A</b> or <b>/AA</b> key to a named action.
<a href="#">SetOnImportJavaScript</a>	Adds a script to the FDF that will execute when it is imported.
<a href="#">SetOpt</a>	Sets the value of one element in a field's <b>/Opt</b> key.
<a href="#">SetResetByNameAction</a>	Sets <b>/A</b> or <b>/AA</b> key to action of type <b>ResetForm</b> using field names.
<a href="#">SetResetFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>ResetForm</b> .
<a href="#">SetRichValue</a>	Sets the value of the <b>/RV</b> key.
<a href="#">SetStatus</a>	Sets the value of the <b>/Status</b> key.
<a href="#">SetSubmitByNameAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>SubmitForm</b> using field names.
<a href="#">SetSubmitFormAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>SubmitForm</b> .
<a href="#">SetTargetFrame</a>	Sets the value of the <b>/Target</b> key.
<a href="#">SetURIAction</a>	Sets <b>/A</b> or <b>/AA</b> key to type action of <b>URI</b> .
<a href="#">SetValue</a>	Sets the value of a field ( <b>/V</b> key).
<a href="#">SetValues</a>	Sets the value of a field to an array.

The following example shows how to use some of the Perl methods to generate FDF:

**EXAMPLE 5.1    Using Perl Methods To Generate FDF Data**

```
/* Create a new FDF file-Example */
$inFdf = new Acrobat::FDF;

/*Open existing FDF file or read stdin-Example */
$inFDF = new Acrobat::FDF('in.fdf');

/* or */
$inFdf=new Acrobat::FDF('-', $ENV{'CONTENT_LENGTH'});
```



# 6

## FDF Toolkit For Java

The 6.0 release of the Java version of the FDF Toolkit is a platform-independent, 100% pure Java library which provides full support for features in the 6.0 release of the C/C++/Active X/Perl version.

The Java FDF Toolkit can be used on any platform with an installed Java Runtime Environment (JRE), version 1.2 or higher.

**NOTE:** For complete information on the Java classes and methods, see the HTML documentation provided with the Java FDF Toolkit. (The reference chapter of this document does not include Java.)

---

### Setup and Sample Usage of the Java FDF Toolkit

To use the 6.0 version of the Java FDF Toolkit, you should have version 1.2 or higher of the Java Development Kit installed on your machine.

The provided `jFdfTk.jar` file, which contains the classes needed to compile your applications with the FDF Toolkit Library should be added in your **CLASSPATH** during compilation and execution. You can either add the path where the `.jar` is located to your **CLASSPATH** environment variable, or provide the path as an argument to the compiler or VM (`javac` and `java`).

To see an illustration of the use of the different methods of the Java FDF Toolkit, you can look at a sample application provided in the `samples` directory.

---

### Changes to the API

Several changes have been made to the API since its version 4.05 release.

- Clients now use the **FDFDoc** class to parse, generate, modify, and output FDF content. The **FDFDoc** constructors parse or create FDF files, while the instance methods provide access to the majority of the Toolkit. The only static method, **GetVersion**, returns the current version of the Toolkit, which is 6.0.
- Classes in packages `com.adobe.fdf` and `com.adobe.fdf.exceptions` should be imported into applications using the Toolkit. The interfaces **FDFActionTrigger**, **FDFAppFace**, **FDFItem**, and **FDFScaleWhen** in the `com.adobe.fdf` package are used as containers of constants used in various methods in the Toolkit, and their use is documented in those methods.
- The **FDFTk** class has been deprecated, and no initialization/finalization routines need to be invoked.

- Please see the documentation of the **FDFDoc** class and its methods for further instructions on the use of the Java version of the FDF Toolkit.

---

## Other Useful Documentation

Documentation and tutorials for the Java programming language and API's can be found at Sun's Documentation & Training website. There, you will find information about installing and configuring the Java Runtime environment, as well as links to downloads of Java products.

# 7

## FDF Toolkit Reference

This document describes the data structures, methods, and callbacks used in the FDF Toolkit API. It has the following sections:

- [“Language Differences” on page 43](#)
- [“Data Types” on page 44](#)
- [“Methods” on page 51](#)
- [“Callbacks” on page 154](#)

---

### Language Differences

The FDF Toolkit is available in 4 languages:

- C/C++
- ActiveX
- Perl
- Java (not covered in this document, but has separate HTML documentation).

Methods and data structures are similar for all languages. This section describes some of the most important differences between the languages.

### Method Names

In C/C++ and ActiveX, most function names have an **FDF** prefix, while in Perl and Java they don't. Example: **FDFGetID** vs. **GetID**.

### Parameters

In C/C++, most methods take as their first parameter an **FDFDoc** which was returned by a call to [FDFCreate](#) or [FDFOpen](#). Perl and ActiveX do not take this parameter because it is the object whose method is called.

In the interest of space and consistency between languages, the description of this parameter is not repeated for all functions. It would look like this:

```
FDFErrc FDFClose (FDFDoc theFDF);
```

**theFDF** (C only)      Valid **FDFDoc** returned from [FDFCreate](#) or [FDFOpen](#).

In addition, C sometimes takes parameters such as buffer lengths which are not required in the other languages. When a parameter is applicable to a particular language, it is indicated in the parameter table.

## Errors and Return Values

Most methods can generate errors. See ["Error Codes"](#) for a complete list.

All C/C++ and Perl methods return an error code(of type **FDFErc** in C), unless the return value is explicitly specified otherwise. In ActiveX and Java, methods raise exceptions. The method descriptions list the possible errors that a method can return (with the exception of **FDFErcOK**, which means no error).

## Encoding

Several methods take string parameters that are described as being either in PDFDocEncoding or Unicode. This applies to C and Perl only; since ActiveX only uses Unicode, all encoding is done internally, and the client sees only Unicode. For this reason, the ActiveX API does not include the functions **GetEncoding** and **SetEncoding**.

---

## Data Types

This section lists a number of constants and data structures that are used in the FDF Toolkit methods.

Data structure/enum	Description
<a href="#">Error Codes</a>	Error codes.
<a href="#">FDFItem</a>	Key values.
<a href="#">FDFAppFace</a>	Face values.
<a href="#">FDFScaleWhen</a>	Scaling option values.
<a href="#">FDFActionTrigger</a>	Action Trigger values.
<a href="#">pdfFileSpec</a>	PDF file specification.

## Error Codes

In C/C++ and Perl most functions return error codes of type **FDFErc**. In ActiveX and Java, functions raise exceptions. The following table describes the errors that can occur. The descriptions of each function list the possible errors that can occur.

**NOTE:** In the ActiveX implementation, there is no corresponding string for the case where there is no error, or error number property 0. In this case, be sure to write your application comparing the number property to 0 rather than the string property to **FDFErcOK**.

Error	#	Description
<b>FDFErcOK</b>	0	The function returned successfully.
<b>FDFErcInternalError</b>	1	An internal FDF Library error occurred.
<b>FDFErcBadParameter</b>	2	One or more of the parameters passed to the function are invalid.
<b>FDFErcFileSysErr</b>	3	A file system error occurred, including "file not found".
<b>FDFErcBadFDF</b>	4	The FDF file being opened or parsed is invalid.
<b>FDFErcFieldNotFound</b>	5	The field whose name was passed in the parameter <b>fieldName</b> does not exist in the FDF file.
<b>FDFErcNoValue</b>	6	The field whose value was requested has no value.
<b>FDFErcEnumStopped</b>	7	Enumeration was stopped by <a href="#">FDFEnumValuesProc</a> by returning <b>false</b> . (C only).
<b>FDFErcCantInsertField</b>	8	The field whose name was passed in the parameter <b>fieldName</b> cannot be inserted into the FDF file. This might happen if you try to insert "a.b" into an FDF file that already has a field, such as "a.b.c". Conversely, you might try to insert "a.b.c" into a FDF file already containing "a.b".
<b>FDFErcNoOption</b>	9	The requested element in a field's <b>/Opt</b> key does not exist, or the field has no <b>/Opt</b> key.
<b>FDFErcNoFlags</b>	10	The field has no <b>/F</b> or <b>/Ff</b> keys.
<b>FDFErcBadPDF</b>	11	The PDF file passed as the parameter to <a href="#">FDFSetAP</a> is invalid, or does not contain <b>pageNum</b> .
<b>FDFErcBufTooShort</b>	12	The buffer passed as a parameter is too short for the length of the data that the function wants to return.

Error	#	Description
<b>FDFErcNoAP</b>	13	The field has no <b>/AP</b> key.
<b>FDFErcIncompatibleFDF</b>	14	An attempt to mix classic and template-based FDF files was made.
<b>FDFErcNoAppendSaves</b>	15	The FDF does not include a <b>/Differences</b> key.
<b>FDFErcValueIsArray</b>	16	The value of this field is an array Use <a href="#">FDFGetNthValue</a> .
<b>FDFErcEmbeddedFDFs</b>	17	The FDF you passed as a parameter is a container for one or more FDFs embedded within it. Use <a href="#">FDFOpenFromEmbedded</a> to gain access to each embedded FDF.
<b>FDFErcNoMoreFDFs</b>	18	Returned by <a href="#">FDFOpenFromEmbedded</a> when parameter <b>iWhich</b> >= the number of embedded FDFs (including the case when the passed FDF does not contain any embedded FDFs).
<b>FDFErcInvalidPassword</b>	19	Returned by <a href="#">FDFOpenFromEmbedded</a> when the embedded FDF is encrypted, and the correct password is not provided.
<b>FDFErcLast</b>	20	Reserved for future use.

## FDFItem

The following values are defined in the C enum **FDFItem**. They represent specific items in an FDF file.

C, ActiveX	Perl	Description
<b>FDFvalue</b>	<b>value</b>	Flag for the <b>/Value</b> key.
<b>FDFStatus</b>	<b>Status</b>	Flag for the <b>/Status</b> key.
<b>FDFFile</b>	<b>File</b>	Flag for the <b>/File</b> key.
<b>FDFID</b>	<b>ID</b>	Flag for the <b>/ID</b> key.
<b>FDDOpt</b>	<b>Opt</b>	Flag for the <b>/Opt</b> key.
<b>FDFff</b>	<b>Ff</b>	Flag for the <b>/Ff</b> key.
<b>FDFSetFf</b>	<b>SetFf</b>	Flag for the <b>/SetFf</b> key.

C, ActiveX	Perl	Description
<b>FDFClearFf</b>	<b>ClearFf</b>	Flag for the <b>/ClrFf</b> key.
<b>FDFFlags</b>	<b>Flags</b>	Flag for the <b>/Flags</b> key.
<b>FDFSetF</b>	<b>SetF</b>	Flag for the <b>/SetF</b> key.
<b>FDFClrF</b>	<b>ClrF</b>	Flag for the <b>/ClrF</b> key.
<b>FDFAP</b>	<b>AP</b>	Flag for the <b>/AP</b> key.
<b>FDFAS</b>	<b>AS</b>	Flag for the <b>/AS</b> key.
<b>FDFAction</b>	<b>Action</b>	Flag for the <b>/Action</b> key.
<b>FDFAA</b>	<b>AA</b>	Flag for the <b>/AA</b> key.
<b>FDFAPRef</b>	<b>APRef</b>	Flag for the <b>/APRef</b> key.
<b>FDFIF</b>	<b>IF</b>	Flag for the <b>/IF</b> key.
<b>FDFTargetFrame</b>	<b>TargetFrame</b>	Flag for the <b>/Target</b> key (see in <a href="#">FDFSetTargetFrame</a> ).
<b>FDFEncoding</b>	<b>Encoding</b>	Flag for the <b>/Encoding</b> key (see <a href="#">FDFGetEncoding</a> and <a href="#">FDFSetAS</a> ).
<b>FDFJavaScript</b>	<b>FDFJavaScript</b>	Flag for the <b>/JavaScript</b> key (see <a href="#">FDFAddDocJavaScript</a> ).
<b>FDFAppendSaves</b>	<b>AppendSaves</b>	Flag for the <b>/Differences</b> key (see <a href="#">FDFExtractAppendSaves</a> ).

### Used in Functions

[FDFGetFlags](#)

[FDFRemoveItem](#)

[FDFSetFlags](#)

[FDFSetTargetFrame](#)

[FDFGetEncoding](#)

[FDFSetAS](#)

[FDFExtractAppendSaves](#)

## FDFAppFace

The following values are defined in the C enum **FDFAppFace**.

TABLE 7.1

C, ActiveX	Perl	Description
<b>FDFNormalAP</b>	<b>NormalAP</b>	Normal <b>/AP</b> face ( <b>/N</b> key)
<b>FDFRolloverAP</b>	<b>RolloverAP</b>	Rollover <b>/AP</b> face ( <b>/R</b> key)
<b>FDFDownAP</b>	<b>DownAP</b>	Down <b>/AP</b> face ( <b>/D</b> key)

### Used in Functions

[FDFGetAP](#)

[FDFSetAP](#)

[FDFSetAPRef](#)

## FDFScaleWhen

The following values are defined in the C enum **FDFScaleWhen**.

TABLE 7.2

C, ActiveX	Perl	Description
<b>FDFAlways</b>	<b>Always</b>	Always scale the icon to fit within the rectangle of the annotation.
<b>FDFTooSmall</b>	<b>TooSmall</b>	Scale the icon to fit if it is smaller than the rectangle.
<b>FDFTooBig</b>	<b>TooBig</b>	Scale the icon to fit if it is larger than the rectangle.
<b>FDFNever</b>	<b>Never</b>	Never scale the icon to fit within the rectangle.

### Used in Functions

[FDFSetIF](#)

## FDFActionTrigger

These are the defined values for **FDFActionTrigger**:

- **FDFEnter**
- **FDFExit**
- **FDFDown**



- FDFUp
- FDFFormat
- FDFValidate
- FDFKeystroke
- FDFCalculate
- FDFOnFocus
- FDFOnBlur

#### Used in Functions

[FDFSetGoToAction](#)  
[FDFSetGoToRAction](#)  
[FDFSetHideAction](#)  
[FDFSetImportDataAction](#)  
[FDFSetJavaScriptAction](#)  
[FDFSetNamedAction](#)  
[FDFSetResetByNameAction](#)  
[FDFSetResetFormAction](#)  
[FDFSetSubmitByNameAction](#)  
[FDFSetSubmitFormAction](#)  
[FDFSetURIAction](#)

### pdfFileSpec

The elements of **pdfFileSpec** are all the possible components of a PDF file specification.

- char \*FS
- char \*F
- char \*Mac
- char \*DOS
- char \*Unix
- char \*ID[2]
- ASBool bVolatile

#### Used in Functions

[FDFAddTemplate](#)  
[FDFSetAP](#)

**FDFSetFileEx**

## Methods

### FDFAddDocJavaScript

#### (Perl) AddDocJavaScript

Adds a script to the FDF, which Acrobat then adds to the document-level scripts of a document, once the FDF is imported into it. Scripts are added to a dictionary associated with the **/JavaScript** key.

#### C Syntax

```
FDFErc FDFAddDocJavaScript (
    FDFDoc theFDF,
    const char* cScriptName,
    const char* cScript);
```

#### Perl Syntax

```
$outFDF->AddDocJavaScript ($cScriptName, $cScript)
```

#### ActiveX Syntax

```
Sub FDFAddDocJavaScript (cScriptName As String, cScript As String)
```

#### Parameters

**Table 1:**

<b>cScriptName</b>	Name of the script. Must be in either PDFDocEncoding or Unicode.
<b>cScript</b>	The script text. Must be in either PDFDocEncoding or Unicode. Hint: use "\r" as line separator within the script. Use "\t" for tabs.

#### Errors

[FDFErcIncompatibleFDF](#), [FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

#### Related Functions

[FDFSetOnImportJavaScript](#)

#### C Example

```
FDFErc erc = FDFAddDocJavaScript (theFDF, "PlusOne",
    "function PlusOne(x)\r{\r\treturn x + 1;\r}");
```

#### ActiveX Example

```
objFdf.FDFAddDocJavaScript "PlusOne", "function PlusOne(x)\r{\r\treturn x + 1;\r}"
```

## FDFAddTemplate

### (Perl) AddTemplate

Adds a template to the FDF file. There are two types of FDF:

- Classic — first introduced with Acrobat 3.0.
- Template-based — Upon import into Acrobat, directs the construction of a brand new PDF document from templates found inside specified PDF documents.

These two types of FDF are mutually exclusive:

- If you try to add templates to a classic FDF file, you will get **FDFErcIncompatibleFDF**
- Many of the calls in the FDF Toolkit (for example, **FDFSetFile**) are incompatible with a template-based FDF (an FDF for which **FDFAddTemplate** has been called), and will as well return **FDFErcIncompatibleFDF** if called with such an FDF.
- Other calls (for example, **FDFSetValue**) are OK with either kind of FDF. If called with a template-based FDF, they act on the most recently added template.

**NOTE:** Template functionality is not supported in Acrobat Reader. Therefore, if you create an Acrobat application that uses template functionality, a user who only has access to Acrobat Reader will not be able to use your application.

For more information on setting up templates, choose **Help -> Acrobat Help** in Acrobat.

### C Syntax

```
FDFErc FDFAddTemplate (FDFDoc theFDF, ASBool bNewPage,
    const pdfFileSpec fileSpec, const char* templateName,
    ASBool bRename);
```

### ActiveX Syntax

```
Sub FDFAddTemplate (bNewPage As Boolean, fileName As String,
    templateName As String, bRename As Boolean)
```

### Perl Syntax

```
$outFDF->AddTemplate ($bNewPage, $fileSpec, $templateName, $bRename)
```

### Parameters

**Table 2:**

<b>bNewPage</b>	<p>If <b>true</b>, the template is used to start a new page. If <b>false</b>, it is appended to the last page. If this is the first template added to the FDF data, this parameter is ignored.</p> <p>When a template-based FDF is used to construct a new PDF document, each page can be the result of overlaying multiple templates one after another.</p>
-----------------	--

Table 2:

<b>fileSpec</b>	<p>C/C++: A <b>pdfFileSpec</b> structure containing the file specification for the PDF where the template is stored. This is the value for the <b>/F</b> key within the <b>/TRef</b> entry.</p> <p>If <b>fileSpec</b> is <b>NULL</b>, when the FDF data gets imported into Acrobat, the template is expected to reside inside the PDF file currently being viewed, which will get replaced as the top most document with the new PDF constructed as a result of importing the FDF.</p> <p>If <b>fileSpec</b> is not <b>NULL</b>, fill in the fields of the <b>pdfFileSpec</b> data structure that you care about with null-terminated strings, and set the other fields to <b>NULL</b>. In the most common case where only the <b>/F</b> field is used, a convenience macro, <b>SIMPLE_FILESPEC</b>, is available.</p> <p><i>Perl, ActiveX</i>: This value is a simply string containing the pathname. The other fields of <b>pdfFileSpec</b> are not used.</p>
<b>templateName</b>	<p>String containing the name of the template within the PDF file specified by <b>fileSpec</b>.</p> <p><i>C, Perl</i>: Should be in PDFDocEncoding or Unicode.</p>
<b>bRename</b>	<p>If <b>bRename</b> is <b>false</b>, a <b>/Rename</b> key whose value is <b>false</b> is added, and renaming does not occur. If <b>bRename</b> is <b>true</b>, the <b>/Rename</b> key is not added because the default value is <b>true</b>; renaming occurs.</p> <p>The purpose of the <b>/Rename</b> key is to provide the flexibility of renaming fields as templates get spawned. This prevents potential conflicts between field names originating in different templates.</p> <p>When renaming is indicated (for example, because new values for them are included), Acrobat renames all fields in the template modified by the FDF file.</p> <p>The renaming scheme consists of prepending "P" + page # "template name_" + template # before the field name. For example, if a template called "flower" gets spawned into page 3, and this is the second template for that page in the FDF data, field "color" on that template gets renamed to "P3.flower_1.color."</p>

## Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcIncompatibleFDF**,  
**FDFErcEmbeddedFDFs**, **FDFErcInternalError**

**Related Functions****FDFSetValue****C Example**

```

void main()
{
    FDFErc retCode;
    FDFDoc outputFDF = NULL;
    pdfFileSpecRec filespec;
    SIMPLE_FILESPEC (filespec, "C:/application/example.pdf")

    /*
        Create a new FDF.
    */
    retCode = FDFCreate (&outputFDF);

    retCode = FDFAddTemplate (outputFDF, true, &filespec, "FillIn",
                              true);

    /*
        Now, take the pointer to the FDFDoc just created and
        add a new field.
    */

    retCode = FDFSetValue (outputFDF,
    "Date",
    "June 30, 2000",
                                false);
    retCode = FDFSetValue (outputFDF,
    "CustName",
    "John C Dulay",
                                false);
    retCode = FDFSetValue (outputFDF,
    "LicenseTo",
                                "Licensed To
                                YourCorp", false);

    /*
        Save the FDF data
    */
    retCode = FDFSave (outputFDF, "C:\\application\\example.fdf");

    /* Close the FDF File */
    retCode = FDFClose (outputFDF);
}

```

**ActiveX Example**

```
objFdf.FDFAddTemplate True, "MyTemplates.pdf", "logo", True
```

---

## FDFClose

**(C, ActiveX only)**

Frees resources used by the FDF file. Before exiting, client should call this function for each open FDF file. The client should not use the **FDFDoc** after the FDF file has been closed.

### C/C++ Syntax

```
FDFErc FDFClose (FDFDoc theFDF);
```

### ActiveX Syntax

```
Function FDFClose()
```

### Errors

**FDFErcBadParameter** (C only)

### Related Functions

[FDFCreate](#)

[FDFOpen](#)

[FDFFinalize](#)

### C Example

```
retCode = FDFClose (theFDF);
```

## FDFCreate

### (C, ActiveX only)

Creates a new FDF file. Once you are finished using the file created with this function, close it by using [FDFClose](#).

### C Syntax

```
FDFErc FDFCreate (FDFDoc* pTheFDF);
```

### ActiveX Syntax

```
Function FDFCreate() As FDFACXLib.FdfDoc
```

### Parameters

**Table 3:**

<b>pTheFDF</b>	A pointer to the <b>FDFDoc</b> . Other functions in the API use the pointer created here.
----------------	---

### Errors

[FDFErcBadParameter](#), [FDFErcInternalError](#)

### Related Functions

[FDFClose](#)

### C Example

```
FDFDoc theFDF;  
FDFErc fdfErC;  
  
fdfErC = FDFCreate (&theFDF);
```

### ActiveX Example

```
'The Dim statement does not apply to VBScript,  
'which only supports Variant  
Dim objFdf As FDFACXLib.FdfDoc  
  
'Create the FDF  
Set objFdf = FdfAcX.FDFCreate
```



## FDFEmbedAndClose

### (Perl) EmbedAndClose

Embeds an FDF inside another, then closes it. A container FDF can be a carrier for multiple embedded FDFs. The container FDF cannot be a templates-based FDF (if it is, the system returns [FDFErcIncompatibleFDF](#).) The FDF to embed cannot itself be a container (if it is, [FDFErcEmbeddedFDFs](#) is returned) or be a templates-based FDF ([FDFErcIncompatibleFDF](#) returns).

Each embedded FDF may optionally be password protected.

A container FDF can optionally include, besides the embedded FDFs, only an **/F** and/or **/ID** keys (see [FDFSetFile](#) and [FDFSetID](#).) Any other desired attributes belong in the embedded FDFs themselves. In turn, Acrobat ignores any **/F** or **/ID** keys in the embedded FDFs on FDF import, since it only looks for those keys in the container FDF.

You can use this call multiple times for the same container FDF. Each successive FDF is embedded after any previous FDFs already present in the container.

### C/C++ Syntax

```
FDFErc FDFEmbedAndClose (FDFDoc theContainerFDF,
                          FDFDoc theEmbeddedFDF,
                          const char* cPassword);
```

### ActiveX Syntax

```
Sub FDFEmbedAndClose (theEmbeddedFDF As FdfDoc, cPassword As String)
```

### Perl Syntax

```
FDFErc EmbedAndClose ($theEmbeddedFDF, $cPassword);
```

### Parameters

Table 4:

<b>theContainerFDF</b> (C only)	The container that holds one or more FDFs
<b>theEmbeddedFDF</b>	The FDF to embed. It is automatically closed with this call unless <a href="#">FDFErcBadParameter</a> is returned.
<b>cPassword</b>	If the embedded FDF should be stored encrypted, you need to provide the correct password. The password is a null-terminated array of bytes. If you pass an empty string (or NULL in C), the FDF will be embedded in an unencrypted state.

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

**Related Functions**[FDFClose](#)[FDFSetFile](#)[FDFSetID](#)**C Example**

```
FDFDoc theFDF, outerFDF;  
FDFErc erc = FDFOpen("c:\\temp\\in.fdf", 0, &theFDF);  
erc = FDFCreate (&outerFDF);  
erc = FDFEmbedAndClose (outerFDF, theFDF, "mypassword");  
erc = FDFSave (outerFDF, "c:\\temp\\out.fdf");  
FDFClose (outerFDF);
```

## FDFEnumValues

### (C only)

Enumerates the field names and values in the FDF file by calling a user-supplied procedure for each one. See [FDFNextFieldName](#) for an alternative way to obtain all the field names in the FDF file.

### Syntax

```
FDFErc FDFEnumValues (FDFDoc theFDF,
    FDFEnumValuesProc enumValuesProc, char* bufFldName,
    ASInt32 szBufFldName, char* bufFldVal,
    ASInt32 szBufFldVal, void* clientData,
    ASBool bSkipEmpty);
```

### Parameters

Table 5:

<b>enumValuesProc</b>	User-supplied callback of type <a href="#">FDFEnumValuesProc</a> that is called for each field value. Enumeration halts if <b>enumValuesProc</b> returns <b>false</b> .
<b>bufFldName</b>	Buffer that contains each field name as <b>enumValuesProc</b> gets called.
<b>szBufFldName</b>	The maximum number of bytes that can be written into <b>bufFldName</b> .
<b>bufFldVal</b>	Buffer that contains each field value as <b>enumValuesProc</b> gets called.
<b>szBufFldVal</b>	The maximum number of bytes that can be written into <b>bufFldVal</b> .
<b>clientData</b>	Pointer to user-supplied data passed to <b>enumValuesProc</b> each time it is called.
<b>bSkipEmpty</b>	If <b>true</b> , skips enumeration of fields that do not have a value. Otherwise, for such fields will call <b>enumValuesProc</b> with <b>bufFldVal</b> containing the empty string.

### Errors

[FDFErcBadParameter](#), [FDFErcEnumStopped](#), [FDFErcBadFDF](#),  
[FDFErcIncompatibleFDF](#), [FDFErcBufTooShort](#), [FDFErcInternalError](#)

### Related Functions

[FDFNextFieldName](#)

**C Example**

```
FDFDoc theFDF;  
FDFErc theErc;  
ASInt32 howMany = atoi(getenv"CONTENT_LENGTH");  
char cName[255];  
char cVal[255];  
  
theErc = FDFOpen("-", howMany, &theFDF);  
theErc = FDFEnumValues (theFDF, myFDFEnumValuesProc, cName,  
                        sizeof (cName), cVal, sizeof (cVal), NULL, false);
```

---

## FDFExtractAppendSaves

### (Perl) ExtractAppendSaves

Extracts the incremental changes to the PDF that were submitted inside the FDF, and creates a file out of them at the requested location. This is indicated by the **/Differences** key in the FDF.

**NOTE:** The append-saves are not removed from the FDF.

### C Syntax

```
FDFErc FDFExtractAppendSaves(  
    FDFDoc theFDF,  
    const char* cFileName);
```

### ActiveX Syntax

```
Sub FDFExtractAppendSaves (bstrDestFileName As String)
```

### Perl Syntax

```
$inFDF->ExtractAppendSaves($cFileName);
```

### Parameters

**Table 6:**

<b>cFileName</b>	Host-encoded string for the pathname of the file to create with the extracted bytes.
------------------	--

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcFileSysErr](#), [FDFErcNoAppendSaves](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetFDFVersion](#)

### C Example

```
theErc = FDFExtractAppendSaves (inFDF, "c:\\temp\\AppendSaves.tmp");
```

## FDFExtractAttachment

### (Perl) ExtractAttachment

Extracts the file uploaded by means of a "file selection" field, and creates a file out of it at the requested location.

**NOTE:** The attachment is not removed from the FDF.

### C Syntax

```
FDFErc FDFExtractAttachment(
    FDFDoc theFDF,
    const char* cFieldName,
    char* cFileName,
    ASInt32 nFileName
    ASBool bIsFilePath,
    char* cMimeType,
    ASInt32 nMimeTypeSize);
```

### ActiveX Syntax

```
Function FDFExtractAttachment(
    bstrFieldName As String,
    bstrFileName As String,
    bIsFilePath As Boolean) As String
```

### Perl Syntax

```
@value = $inFDF->ExtractAttachment($FieldName, $FileName, $bIsFilePath);
```

### Parameters

**Table 7:**

<b>cFieldName</b>	String representing the fully qualified name of the field.
<b>cFileName</b>	Host-encoded string for the pathname of the file to create with the extracted attachment. If <b>cFileName</b> is <b>NULL</b> or an empty string, Acrobat saves the file in the current folder, using the filename included with the attachment. If <b>bIsFilePath</b> is <b>true</b> and <b>cFileName</b> is not empty, Acrobat appends the filename included with the attachment to <b>cFileName</b> and save the attachment there. For example, if you pass <code>c:\\temp\\</code> on a Windows system, and the filename found in the attachment is <code>photo.jpg</code> , Acrobat will save the file to <code>c:\\temp\\photo.jpg</code> .
<b>nFileName</b> (C only)	If greater than zero, and <b>cFileName</b> is not <b>NULL</b> , <b>FDFExtractAttachment</b> fills <b>cFileName</b> with the full pathname (up to <b>nFileName</b> bytes) where the attachment was saved (only if <b>FDFExtractAttachment</b> returns <b>FDFErcOK</b> ).

Table 7:

<b>bIsFilePath</b>	If <b>true</b> and <b>cFileName</b> is not empty, Acrobat appends the filename included with the attachment to <b>cFileName</b> and saves the attachment there.
<b>cMimeType</b> (C only)	Filled by <b>FDFExtractAttachment</b> , if it returns <b>FDFErcOK</b> . buffer containing the MIME type of the file. Can be <b>NULL</b> , if this information is of no interest to the caller.
<b>nMimeTypeSize</b> (C only)	The maximum number of bytes that can be written into <b>cMimeType</b> . If <b>cMimeType</b> is <b>NULL</b> , then <b>bufSize</b> is <b>nMimeTypeSize</b> .

**Return Value**

C: error code.

ActiveX: string containing MIME type of extracted file.

Perl: An array of two string values. The first element of the array is the possibly modified FileName and the second element of the array is the MimeType.

**Errors**

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcIncompatibleFDF**,  
**FDFErcEmbeddedFDFs**, **FDFErcFieldNotFound**, **FDFErcFileSysErr**,  
**FDFErcNoValue**, **FDFErcInternalError**

**Related Functions**

**FDFGetFDFVersion**

**C Example**

```
FDFErc theErc;
char cFileName[255] = "/tmp/";
char cMIME[255];
theErc = FDFExtractAttachment (inFDF, "filename", cFileName, sizeof
(cFileName), true/*bIsFilePath*/, cMIME, sizeof (cMIME));
```

## FDFFinalize

*(C only; UNIX only; ignored on Windows)*

Finalizes the FDF Library. The client should call **FDFFinalize** after it is done with all other FDF Library calls. It should only be called once, by the last thread in the application.

### C Syntax

```
FDfErc FDFFinalize (void);
```

### Errors

None except for **FDfErcOK**

### Related Functions

**FDFFinalize**

**FDfClose**

### C Example

```
retCode = FDfClose (FdfOut)  
FDFFinalize();
```



## FDFGetAP

### (Perl) GetAP

Gets the appearance of a field (the value of one of the faces of the **/AP** key) and creates a PDF document out of it.

### C Syntax

```
FDFErc FDFGetAP (FDFDoc theFDF,
                 const char* fieldName, FDFAppFace whichFace,
                 const char* fileName);
```

### ActiveX Syntax

```
Sub FDFGetAP (fieldName As String,
              whichFace As Integer, fileName As String)
```

### Perl Syntax

```
$inFDF->GetAP($fieldName, $whichFace, $fileName);
```

### Parameters

<b>fieldName</b>	String representing the fully qualified name of the field.
<b>whichFace</b>	A value indicating which face of the <b>/AP</b> key is requested. See <a href="#">FDFAppFace</a> for possible values: <b>FDFNormalAP</b> , <b>FDFRolloverAP</b> , <b>FDFDownAP</b>
<b>fileName</b>	Host-encoded string for the pathname of the PDF file to create.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcIncompatibleFDF](#), [FDFErcFieldNotFound](#), [FDFErcFileSysErr](#),  
[FDFErcNoAP](#), [FDFErcInternalError](#)

### Related Functions

[FDFSetAP](#)

[FDFSetAPRef](#)

### C Example

```
/* Extract the picture of the accident, which is in the FDF submitted
from the client, who just filled out the insurance claim form */
theErc = FDFGetAP (theFDF, "Picture", FDFNormalAP, C:\\App\\back.pdf);
```

### ActiveX Example

```
Const FDFNormalAP    = 0
Const FDFRolloverAP  = 1
Const FDFDownAP      = 2
objFdf.FDFGetAP "my button", FDFNormalAP, "c:\\app\\norm_button.pdf"
```

## FDFGetEncoding

### (Perl) GetEncoding

#### (C, Perl only)

Gets the value of the FDF file's **/Encoding** key as a string. If this key exists, then all values (see [FDFEnumValues](#), [FDFGetValue](#), [FDFGetNthValue](#)), as well as options (see [FDFGetOpt](#)) within that FDF are encoded using the flavor of Host encoding indicated by that key. Possible encodings are: **Shift\_JIS**, **UHC**, **GBK**, **BigFive**. If the FDF contains no **/Encoding** key, then values and options are all in either PDFDocEncoding, or in Unicode if they cannot be represented in PDFDocEncoding (i.e. they contain double-byte characters).

### C Syntax

```
FDFErc FDFGetEncoding (FDFDoc theFDF,
                      char* buffer, ASInt32 bufSize,
                      ASInt32* nBytes);
```

### Perl Syntax

```
$str = $inFDF->GetEncoding();
```

### Parameters

Table 8:

<b>buffer</b> (C only)	(Filled by the function) The value of the FDF's <b>/Encoding</b> key as a string. If the FDF file contains no <b>/Encoding</b> key, <b>buffer</b> will contain the empty string. If you pass <b>NULL</b> , the function will return the length of <b>/Encoding</b> in <b>nBytes</b> .
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . You must pass at least the length + 1, since the routine adds a '\0' terminator. If <b>bufSize</b> is too small, the function returns <a href="#">FDFErcBufTooShort</a> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused.
<b>nBytes</b> (C only)	(Filled by the function) The length of the <b>/Encoding</b> key, not including the null terminator. If there is an error other than <a href="#">FDFErcBufTooShort</a> or <a href="#">FDFErcOK</a> , zero is returned.

### Return Value

C: Error code

Perl: String containing the encoding flavor.

### Errors

[FDFErcBadParameter](#), [FDFErcEmbeddedFDFs](#), [FDFErcIncompatibleFDF](#),  
[FDFErcBufTooShort](#), [FDFErcInternalError](#)

### Related Functions

[FDFSetEncoding](#)

**C Example**

```
char cBuf[255];  
ASInt32 nCount;  
FDFErc erc;  
erc = FDFGetEncoding (theFDF, cBuf, sizeof(cBuf), &nCount);
```

## FDFGetFDFVersion

### (Perl) GetFDFVersion

Returns a constant string with the version of the FDF file. Possible return values are 1.2, 1.3, 1.4, or 1.5 which correspond to the version of the *PDF Reference* where the various FDF features were first introduced.

### C Syntax

```
const char* FDFGetFDFVersion (FDFDoc theFDF);
```

### ActiveX Syntax

```
Function FDFGetFDFVersion() As String
```

### Perl Syntax

```
$str = $inFDF->GetFDFVersion();
```

### Return Value

A constant string representing the FDF version that includes all features actually used by the current FDF. Valid results are "1.2", "1.3", "1.4", or "1.5".

### Related Functions

[FDFSetFDFVersion](#)

### ActiveX Example

```
version = objFdf.FDFGetFDFVersion()
```

## FDFGetFile

### (Perl) GetFile

Gets the value of the FDF file's **/F** key, which points to the PDF form that this FDF data came from, or is meant for. It is assumed to be a PDF string, since that is what Acrobat produces when exporting FDF.

**NOTE:** An **/F** key generated by [FDFSetFileEx](#), which can be a more complex file specification, cannot be read back with **FDFGetFile** (**nBytes** would be set to zero).

### C Syntax

```
FDFErc FDFGetFile(
    FDFDoc theFDF,
    char* buffer,
    ASInt32 bufSize,
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFGetFile() As String
```

### Perl Syntax

```
$str = $inFDF->GetFile();
```

### Parameters

**Table 9:**

<b>buffer</b> (C only)	(Filled by the function) The value of the FDF's <b>/F</b> key as a Host-encoded string. If the FDF file contains no <b>/F</b> key, or the value of <b>/F</b> is not a PDF string, <b>buffer</b> will contain the empty string. If you pass <b>NULL</b> , the function will return the length of <b>/F</b> in <b>nBytes</b> .
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . You must pass at least the length + 1, since the routine adds a '\0' terminator. If <b>bufSize</b> is too small, the function returns <a href="#">FDFErcBufTooShort</a> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused.
<b>nBytes</b> (C only)	(Filled by the function) The length of the <b>/F</b> key, not including the null terminator. If there is an error other than <a href="#">FDFErcBufTooShort</a> or <a href="#">FDFErcOK</a> , zero is returned.

### Return Value

C: error code.

ActiveX, Perl: A string containing the value of the FDF's **/F** key.

**Errors**

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcBufTooShort](#),  
[FDFErcInternalError](#)

**Related Functions**

[FDFSetFile](#)

**C Example**

```
/* get our file string */  
retcode = FDFGetFile (theFDF, file, sizeof (file), &nBytes);
```

**ActiveX Example**

```
strFormFileName = objFdf.FDFGetFile
```

## FDFGetFlags

### (Perl) GetFlags

Gets the flags of a field (the value of the **/Ff** or **/F** keys).

**NOTE:** FDF data exported from an Acrobat form does not contain flags. However, this function can be useful when parsing FDF data produced some other way. For example, the FDF data may have previously been produced with the FDF Toolkit and saved to a file.

### C Syntax

```
FDFErc FDFGetFlags(
    FDFDoc theFDF,
    const char* fieldName,
    FDFItem whichFlags,
    ASUns32* pTheFlags);
```

### ActiveX Syntax

```
Function FDFGetFlags(
    fieldName As String,
    whichFlags As Integer) As Long
```

### Perl Syntax

```
$str = $inFDF->GetFlags($fieldName, $whichFlags);
```

### Parameters

**Table 10:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichFlags</b>	Value specifying which flag to get; must be either <b>FDFFf</b> or <b>FDFFlags</b> (see <a href="#">FDFItem</a> ).
<b>pTheFlags</b>	(Filled by the function) If <b>FDFGetFlags</b> returns <a href="#">FDFErcOK</a> , <b>pTheFlags</b> points to the requested flags.

### Return Value

C: Error code.

ActiveX: Long containing the flags for the specified field.

Perl: String containing the flags for the specified field.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcIncompatibleFDF](#), [FDFErcFieldNotFound](#), [FDFErcNoFlags](#),  
[FDFErcInternalError](#)

**Related Functions**[FDFSetFlags](#)**C Example**

```
ASUns32 flags;  
FDFErc retcode;  
retcode = FDFGetFlags (theFDF, "SubmitButton", FDFff, &flags);
```

**ActiveX Example**

```
Const FDFff      = 5  
Const FDFFlags = 8  
lngAnnotFlags = objFdf.FDFGetFlags ("employee.name.last", FDFFlags)
```



## FDFGetID

### (Perl) GetID

Gets the value of one element in the FDF's **/ID** key.

### C Syntax

```
FDFErc FDFGetID (FDFDoc theFDF,
                 ASInt32 nElemNum, const ASUns8* buffer,
                 ASInt32 bufSize, ASInt32* nBytes);
```

### ActiveX Syntax

```
FDFGetID (nElemNum As Short, buffer As String)
```

### Perl Syntax

```
@ids = $inFDF->GetID($nElemNum);
```

### Parameters

**Table 11:**

<b>nElemNum</b>	The element number to get in the FDF's <b>/ID</b> key, either: <ul style="list-style-type: none"> <li>● 0: the permanent ID</li> <li>● 1: the changing ID</li> </ul>
<b>buffer</b>	<i>(Filled by the function)</i> Buffer containing the requested ID element. If you pass <b>NULL</b> , the function returns the length of the ID element in <b>nBytes</b> .  <b>NOTE:</b> This is not a null-terminated string.
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . If <b>bufSize</b> is too small, the function returns <b>FDFErcBufTooShort</b> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused.
<b>nBytes</b> (C only)	<i>(Filled by the function)</i> The length of the ID element in bytes. If there is an error other than <b>FDFErcBufTooShort</b> or <b>FDFErcOK</b> , zero is returned.

### Return Value

*Perl:* Returns an array containing one element of the FDF's **/ID** key.

### Errors

**FDFErcBadParameter**, **FDFErcIncompatibleFDF**, **FDFErcBufTooShort**,  
**FDFErcInternalError**

### Related Functions

**FDFSetID**

**C Example**

```
ASUns8 buf[32];
ASInt32 nBytes;
FDFErc erc = FDFGetID (theFDF, 0 /*nElemNum*/, buf, sizeof (buf),
&nBytes);
```

---

## FDFGetNthValue

### (Perl) GetNthValue

Retrieves a specified element from a field's value (the ***N*** key) if it is an array. Use this call when **FDFGetValue** returns **FDFErcValueIsArray**.

You can use this function instead of **FDFGetValue**, even if the field's value is not an array, in which case Acrobat ignores the **iWhich** parameter.

### C Syntax

```
FDFErc FDFGetNthValue(  
    FDFDOC the FDF,  
    const char* fieldName,  
    ASInt32 iWhich,  
    char* buffer,  
    ASInt32 bufSize,  
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFGetNthValue (FieldName, iWhich) As String
```

### Perl Syntax

```
$str = $inFDF->GetNthValue($fieldName, $iWhich);
```

## Parameters

Table 12:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>iWhich</b>	Zero-based index of the desired value. Ignored if the field's value is not an array.
<b>buffer</b>	<i>(Filled by the function)</i> The value of the field. If the FDF includes an <b>/Encoding</b> key (see <a href="#">FDFGetEncoding</a> ), then the returned value is in that encoding. Otherwise, the value is in either PDFDocEncoding, or in Unicode if it contains double-byte characters. If you pass <b>NULL</b> , the function will return the length of the field in <b>nBytes</b> .
<b>bufSize</b>	The maximum number of bytes that can be written into <b>buffer</b> . You must pass at least the length + 2, since the routine adds a null terminator (which consists of 2 null bytes in the case of Unicode). If <b>bufSize</b> is too small, the function returns <a href="#">FDFErcBufTooShort</a> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused.
<b>nBytes</b>	<i>(Filled by the function)</i> The length of the value corresponding to <b>fieldName</b> , not including the null terminator. If there is an error other than <a href="#">FDFErcBufTooShort</a> or <a href="#">FDFErcOK</a> , zero is returned.

## Return Value

C: Error code.

ActiveX, Perl: A string containing the requested value.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcIncompatibleFDF](#),  
[FDFErcFieldNotFound](#), [FDFErcNoValue](#), [FDFErcBufTooShort](#),  
[FDFErcInternalError](#)

## Related Functions

[FDFGetValue](#)

## C Example

```
char cBuf[255];
ASInt32 nBytes;
FDFErc erc = FDFGetNthValue (theFDF, "mylistbox", 3/*iWhich*/, cBuf,
sizeof (cBuf), &nBytes);
```

---

## FDFGetOpt

### (Perl) GetOpt

Gets the value of one element in a field's **/Opt** array.

The **/Opt** key for a field of type listbox or combobox consists of an array. Each element in this array may be a single string, or an array of two strings. In the former case, this single string is used as both the item name, and the export value. In the latter case, the first string is the export value, and the second is the item name.

In the ActiveX version of this method, the **bFirstString** parameter indicates whether the first or second string is returned. Therefore, in ActiveX, if you want both the export value and the item name, you must call the method twice.

**NOTE:** An FDF file exported from an Acrobat form does not contain an **/Opt** field. However, **FDFGetOpt** can still be useful when parsing an FDF file produced some other way. For example, the FDF file may have been created with the FDF Toolkit and saved to a file.

### C Syntax

```
FDFErc FDFGetOpt(  
    FDFDoc theFDF,  
    const char* fieldName,  
    ASInt32 nElemNum,  
    char* buffer1,  
    char* buffer2,  
    ASInt32 bufSize,  
    ASInt32* nRet);
```

### ActiveX Syntax

```
Function FDFGetOpt(  
    fieldName As String,  
    nElemNum As Integer,  
    bFirstString As Boolean) As String
```

### Perl Syntax

```
@opts = $inFDF->GetOpt($fieldName, $nElemNum);
```

## Parameters

Table 13:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>nElemNum</b>	Integer representing the element number to get from the field's <b>/Opt</b> array. The first element in the <b>/Opt</b> array has an index of zero. If this parameter is -1, the function returns in <b>nRet</b> the number of elements in the field's <b>/Opt</b> array, and the parameters <b>buffer1</b> , <b>buffer2</b> , and <b>bufSize</b> remain unused.
<b>buffer1</b> (C only)	<i>(Filled by the function)</i> If the <b>/Opt</b> array element requested is itself composed of an array containing two strings, <b>buffer1</b> contains the first string (which is the export value). See <a href="#">FDFGetEncoding</a> for a discussion of encodings.  If the <b>/Opt</b> array element requested is a single string, <b>buffer1</b> contains that string (which is the item name). If <b>buffer1</b> is <b>NULL</b> , the function returns in <b>nRet</b> the number of bytes required.
<b>buffer2</b> (C only)	<i>(Filled by the function)</i> If the <b>/Opt</b> array element requested is itself composed of an array containing two strings, <b>buffer2</b> contains the second string (which is the item name). Otherwise, <b>buffer2</b> contains the empty string. If <b>buffer1</b> is <b>NULL</b> , <b>buffer2</b> remains unused.
<b>bufSize</b> (C only)	The maximum number of bytes written into <b>buffer1</b> and <b>buffer2</b> (which are assumed to be the same size). You must pass at least the length + 2 as the, since the routine adds a null terminator (which consists of 2 null bytes in the case of Unicode). If <b>buffer1</b> is <b>NULL</b> , <b>bufSize</b> remains unused.
<b>nRet</b> (C only)	<i>(Filled by the function)</i> If <b>nElemNum</b> is -1, the number of elements in the field's <b>/Opt</b> array is returned here.  Otherwise, <b>nret</b> will be the number of bytes copied into <b>buffer1</b> (excluding the terminating <b>NULL</b> ), if <b>buffer1</b> is not <b>NULL</b> , or the number of bytes that <b>buffer1</b> and <b>buffer2</b> need to be (if <b>buffer1</b> is <b>NULL</b> or <b>FDFGetOpt</b> returns <a href="#">FDFErcBufTooShort</a> ). If there is an error other than <a href="#">FDFErcBufTooShort</a> or <a href="#">FDFErcOK</a> , zero is returned.

Table 13:

<b>bFirstString</b> (ActiveX only)	Boolean that determines which of the two strings in the desired element should be returned. If <b>TRUE</b> , gets the first string; <b>FALSE</b> , the second string. If the element contains only one string, and <b>bFirstString</b> is <b>FALSE</b> , an empty string is returned.
---------------------------------------	---

**Return Value**

C: Error code.

ActiveX: one of the strings in the desired element.

Perl: An array of one or two strings from the desired element.

**Errors**

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcBufTooShort](#),  
[FDFErcCantInsertField](#), [FDFErcEmbeddedFDFs](#), [FDFErcFieldNotFound](#),  
[FDFErcIncompatibleFDF](#), [FDFErcInternalError](#), [FDFErcNoOption](#)

**Related Functions**

[FDFSetOpt](#)

**C Example**

```
retcode = FDFGetOpt (theFDF, "MyListBox", -1, NULL, NULL, 0, &nRet);
```

**ActiveX Example**

```
strOption = objFdf.FDFGetOpt("Credit Card", 2, True)
```

## FDFGetOptNumElem

**(ActiveX only)**

Returns an integer holding the number of elements in the field's **/Opt** array.

### ActiveX Syntax

```
Function FDFGetOptNumElem (fieldName As String) As Integer
```

### Parameters

**Table 14:**

<b>fieldName</b>	String representing the fully-qualified name of the field (for example, <b>employee.name.last</b> )
------------------	---

### Return Value

Integer holding the number of elements in the field's **/Opt** array.

### Errors

[FDFErcBadFDF](#), [FDFErcIncompatibleFDF](#), [FDFErcFieldNotFound](#),  
[FDFErcNoOption](#), [FDFErcInternalError](#), E\_OUTOFMEMORY

### Related Functions

[FDFGetOpt](#)

[FDFSetOpt](#)

### ActiveX Example

```
iNumElem = objFdf.FDFGetOptNumElem("Credit Card")
```



## FDFGetRichValue

### (Perl) GetRichValue

Gets the rich text value of a field as a string (the **/RV** key).

### C Syntax

```
FDFErc FDFGetRichValue(FDFDoc theFDF,
    const char* fieldName,
    char* buffer,
    ASInt32 bufSize,
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFGetRichValue(fieldName As String) As String
```

### Perl Syntax

```
$str = $inFDF->GetRichValue($fieldName);
```

### Parameters

**Table 15:**

<b>fieldName</b>	String representing the fully-qualified name of the field (for example, <b>employee.name.last</b> ).
<b>buffer</b> (C only)	(Filled by the function) Buffer containing the rich text value of the field. If the FDF includes an <b>/Encoding</b> key (see <b>FDFGetEncoding</b> ), then the returned value will be in that encoding. Otherwise, the value will be in either PDFDocEncoding, or in Unicode if it cannot be represented in PDFDocEncoding (i.e., it contains double-byte characters). If you pass <b>NULL</b> , the function returns the length of the value in <b>nBytes</b> .
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused. You must pass at least the length + 2 as the buffersize since the routine adds a null terminator (which consists of 2 null bytes in the case of Unicode).
<b>nBytes</b> (C only)	(Filled by the function) The function sets this value to zero before doing any work or error checking. If <b>buffer</b> is <b>NULL</b> and the function returns <b>FDFErcOK</b> , or <b>buffer</b> is not <b>NULL</b> and the function returns <b>FDFErcBufTooShort</b> , the number of bytes in the specified field's rich value is returned in <b>nBytes</b> . Otherwise, if the function returns <b>FDFErcOK</b> and <b>buffer</b> is not <b>NULL</b> , the number of bytes copied into it is returned, excluding the null terminator.

**Return Value**

C: error code

*Perl, ActiveX*: A string containing the value of the **/RV** key.

**Errors**

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcIncompatibleFDF](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcFieldNotFound](#), [FDFErcNoValue](#),  
[FDFErcValueIsArray](#), [FDFErcBufTooShort](#), [FDFErcInternalError](#)

**Related Functions**

[FDFSetRichValue](#)

## FDFGetStatus

### (Perl) GetStatus

Gets the value of the **/Status** key.

**NOTE:** FDF data exported from an Acrobat form does not contain a **/Status** key. However, the FDF file may have previously been produced with this library, had the **/Status** key set with **FDFSetStatus**, and saved as a file.

When an FDF file containing a **/Status** key is returned from a server after a submission, the value of this key is displayed in an alert box to the user.

### C Syntax

```
FDFErc FDFGetStatus(
    FDFDoc theFDF,
    char* buffer,
    ASInt32 bufSize,
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFGetStatus() As String
```

### Perl Syntax

```
$str = $inFDF->GetStatus();
```

### Parameters

**Table 16:**

<b>buffer</b> (C only)	(Filled by the function) The value of the FDF's <b>/Status</b> key as a Host-encoded string. If the FDF file contains no <b>/Status</b> key, <b>buffer</b> will contain the empty string. If you pass <b>NULL</b> , the function will return the length of <b>/Status</b> in <b>nBytes</b> .
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . It must be at least <b>nbytes</b> + 1, since the routine adds a <b>NULL</b> terminator. If <b>bufSize</b> is too small, the function returns <b>FDFErcBufTooShort</b> . If <b>buffer</b> is <b>NULL</b> , <b>bufSize</b> remains unused.
<b>nBytes</b> (C only)	(Filled by the function) The length of the <b>/Status</b> key, not including the null terminator. If there is an error other than <b>FDFErcBufTooShort</b> or <b>FDFErcOK</b> , zero is returned.

### Return Value

C: error code

Perl, ActiveX: A string containing the value of the **/Status** key.

**Errors**

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcBufTooShort](#),  
[FDFErcInternalError](#)

**Related Functions**

[FDFSetStatus](#)

**C Example**

```
retcode = FDFGetStatus (theFDF, statusText, sizeof (statusText),  
                        &nBytes);
```

**ActiveX Example**

```
strStatus = objFdf.FDFGetStatus
```

## FDFGetValue

### (Perl) GetValue

Gets the value of a field as a string (the **/V** key).

If the value of the **/V** key is an array, an **FDFErcValueIsArray** error is generated.

**FDFGetNthValue** can be called to get the values of the array.

### C Syntax

```
FDFErc FDFGetValue(
    FDFDoc theFDF,
    const char* fieldName,
    char* buffer,
    ASInt32 bufSize,
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFGetValue (fieldName As String) As String
```

### Perl Syntax

```
$str = $inFDF->GetValue($fieldName);
```

### Parameters

Table 17:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>buffer</b> (C only)	(Filled by the function) The value of the field. If the FDF includes an <b>/Encoding</b> key (see <b>FDFGetEncoding</b> ), then the returned value will be in that encoding. Otherwise, the value will be in either PDFDocEncoding, or in Unicode if it contains double-byte characters. If you pass <b>NULL</b> , the function will return the length of the value <b>nBytes</b> .
<b>bufSize</b> (C only)	The maximum number of bytes that can be written into <b>buffer</b> . You must pass at least the length + 2, since the routine adds a null terminator, which is 2 bytes for Unicode. If <b>bufSize</b> is too small, the function returns <b>FDFErcBufTooShort</b> . If <b>buffer</b> is <b>NULL</b> , then <b>bufSize</b> is unused.
<b>nBytes</b> (C only)	(Filled by the function) The length of the value, not including the null terminator. If there is an error other than <b>FDFErcBufTooShort</b> or <b>FDFErcOK</b> , zero is returned.

### Return Value

C: Error code

*ActiveX, Perl:* A string with the value of the `/V` key.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcIncompatibleFDF](#), [FDFErcFieldNotFound](#), [FDFErcNoValue](#),  
[FDFErcBufTooShort](#), [FDFErcInternalError](#)

[FDFErcValueIsArray](#): User should call [FDFGetNthValue](#) to get the values of the array.

## Related Functions

[FDFGetNthValue](#)

[FDFSetValue](#)

## C Example

```
FDFDoc theFDF = NULL;
FDFErc theErc;
char cBuf[128];

/* Open the FDF */
theErc = FDFOpen("-", howMany, &theFDF);

/* Get the FDF data */
theErc = FDFGetValue (theFDF, "Field1", cBuf, sizeof(cBuf), &nBytes);
```

## ActiveX Example

```
strLastName=objFdf.FDFGetValue("employee.name.last")
```

---

## FDFGetVersion

### (Perl) GetVersion

Gets the current version of the FDF Toolkit.

### C Syntax

```
const char* FDFGetVersion (void);
```

### ActiveX Syntax

```
Function FDFGetVersion() As String
```

### Perl Syntax

```
$str = $inFDF->GetVersion();
```

### Parameters

None

### Return Value

A string containing the current version of the FDF Toolkit.

### C Example

```
const char* verStr = FDFGetVersion();  
printf ("FDF Toolkit version: %s", verStr);
```

### ActiveX Example

```
strVersion = FdfAcX.FDFGetVersion
```

## FDFInitialize

*(C only; UNIX only; ignored on Windows)*

Initializes the FDF Library. The client should call **FDFInitialize** before using any other FDF Library calls. It should only be called once, by the first thread in the application.

### C Syntax

```
FDFErc FDFInitialize (void);
```

### Parameters

None

### Errors

None except for [FDFErcOK](#)

### Related Functions

[FDFFinalize](#)

### C Example

```
FDFInitialize();
```



## FDFNextFieldName

### (Perl) NextFieldName

Can be used to enumerate the field names in the FDF data. It returns the field name that comes after the one passed as a parameter (or the first field, if none is passed). The client can then use the returned field name as a parameter to many of the calls in this API.

**NOTE:** In C only, to enumerate all the field values in the FDF file, use [FDFEnumValues](#).)

### C Syntax

```
FDFErc FDFNextFieldName(
    FDFDoc theFDF,
    const char* fieldName,
    char* nextFldName,
    ASInt32 szNextFldName,
    ASInt32* nBytes);
```

### ActiveX Syntax

```
Function FDFNextFieldName (fieldName As String) As String
```

### Perl Syntax

```
$str = NextFieldName ($fieldName)
```

### Parameters

Table 18:

<b>fieldName</b>	String representing the fully-qualified name of the previous field (for example, <b>employee.name.last</b> ). If <b>fieldName</b> is <b>NULL</b> , the first field name in the FDF file is returned.
<b>nextFldName</b> (C only)	<i>(Filled by the function)</i> The next field name in the FDF file. If you have reached the end of the FDF file, <b>nextFldName</b> contains an empty string and <b>nBytes</b> is 0. If you pass <b>NULL</b> , the function will return the length of the field name in <b>nBytes</b> . You may pass the same buffer for the <b>fieldName</b> and <b>nextFldName</b> parameters.
<b>szNextFldName</b> (C only)	The maximum number of bytes that can be written into <b>nextFldName</b> (including the <b>NULL</b> terminator). If <b>nextFldName</b> is <b>NULL</b> , <b>szNextFldName</b> remains unused.
<b>nBytes</b> (C only)	<i>(Filled by the function)</i> The length of <b>nextFldName</b> , not including the null terminator. If there is an error other than <a href="#">FDFErcBufTooShort</a> or <a href="#">FDFErcOK</a> , zero is returned.

### Return Value

C: error code

*ActiveX, Perl*: string containing the next field name.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcIncompatibleFDF](#), [FDFErcFieldNotFound](#), [FDFErcBufTooShort](#),  
[FDFErcInternalError](#)

## Related Functions

[FDFEnumValues](#)

## C Example

```
FDFDoc theFDF;  
FDFErc fdfErC;  
char cBuf[255];  
ASInt32 nBytes;  
  
fdfErC = FDFNextFieldName (theFDF, NULL, cBuf, sizeof(cBuf), &nBytes);
```

## ActiveX Example

```
strFirstField = objFdf.FDFNextFieldName("")  
strSecondField = objFdf.FDFNextFieldName (strFirstField)
```

## FDFOpen

### (C only)

Opens an existing FDF file. Once you are finished using the open file, close it with the [FDFClose](#) function.

### C Syntax

```
FDFErc FDFOpen (
    const char* fileName,
    ASInt32 howMany,
    FDFDoc* pTheFDF);
```

### Parameters

**Table 19:**

<b>fileName</b>	Complete pathname (in Host encoding) or "-" to read from <b>stdin</b> .
<b>howMany</b>	If <b>fileName</b> is <b>stdin</b> , the number of characters to read; otherwise ignored. In a Web server environment, this is available as the value of the <b>CONTENT_LENGTH</b> environment variable. It's needed because in some servers executing <b>cgi-bin</b> scripts, the script hangs if it tries to read <b>stdin</b> until an EOF is reached.
<b>pTheFDF</b>	A pointer to an <b>FDFDoc</b> , which is referenced by most other calls in the API.

### Errors

[FDFErcBadParameter](#), [FDFErcFileSysErr](#), [FDFErcBadFDF](#),  
[FDFErcInternalError](#)

### Related Functions

[FDFClose](#)

### C Examples

```
/* The following is an example of opening an FDF file with a
   fully qualified pathname as the input. */
```

```
FDFDoc theFDF;
fdFErC = FDFOpen("C:\\InetPub\\wwwroot\\myapp\\in.fdf", 0, &theFDF);
```

or

```
/* The following is an example of opening an FDF file from stdin. */
```

```
ASInt32 howMany = atoi(getenv("CONTENT_LENGTH"));
fdFErC = FDFOpen("-", howMany, &theFDF);
```

## FDFOpenFromBuf

### (ActiveX only)

Opens FDF data from a buffer.

**FDFOpenFromBuf** takes an array of unsigned bytes (Variant of type `VT_ARRAY | VT_UI1`) which holds the contents of the FDF data, and returns an object of type **FDFACXLib.FdfDoc**. Client applications should call **FDFClose** when the FDF returned by **FDFOpenFromEmbedded** is no longer needed.

**NOTE:** You need to have a minimum of Windows NT Service Pack 3 (SP3) installed on your Windows NT Server v4.0 system for this function to work. SP3 includes version 1.0b of ASP, which adds `Request.BinaryRead` and `Request.TotalBytes`.

**NOTE:** The equivalent Perl function is **newFromBuf**.

### ActiveX Syntax

```
Function FDFOpenFromBuf (varFdfData As Array) As FDFACXLib.FdfDoc
```

### Parameters

**Table 20:**

<b>varFdfData</b>	An array of bytes holding the contents of the FDF data.
-------------------	---

### Return Value

An object of type **FDFACXLib.FdfDoc**.

### Errors

**FDFErcFileSysErr**, **FDFErcBadFDF**, **FDFErcInternalError**, **E\_OUTOFMEMORY**

### Related Functions

**FDFClose**

**FDFOpenFromFile**

**FDFOpenFromStr**

**newFromBuf**

### ActiveX Example

```
'The Dim statement does not apply to VBScript,
'which only supports Variant
Dim objFdf As FDFACXLib.FdfDoc

Set objFdf = FdfAcX.FDFOpenFromBuf _
(Request.BinaryRead(Request.TotalBytes))
```

## FDFOpenFromEmbedded

(C, ActiveX only)

Use this call when you discover that your FDF is actually only a container for one or more "real" FDFs (that is, when any of the calls return the error code [FDFErcEmbeddedFDFs](#)). When the FDF returned by this function is no longer needed, clients should call [FDFClose](#), in addition to calling [FDFClose](#) on the container.

**NOTE:** For Perl, see [newFromEmbedded](#).

### C Syntax

```
FDFErc FDFOpenFromEmbedded (
    FDFDoc the ContainerFDF,
    ASInt32 iWhich,
    const char* cPassword,
    FDFDoc* pTheEmbeddedFDF);
```

### ActiveX Syntax

```
Function FDFOpenFromEmbedded (iWhich As Integer,
    cPassword As String)
    As FDFACXLib.FdfDoc
```

### Parameters

Table 21:

<b>theContainerFDF</b> (C only)	A container that holds one or more FDFs
<b>iWhich</b>	Zero-based index of the embedded FDF that should be opened.
<b>cPassword</b>	If the embedded FDF is encrypted, you need to pass in the correct password to open the file. If it is not encrypted, the system ignores this parameter. It consists of a null-terminated string.
<b>pTheEmbeddedFDF</b> (C only)	If this function returns <a href="#">FDFErcOK</a> , <b>pTheEmbeddedFDF</b> points to the opened <b>FDFDoc</b> , which can then be used for other API calls.

### Return Value

C: Error code.  
ActiveX: An object of type **FDFACXLib.FdfDoc**.

**Errors**

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcBadFDF](#),  
[FDFErcInvalidPassword](#), [FDFErcNoMoreFDFs](#), [FDFErcInternalError](#)

**Related Functions**

[FDFOpen](#)  
[FDFOpenFromBuf](#)  
[FDFOpenFromFile](#)  
[FDFOpenFromStr](#)  
[newFromEmbedded](#)

**C Example**

```
FDFDoc theEmbeddedFDF;  
FDFErc erc = FDFOpenFromEmbedded (theContainerFDF, 0/*iWhich*/,  
0 /*cPassword*/, &theEmbeddedFDF);
```

## FDFOpenFromFile

(ActiveX only)

Opens an existing FDF file and returns an object of type **FDFACXLib.FdfDoc**.

The client should call the function **FDFClose** on the **FDFDoc** object when the **FDFDoc** is no longer needed.

### ActiveX Syntax

```
Function FDFOpenFromFile (bstrFileName As String) As FDFACXLib.FdfDoc
```

### Parameters

Table 22:

<b>bstrFileName</b>	String containing the pathname for the FDF file to be opened.
---------------------	---

### Return Value

An object of type **FDFACXLib.FdfDoc**.

### Errors

**FDFErcFileSysErr**, **FDFErcBadFDF**, **FDFErcInternalError**

### Related Functions

- FDFClose**
- FDFOpenFromBuf**
- FDFOpenFromStr**

## FDFOpenFromStr

### (ActiveX only)

Opens FDF data from a string.

**FDFOpenFromStr** takes a string which holds the contents of FDF data, and returns an object of type **FDFACXLib.FdfDoc**. The client should call the function **FDFClose** on the **FdfDoc** object when the **FdfDoc** is no longer needed.

**NOTE:** The preferred way to open FDF data from a stream of bytes is by using **FDFOpenFromBuf**, because FDF contains binary data. However, **FDFOpenFromStr** is provided for the benefit of some environments that do not support the data type *Variant of type VT\_ARRAY | VT\_UI1*.

### Syntax

```
Function FDFOpenFromStr (bstrFdfData As String) As FDFACXLib.FdfDoc
```

### Parameters

Table 23:

<b>bstrFdfData</b>	String containing the contents of the FDF data.
--------------------	---

### Return Value

An object of type **FDFACXLib.FdfDoc**.

### Exceptions

**FDFErcFileSysErr**, **FDFErcBadFDF**, **FDFErcInternalError**

### Related Functions

**FDFNextFieldName**

**FDFOpenFromBuf**

**FDFOpenFromFile**



## FDFRegisterThreadsafeCallbacks

### (C only)

(UNIX only; ignored on Windows) Registers the callbacks to thread-safe operations for lock, unlock, and destroy. UNIX systems vary in ways of doing multi-threading. How it is supported is dependent on modules installed at the kernel level.

If you call this function by passing **NULL** for all the arguments, you get the default OS mutex. If you call it with only one or two **NULL** arguments, thread safety is created using just the methods you do supply.

### C Syntax

```
FDFErc FDFRegisterThreadsafeCallbacks (
    ThreadsafeCallback lockProc,
    ThreadsafeCallback unlockProc,
    ThreadsafeCallback destroyProc,
    void *clientData);
```

### Parameters

Table 24:

<b>lockProc</b>	<b>ThreadsafeCallback</b> used for locking the mutex operation.
<b>unlockProc</b>	<b>ThreadsafeCallback</b> used for unlocking the mutex operation.
<b>detroyProc</b>	<b>ThreadsafeCallback</b> used for destroying the mutex operation.
<b>clientData</b>	Data passed through to each of the operations. Typically the data is initialized by the user before calling <b>FDFRegisterThreadsafeCallbacks</b> by a call to a mutex initialize operation like Sun Solaris' <b>mutex_init</b> ).

### Errors

None except for **FDFErcOK**.

### Related Functions

**FDFInitialize**

**FDFFinalize**

### C Example

```
/* Definitions needed only on Unix systems for multithreading. */
#ifdef SOLARIS
static mutex_t mutexObj;
mutex_init( &mutexObj, NULL, NULL );
FDFRegisterThreadsafeCallbacks(
    (ThreadsafeCallback)mutex_lock,
```

```
        (ThreadsafeCallback)mutex_unlock,  
        (ThreadsafeCallback)mutex_destroy,  
        &mutexObj );  
#endif
```

## FDFRemoveItem

### (Perl) RemoveItem

Removes a key-value pair from the FDF data.

### C Syntax

```
FDFErc FDFRemoveItem(
    FDFDoc theFDF,
    const char* fieldName,
    FDFItem whichItem);
```

### ActiveX Syntax

```
Sub FDFRemoveItem(fieldName As String, whichItem As Integer)
```

### Perl Syntax

```
$inFDF->RemoveItem($fieldName, $whichItem);
```

### Parameters

**Table 25:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ) from which the key-value pair is to be removed. If the item to be removed is not specific to one particular field ( <b>FDFStatus</b> , <b>FDFFile</b> , <b>FDFID</b> , <b>FDFTargetFrame</b> , <b>FDFEncoding</b> , <b>FDFJavaScript</b> , <b>FDFAppendSaves</b> ), pass <b>NULL</b> .
<b>whichItem</b>	An value identifying the item to be removed. If the item to be removed is not present, no error occurs (the function returns <b>FDFErcOK</b> ). See <b>FDFItem</b> for the list of possible values.

### Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcEmbeddedFDFs**,  
**FDFErcIncompatibleFDF**, **FDFErcFieldNotFound**, **FDFErcInternalError**

### C Example

```
retcode = FDFRemoveItem (theFDF, 0/*whichItem*/, FDFFile);
```

### ActiveX Example

```
Const FDFFile = 2
theFdf.FDFRemoveItem "", FDFFile
```

## FDFSave

### (Perl) Save

(C, Perl only)

Writes out an FDF file.

**NOTE:** The FDF cannot be saved to a file that was opened with [FDFOpen](#) and hasn't yet been closed with [FDFClose](#).

### C Syntax

```
FDFErc FDFSave (FDFDoc theFDF, const char* fileName);
```

### Perl Syntax

```
$inFDF->Save($fileName);
```

### Parameters

**Table 26:**

<code>fileName</code>	Complete pathname (in Host encoding), or "-" to write to <code>stdout</code> .
-----------------------	--

### Errors

[FDFErcBadParameter](#), [FDFErcFileSysErr](#), [FDFErcInternalError](#)

### Related Functions

[FDFClose](#)

[FDFOpen](#)

### C Example

```
retCode = FDFSave(theFDF, "d:\\TEMP\\MyData.fdf");
```

---

## FDFSavetoBuf

**(ActiveX only)**

Returns an array of unsigned bytes (Variant of type *VT\_ARRAY | VT\_UI1*) containing the FDF data.

### ActiveX Syntax

```
Function FDFSavetoBuf() As Array
```

### Parameters

None

### Return Value

An array of unsigned bytes containing the FDF data.

### Errors

[FDFErcFileSysErr](#), [FDFErcInternalError](#)

### Related Functions

[FDFSavetoFile](#)

[FDFSavetoStr](#)

### ActiveX Example

```
Response.ContentType = "application/Vnd.fdf"  
Response.BinaryWrite theFdf.FDFSavetoBuf
```

## FDFSavetoFile

**(ActiveX only)**

Saves FDF data to a file.

### ActiveX Syntax

```
Sub FDFSavetoFile (fileName As String)
```

### Parameters

**Table 27:**

<b>fileName</b>	String containing the pathname where the FDF file should be saved. The FDF data cannot be saved to a file that was opened with <a href="#">FDFOpenFromFile</a> and has not yet been closed with <a href="#">FDFClose</a> .
-----------------	--

### Errors

[FDFErcFileSysErr](#), [FDFErcInternalError](#)

### Related Functions

[FDFOpenFromFile](#)

[FDFClose](#)

### ActiveX Example

```
objFdf.FDFSavetoFile "d:\out.fdf"
```

---

## FDFSaveToStr

**(ActiveX only)**

Returns a string containing the FDF data.

**NOTE:** The preferred way to get the FDF data as a stream of bytes is by using [FDFSave](#), because the FDF contains binary data. **FDFSaveToStr** is provided for the benefit of some environments that do not support a return value of type *Variant of type VT\_ARRAY|VT\_UI1*.

### ActiveX Syntax

```
Function FDFSaveToStr() As String
```

### Parameters

None

### Return Value

A string containing the FDF data.

### Errors

[FDFErcFileSysErr](#), [FDFErcInternalError](#), E\_OUTOFMEMORY

### Related Functions

[FDFSave](#)

[FDFSavetoFile](#)

### ActiveX Example

```
Dim strFDF As String  
strFDF = objFdf.FDFSaveToStr
```

## FDFSetAP

### (Perl) SetAP

Sets the appearance of a button field (the value of one of the faces of the **/AP** key) from a PDF document. If the field does not exist in the FDF file, it is created. If it does, the requested face in the **/AP** key is replaced. If the FDF file is template-based (**FDFAddTemplate** has been called), this function acts on the most recently added template.

Both **FDFSetAP** and **FDFSetAPRef** work with buttons only. The difference between the two is that **FDFSetAP** resolves the PDF on the server, whereas **FDFSetAPRef** resolves it on the client. In a Web application, when calling **FDFSetAP** you typically pass as a parameter the pathname for the PDF file that contains the **/AP** key you want. If you call **FDFSetAPRef**, you must pass the URL of the PDF file that contains the **/AP** key you want.

Acrobat will only import an **/AP** key from the FDF file into fields of type "Button". Also, the new imported **/AP** will not show if the "Layout" for the button is of type "Text only".

Once the FDF file containing the new **/AP** is imported into the Acrobat Form, if the picture looks too small inside the button field, with too much white space around it, you may want to crop (using Acrobat) the PDF page used as the source of the **/AP** (the one identified by parameters **fileName** and **pageNum**).

### C Syntax

```
FDFErc FDFSetAP(
    FDFDoc theFDF,
    const char* fieldName,
    FDFAppFace whichFace,
    const char* subAP,
    const char* fileName,
    ASInt32 pageNum);
```

### ActiveX Syntax

```
Sub FDFSetAP(
    fieldName As String,
    whichFace As Integer,
    fileName As String,
    pageNum As Integer)
```

### Perl Syntax

```
$outFDF->SetAP($fieldName, $whichFace, $subAP, $fileName, $page);
```



## Parameters

Table 28:

<b>fieldName</b>	Host-encoded string representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichFace</b>	An value indicating which face is to be set. Possible values are <b>FDFNormalAP</b> , <b>FDFDownAP</b> , and <b>FDFRolloverAP</b> (see <a href="#">FDFAppFace</a> ).  <b>NOTE:</b> Only the <b>FDFNormalAP</b> will be imported (and <b>FDFDownAP</b> , <b>FDFRolloverAP</b> will be ignored) unless the button that the <b>/AP</b> is being imported into has a "Highlight" of type "Push".
<b>subAP</b>	String to indicate which sub-appearance is to be set. Currently, Acrobat does not import sub-appearances; therefore, you should pass <b>NULL</b> for this parameter.
<b>fileName</b>	String for the pathname of the PDF file representing the button's appearance.
<b>pageNum</b>	Page within the PDF file to be used for the appearance. The first page for this parameter begins with 1, not 0.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcBadPDF](#), [FDFErcEmbeddedFDFs](#), [FDFErcFileSysErr](#), [FDFErcCantInsertField](#), [FDFErcInternalError](#)

## Related Functions

[FDFGetAP](#)

[FDFSetAPRef](#)

## C Example

```
/* Set the down appearance of a button */

retCode = FDFSetAP (theFDF, "MyButtonField", FDFDownAP, NULL,
                    "MyDOC.pdf", 5);
```

## ActiveX Example

```
Const FDFNormalAP    = 0
Const FDFRolloverAP  = 1
Const FDFDownAP      = 2

theFdf.FDFSetAP "Button", FDFNormalAP, "", "d:\app\nrm_button.pdf", 1
```

## FDFSetAPRef

### (Perl) SetAPRef

Sets a reference to the PDF document to use for the appearance of a field (the value of one of the faces within the **/APRef** key). If the field does not exist in the FDF file, it is created. If it does, the requested face in the **/APRef** is replaced.

If the FDF file is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

Both [FDFSetAP](#) and **FDFSetAPRef** work with buttons only. The difference between the two is that [FDFSetAP](#) resolves the PDF on the server, whereas **FDFSetAPRef** resolves it on the client. In a Web application, when calling [FDFSetAP](#) you typically pass as a parameter the pathname for the PDF file that contains the **/AP** you want. If you call **FDFSetAPRef**, you must pass the URL of the PDF file that contains the **/AP** you want.

**/AP** and **/APRef** are mutually exclusive. If both appear inside an FDF, **/AP** takes precedence and **/APRef** is ignored. Note that while **/AP** inside an FDF corresponds to the **/AP** for a button field in a form, **/APRef** is an FDF-only key, and is resolved to an **/AP** during FDF import. In other words, **/APRef** never shows up inside PDF. The value of **/F** within **/APRef** can be a URL if the PDF document that the FDF's imported into (if any) is being viewed inside a Web browser. In addition, if the FDF file will in fact be imported into an existing PDF (as opposed to being used to construct a brand new document), the value of **/F** can be a relative file specification. **/APRef** does not need to contain an **/F** if the appearance resides (presumably as an invisible template) in the PDF file that the FDF file is imported into.

### C Syntax

```
FDFErc FDFSetAPRef (
    FDFDoc theFDF,
    const char* fieldName,
    FDFAppFace whichFace,
    const pdfFileSpec fileSpec,
    const char* templateName);
```

### ActiveX Syntax

```
Sub FDFSetAPRef (
    fieldName As String,
    whichFace As Integer,
    fileSpec As String,
    templateName As String)
```

### Perl Syntax

```
$outFDF->SetAPRef($fieldName, $whichFace, $fileSpec, $templateName);
```

## Parameters

Table 29:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichFace</b>	Value indicating which face to set. See <a href="#">FDFAppFace</a> for possible values.
<b>fileSpec</b>	The value for the <b>/F</b> key, which is the file specification for the PDF file that contains the page to be used for the appearance. <b>fileSpec</b> can be <b>NULL</b> . In this case, when the FDF file gets imported into an Acrobat Form, the template is expected to reside inside that PDF file. If <b>fileSpec</b> is not <b>NULL</b> , it is only necessary to fill-in the fields of the <b>pdfFileSpec</b> data structure that you care about with Strings, and set the others to <b>NULL</b> . The most common case is to only use the <b>/F</b> field, and to this end a convenience macro <b>SIMPLE_FILESPEC</b> is available. <i>ActiveX, Perl:</i> value must be a string. May be an empty string if no <b>/F</b> is needed.
<b>templateName</b>	Name of the page (template) within the PDF file specified by <b>fileSpec</b> or <b>fileName</b> , to be used for the appearance.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

## Related Functions

[FDFSetAP](#)

## C Example

```
pdfFileSpecRec filespec;
SIMPLE_FILESPEC(filespec, "http://myserver/MyTemplates.pdf");
theErc = FDFSetAPRef(theFDF, "button", FDFNormalAP, &filespec,
                    "template1");
```

## ActiveX Example

```
Const FDFNormalAP = 0
Const FDFRolloverAP = 1
Const FDFDownAP = 2
objFdf.FDFSetAPRef "Button", FDFNormalAP, "MyTemplates.pdf", "logo"
```

## FDFSetAS

### (Perl) SetAS

Sets the value of the **/AS** key for a field. If the field not yet exist in the FDF, Acrobat creates it. If it does exist, and already has an **/AS** key, Acrobat replaces its old value.

**NOTE:** Do not use **SetAS** for fields of type *checkbox* or *radiobutton*. Use **FDFSetValue** instead.

When the value (**/V**) of a field of one of these two type changes (to a valid value, for instance either **OFF** or a value that was entered as the export value when defining the properties of the field), Acrobat automatically sets the **/AS** key to the correct value.

You can use this function for fields of type button, but only if they have subfaces defined. Acrobat does not have a user interface that allows you to define subfaces for buttons, but you can accomplish this using **pdfmark** instead. See <http://partners.adobe.com/asn/developer/acrosdk/docs.html>.

### C Syntax

```
FDFErc FDFSetAS(FDFDoc theFDF, const char* fieldName,
                const char* newAS);
```

### ActiveX Syntax

```
Sub FDFSetAS(
    fieldName As String,
    newAS As String)
```

### Perl Syntax

```
$outFDF->SetAS($fieldName, $newAS);
```

### Parameters

Table 30:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>newAS</b>	String in <b>PDFDocEncoding</b> that will be converted to a PDF Name to use as the new value of <b>/AS</b> .

### Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**,  
**FDFErcIncompatibleFDF**, **FDFErcEmbeddedFDFs**,  
**FDFErcInternalError**

### Related Functions

**FDFSetValue**

## FDFSetEncoding

### (Perl) SetEncoding

(C, Perl only)

Sets the value of the FDF file's **/Encoding** key. If the FDF file already has an **/Encoding** key, its old value is replaced. If you are setting this key in the FDF, it is assumed that all values (see [FDFSetValue](#), [FDFSetValues](#)) and options (see [FDFSetOpt](#)) in the FDF will follow that encoding.

### C Syntax

```
FDFErc FDFSetEncoding (
    FDFDoc theFDF,
    const char* newEncoding);
```

### Perl Syntax

```
$outFDF->SetEncoding($newEncoding);
```

### Parameters

Table 31:

<b>newEncoding</b>	Null-terminated string that will be converted to a PDF Name before making it the value of the <b>/Encoding</b> key. Acceptable encodings are: <b>Shift_JIS</b> , <b>UHC</b> , <b>GBK</b> , <b>BigFive</b> (exact spelling required).
--------------------	--

### Errors

[FDFErcBadParameter](#), [FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetEncoding](#)

### C Example

```
FDFErc erc = FDFSetEncoding (theFDF, "Shift_JIS");
```

## FDFSetFDFVersion

### (Perl) SetFDFVersion

Sets the FDF version of an FDF file. It is especially important to call this function if the FDF you are generating uses features in versions 1.3, 1.4 and 1.5.

### C Syntax

```
FDFErc FDFSetFDFVersion (FDFDoc the FDF,  
                        const char* cVersion);
```

### ActiveX Syntax

```
Sub FDFSetFDFVersion (bstrVersion As String)
```

### Perl Syntax

```
$outFDF->SetFDFVersion ($Version);
```

### Parameters

**Table 32:**

<b>cVersion</b>	The FDF version that the document will be set to. Either "1.2", "1.3", "1.4", or "1.5".
-----------------	---

### Errors

[FDFErcBadParameter](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetFDFVersion](#)

## FDFSetFile

### (Perl) SetFile

Sets the value of the FDF's **/F** key. If the FDF already has an **/F** key, its old value is replaced. The **/F** key is used to point the FDF file to an existing Acrobat Form (possibly residing on a Web server) that the FDF data is intended to populate. In contrast, [FDFAddTemplate](#) (which is mutually exclusive with **FDFSetFile**) is used to dynamically assemble a brand new Acrobat Form at FDF-import time, from pages located in one or more PDF documents specified by the FDF data, and to populate any fields in the "spawned" pages with FDF.

**NOTE:** An FDF that will populate the same Acrobat form that a Web submission occurred from should not include an **/F** key.

### C Syntax

```
FDFErc FDFSetFile(FDFDoc theFDF, const char* newFile);
```

### ActiveX Syntax

```
Sub FDFSetFile (newFile As String)
```

### Perl Syntax

```
$outFDF->SetFile($newFile);
```

### Parameters

**Table 33:**

<b>newFile</b>	Host-encoded string to use as the new value of the <b>/F</b> key. The value of <b>/F</b> can be a URL, and at FDF import time Acrobat uses the WebLink plug-in to retrieve it. It can also be a path or URL that is relative to that of the FDF file.
----------------	---

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetFile](#)

[FDFSetFileEx](#)

### C Example

```
/* Set the value of the /F key to the PDF on the web server
   for the outgoing fdf */
erc = FDFSetFile(theFDF,
                 "http://myserver.domain.com/app/myfile.pdf");
```

### ActiveX Example

```
objFdf.FDFSetFile "http://myserver/TheForm.pdf"
```

## FDFSetFileEx

### (Perl) SetFileEx

#### (C, Perl only)

Sets the value of the FDF file's **/F** key. The **/F** key is used to point the FDF data to the specific PDF form on a web server. If the FDF already has an **/F** key, its old value is replaced.

**NOTE:** This function is similar to [FDFSetFile](#), except that it allows a more complex file specification.

### C Syntax

```
FDFErc FDFSetFileEx(
    FDFDoc theFDF,
    const pdfFileSpec fileSpec);
```

### Perl Syntax

```
$outFDF->SetFileEx($fileSpec);
```

### Parameters

**Table 34:**

<b>fileSpec</b>	A <a href="#">pdfFileSpec</a> structure containing the new value for the <b>/F</b> key. You need fill in only the fields of the <a href="#">pdfFileSpec</a> that you care about with null-terminated strings, and set the others to <b>NULL</b> . In the most common case, only the <b>/F</b> field is used. In this case, a convenience macro <b>SIMPLE_FILESPEC</b> is available. However, you can simply use the <a href="#">FDFSetFile</a> function.
-----------------	--

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetFile](#)

[FDFSetFile](#)

### C Example

```
/* Set up the pdfFileSpec structure */
pdfFileSpecRec filespec;
memset(&filespec, 0, sizeof(filespec));
filespec.DOS = "C:/theapp/PDFs/MyForm.pdf";

...

/* call FDFSetFileEx to point to the PDF in the apps subdirectory */
theErc = FDFSetFileEx(theFDF, &filespec);
```



## FDFSetFlags

### (Perl) SetFlags

Sets the value of one of the following flags of a field: **/Ff**, **/SetFf**, **/ClrFf**, **/F**, **/SetF**, **/ClrF**. See Section 8.6.6 in the *PDF Reference* (version 1.5, fourth edition) for more information about these flags.

If the field does not exist in the FDF file, Acrobat creates it. If it does exist, Acrobat replaces the old value. If the FDF file is template-based ([FDFAddTemplate](#) has been called), **FDFSetFlags** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetFlags(
    FDFDoc theFDF,
    const char* fieldName,
    FDFItem whichFlags,
    ASUns32 newFlags);
```

### ActiveX Syntax

```
Sub FDFSetFlags(
    fieldName As String,
    whichFlags As Integer,
    newFlags As Integer)
```

### Perl Syntax

```
$outFDF->SetFlags($fieldName, $whichFlags, $newFlags)
```

### Parameters

Table 35:

<b>fieldName</b>	String representing the fully qualified name of the field. (for example, <b>employee.name.last</b> ).
<b>whichFlags</b>	An <a href="#">FDFItem</a> value indicating the flag to set. Must be one of the following flags: <ul style="list-style-type: none"> <li>■ <b>FDFFf</b> sets the <b>/Ff</b> key.</li> <li>■ <b>FDFSetFf</b> sets the <b>/SetFf</b> key.</li> <li>■ <b>FDFClearFf</b> sets the <b>/ClrFf</b> key.</li> <li>■ <b>FDFFlags</b> sets the <b>/F</b> key.</li> <li>■ <b>FDFSetF</b> sets the <b>/SetF</b> key.</li> <li>■ <b>FDFClrF</b> sets the <b>/ClrF</b> key.</li> </ul>
<b>newFlags</b>	The new value for the flags.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

**Related Functions**[FDFAddTemplate](#)[FDFGetFlags](#)**C Example**

```

/*
bit 2 of the annotation flags is the "hidden" bit. The following
statement will hide the field "employee.name.last" (and leave the other
annotation flags untouched) when the FDF gets imported into the PDF
form.
*/
theErc = FDFSetFlags(theFDF, "employee.name.last", FDFSetF, 2);

/* the following statement will make it visible, instead */
theErc = FDFSetFlags(theFDF, "employee.name.last", FDFClrF, 2);

/*
bit 1 of the field flags is the "read-only" flag. The following
statement will make "ssn" read-only (but leave the other field flags
untouched) when the FDF gets imported into the PDF form.
*/
theErc = FDFSetFlags(theFDF, "ssn", FDFSetFf, 1);

```

**ActiveX Example**

```

Const FDFff      = 5
Const FDFSetFf   = 6
Const FDFClearFf = 7
Const FDFFlags   = 8
Const FDFSetF    = 9
Const FDFClrF    = 10

'bit 2 of the annotation flags is the "hidden" bit. The
'following statement hides the field "employee.name.last"
'(and leaves the other annotation flags untouched).
objFdf.FDFSetFlags "employee.name.last", FDFSetF, 2

'Make "employee.name.last" visible
objFdf.FDFSetFlags "employee.name.last", FDFClrF, 2

'bit 1 of the field flags is the "read only" flag. The
'following statement makes "ssn" read-only (but leaves the
'other field flags untouched).
objFdf.FDFSetFlags "ssn", FDFSetFf, 1

```

---

## FDFSetGoToAction

### (Perl) SetGoToAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **GoTo** using a named destination. If the field does not exist in the FDF file, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value.

If the FDF file is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetGoToAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger,  
    const char* theDest,  
    ASBool isName);
```

### ActiveX Syntax

```
Sub FDFSetGoToAction(  
    fieldName As String,  
    whichTrigger As Integer,  
    theDest As String,  
    isName As Boolean)
```

### Perl Syntax

```
$outFDF->SetGoToAction($fieldName, $whichTrigger, $theDest, $isName);
```

**Parameters****Table 36:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <b>FDFActionTrigger</b> value indicating the event trigger for the action: <ul style="list-style-type: none"> <li>• <b>FDFEnter</b></li> <li>• <b>FDFExit</b></li> <li>• <b>FDFDown</b></li> <li>• <b>FDFUp</b></li> <li>• <b>FDFOnFocus</b></li> <li>• <b>FDFOnBlur</b></li> </ul> If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theDest</b>	The string, in PDFDocEncoding or Unicode, to be used as value for the <b>/D</b> key in the <b>GoTo</b> action.
<b>isName</b>	If <b>true</b> , Acrobat converts <b>theDest</b> to a PDF Name before making it the value of <b>/D</b> . If <b>false</b> , <b>theDest</b> is set to a PDF String.

**Errors**

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**,  
**FDFErcEmbeddedFDFs**, **FDFErcInternalError**

**Related Functions**

**FDFSetGoToRAction**

**C Example**

```
retcode = FDFSetGoToAction (theFDF, "my button", FDFDown,
                           "Chap6.begin", true);
```

---

## FDFSetGoToRAction

### (Perl) SetGoToRAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **GoToR** using a named destination. If the field does not exist in the FDF file, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value.

If **theFDF** is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetGoToRAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger,  
    const char* theDest,  
    ASBool isName,  
    const char* theFile,  
    ASBool addNewWindow,  
    ASBool newWindow);
```

### ActiveX Syntax

```
Sub FDFSetGoToRAction(  
    fieldName As String,  
    whichTrigger As Integer,  
    theDest As String,  
    isName As Boolean,  
    theFile As String,  
    addNewWindow As Boolean,  
    newWindow As Integer)
```

### Perl Syntax

```
$outFDF->SetGoToRAction ($fieldName,  
                          $whichTrigger,  
                          $theDest,  
                          $isName,  
                          $theFile,  
                          $addNewWindow,  
                          $newWindow);
```

## Parameters

Table 37:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <b>FDFActionTrigger</b> value indicating the event trigger for the action: <ul style="list-style-type: none"> <li>• <b>FDFEnter</b></li> <li>• <b>FDFExit</b></li> <li>• <b>FDFDown</b></li> <li>• <b>FDFUp</b></li> <li>• <b>FDFOnFocus</b></li> <li>• <b>FDFOnBlur</b></li> </ul> If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theDest</b>	The string to be used as value for the <b>/D</b> key in the <b>GoToR</b> action.
<b>isName</b>	If <b>true</b> , <b>theDest</b> is converted to a PDF Name before making it the value of <b>/D</b> . If <b>false</b> , <b>theDest</b> is set to a PDF string.
<b>theFile</b>	The String to be used as value for the <b>/F</b> key in the <b>GoToR</b> action.
<b>addNewWindow</b>	If <b>true</b> , adds a <b>/NewWindow</b> key to the <b>GoToR</b> action and sets the value passed in <b>newWindow</b> parameter. If <b>false</b> , does not add it, and the <b>newWindow</b> parameter is ignored.
<b>newWindow</b>	Value for the <b>/NewWindow</b> key in the <b>GoToR</b> action. Only used if the <b>addNewWindow</b> parameter is set to <b>true</b> .

## Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**,  
**FDFErcEmbeddedFDFs**, **FDFErcInternalError**

## Related Functions

**FDFAddTemplate**  
**FDFSetGoToAction**

## C Example

```
/* When the user clicks the button "SeeSpecButton", the GoToR action for
the button is now changed to open the PDF reference */
theErc = FDFSetGoToAction (theFDF, "SeeSpecButton", FDFUp,
                          "Appendix H", false, "pdfspec.pdf", true,
false);
```

---

## FDFSetHideAction

### (Perl) SetHideAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **Hide**. If the field does not exist in the FDF file, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value.

If **theFDF** is template-based ([FDFAddTemplate](#) has been called), **FDFSetHideAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetHideAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger,  
    const char* theTarget,  
    ASBool isHide);
```

### ActiveX Syntax

```
Sub FDFSetHideAction(  
    fieldName As String,  
    whichTrigger As Integer,  
    theTarget As String,  
    isHide As Boolean)
```

### Perl Syntax

```
$outFDF->SetHideAction($fieldName, $whichTrigger, $theTarget, $isHide);
```

**Parameters****Table 38:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action: <ul style="list-style-type: none"> <li>• <b>FDFEnter</b></li> <li>• <b>FDFExit</b></li> <li>• <b>FDFDown</b></li> <li>• <b>FDFUp</b></li> <li>• <b>FDFOnFocus</b></li> <li>• <b>FDFOnBlur</b></li> </ul> If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theTarget</b>	The string in PDFDocEncoding to be used as value for the <b>/T</b> key in the <b>Hide</b> action.
<b>isHide</b>	This is the value of the <b>/H</b> key in the <b>Hide</b> action. If <b>true</b> , no <b>/H</b> key is added.

**Errors**

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

**Related Functions**

[FDFAddTemplate](#)

**C Example**

```
retCode = FDFSetHideAction (theFDF, "HideFieldsButton", FDFUp,
    "hideable field", true);
```

**ActiveX Example**

```
Const FDFUp = 3
theFdf.FDFSetHideAction "my button", FDFUp, "hideable field", Trues
```



## FDFSetID

### (Perl) SetID

Sets the value of one element in the FDF's **/ID** key. If the FDF already has an **/ID** key, Acrobat replaces the requested element.

### C Syntax

```
FDFErc FDFSetID(  
    FDFDoc theFDF,  
    ASInt32 nElemNum,  
    const ASUns8* elem,  
    ASInt32 bufSize);
```

### ActiveX Syntax

```
Sub FDFSetID (nElemNum, bstrElem)
```

### Perl Syntax

```
$outFDF->SetID($nElemNum, $elem)
```

### Parameters

Table 39:

<b>nElemNum</b>	The element number to set in the FDF's <b>/ID</b> key. Must be either: <ul style="list-style-type: none"><li>● 0: the permanent ID</li><li>● 1: the changing ID</li></ul>
<b>elem</b>	Buffer containing the ID element to set.
<b>bufSize</b> (C only)	The number of bytes in <b>elem</b>

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetID](#)

### C Example

```
ASUns8 buf[16];  
ASInt32 nBytes;  
FDFErc erc = FDFGetID(theFDF1, 0/*nElemNum*/,  
    buf, sizeof(buf), &nBytes);  
erc = FDFSetID(theFDF2, 0/*nElemNum*/, buf, sizeof(buf));
```

---

## FDFSetIF

### (Perl) SetIF

Sets the Icon Fit for the appearance of a button field (the value of the **/IF** key). The **/IF** key determines the placement of the PDF icon used as the appearance of a button (see also [FDFSetAP](#) and [FDFSetAPRef](#)).

If the field does not yet exist in the FDF, it is created. If it does, and already has an **/IF** key, its old value is replaced. If the FDF does not include an **/IF**, then the icon is placed according to the **/IF** already in the Form (or the default, if that field in the Form does not have an **/IF**).

If **theFDF** is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErrc FDFSetIF(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFScaleWhen scaleWhen,  
    ASBool bProportional,  
    float x,  
    float y);
```

### ActiveX Syntax

```
Sub FDFSetIF(  
    fieldName As String,  
    scaleWhen As Integer,  
    bProportional As Boolean,  
    x As Float,  
    y As Float)
```

### Perl Syntax

```
$outFDF->SetIF($fieldName, $scaleWhen, $bProportional, $x, $y);
```

## Parameters

**Table 40:**

<b>fieldName</b>	String representing the fully qualified name of the field. (for example, <b>employee.name.last</b> ).
<b>scaleWhen</b>	A value indicating when to scale the icon to fit within the rectangle of the annotation represented by the field. See <a href="#">FDFScaleWhen</a> for possible values.
<b>bProportional</b>	<b>true</b> if scaling should be proportional; <b>false</b> otherwise.
<b>x</b>	A float value between 0 and 1. 0.0 places the icon at the left edge and 1.0 at the right edge of the annotation rectangle.
<b>y</b>	A float value between 0 and 1. 0.0 places the icon at the bottom edge and 1.0 at the upper edge of the annotation rectangle.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

## Related Functions

[FDFSetAP](#)  
[FDFSetAPRef](#)

## C Example

```
/*
write the icon fit, align to left edge and top edge
*/

retcode = FDFSetIF (theFDF, "MyButton", FDFTooBig, false, 0.5, 0.5);
```

## ActiveX Example

```
Const FDFAlways    = 0
Const FDFTooSmall = 1
Const FDFTooBig    = 2
Const FDFNever     = 3
objFdf.FDFSetIF "my button", FDFTooSmall, True, .5, .5
```

## FDFSetImportDataAction

### (Perl) SetImportDataAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **ImportData**. If the field does not exist in the FDF, it is created. If it does, and already has an **/A** or **/AA** key, its old value is replaced.

If the FDF is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetImportDataAction (FDFDoc theFDF,
    const char* fieldName, FDFActionTrigger whichTrigger,
    const char* theFile);
```

### ActiveX Syntax

```
Sub FDFSetImportDataAction(
    fieldName As String,
    whichTrigger As Integer,
    theFile As String)
```

### Perl Syntax

```
$outFDF->SetImportDataAction($fieldName, $whichTrigger, $theFile);
```

### Parameters

Table 41:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action: <b>FDFEnter</b> , <b>FDFExit</b> , <b>FDFDown</b> , <b>FDFUp</b> , <b>FDFOnFocus</b> , or <b>FDFOnBlur</b> . If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theFile</b>	Host-encoded string to be used as value for the <b>/F</b> key in the <b>ImportData</b> action.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

### C Example

```
/* Write the field with an import data action on FDFUp */
ec = FDFSetImportDataAction (theFDF, "MyField", FDFUp, "myprof.fdf");
```

**ActiveX Example**

```
Const FDFUp = 3  
objFdf.FDFSetImportDataAction "my button", FDFUp, "myprof.fdf"
```

## FDFSetJavaScriptAction

### (Perl) SetJavaScriptAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **JavaScript**. If the field does not exist in the FDF file, it is created. If it does, and already has an **/A** or **/AA** key, its old value is replaced.

If the FDF file is template-based (**FDFAddTemplate** has been called), **FDFSetJavaScriptAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetJavaScriptAction (FDFDoc theFDF,
    const char* fieldName, FDFActionTrigger whichTrigger,
    const char* theScript);
```

### ActiveX Syntax

```
Sub FDFSetJavaScriptAction (fieldName As String,
    whichTrigger As Integer, theScript As String)
```

### Perl Syntax

```
$outFDF->SetJavaScriptAction($fieldName, $whichTrigger, $theScript);
```

### Parameters

Table 42:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <b>FDFActionTrigger</b> value indicating the event trigger for the action. If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theScript</b>	String containing the text for the script. Must be in either PDFDocEncoding or in Unicode. Hint: Use <b>\r</b> as line separator within the script. Use <b>\t</b> for tabs.

### Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**, **FDFErcEmbeddedFDFs**, **FDFErcInternalError**

### C Example

```
theErc = FDFSetJavaScriptAction (theFDF, "checkbox", FDFUp,
    "var valMe = event.target.value;\r"
    "var fRadio = this.getField(\"radio\");\r\r"
    "if (valMe == \"Off\")\r"
    "\tfRadio.value = \"left\"\r"
    "else\r"
    "\tfRadio.value = \"right\";\r");
```

**ActiveX Example**

```
Const FDFUp = 3
'Use Chr(13) to add a CR, Chr(9) for tabs
objFdf.FDFSetJavaScriptAction "my button", FDFUp,
"var f = this.getField(""Approved"");" &
Chr(13) & "f.hidden = false;"
```

## FDFSetNamedAction

### (Perl) SetNamedAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to a named action. If the field does not exist in the FDF file, it is created. If it does, and already has an **/A** or **/AA** key, its old value is replaced.

If the FDF file is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetNamedAction( FDFDoc theFDF,
    const char* fieldName, FDFActionTrigger whichTrigger,
    const char* theName);
```

### ActiveX Syntax

```
Sub FDFSetNamedAction(fieldName As String,
    whichTrigger As Integer, theName As String)
```

### Perl Syntax

```
$outFDF->SetNamedAction($fieldName, $whichTrigger, $theName);
```

### Parameters

Table 43:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action. May be either: <b>FDFEnter</b> , <b>FDFExit</b> , <b>FDFDown</b> , <b>FDFUp</b> , <b>FDFOnFocus</b> , <b>FDFOnBlur</b> If <b>FDFUp</b> , <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theName</b>	String in PDFDocEncoding converted to a PDF name to use as the value of <b>/N</b> in the named action.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

### Related Functions

[FDFAddTemplate](#)

### C Example

```
retCode = FDFSetNamedAction (theFDF, "Named1", FDFUp, "FirstPage");
```



**ActiveX Example**

```
Const FDFUp = 3  
theFdf.FDFSetNamedAction "my button", FDFUp, "FirstPage"
```

## FDFSetOnImportJavaScript

### (Perl) SetOnImportJavaScript

Adds a script to the FDF, which Acrobat then executes when it imports the FDF. You can set two such scripts: one executes just before the data in the FDF is imported and one executes afterwards. (You need to call this function twice to set both scripts.)

### C Syntax

```
FDFErc FDFSetOnImportJavaScript (
    FDFDoc theFDF,
    const char* cScript,
    ASBool bBefore);
```

### Perl Syntax

```
$outFDF->SetOnImportJavaScript($cScript, $bBefore);
```

### ActiveX Syntax

```
Sub FDFSetOnImportJavaScript (cScript As String, bBefore As Boolean)
```

### Parameters

**Table 44:**

<b>cScript</b>	Must either be in PDFDocEncoding or in Unicode. Use "\r" as line separator within the script. Use "\t" for tabs.
<b>bBefore</b>	Pass <b>true</b> to have <b>cScript</b> execute before the data in the FDF is imported, <b>false</b> to indicate execution after import.

### Return Values

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcInternalError](#)

### C Example

```
FDFErc erc = FDFSetOnImportJavaScript (theFDF,
    "app.alert(\"FDF came back\");", true/*bBefore/);
```

## FDFSetOpt

### (Perl) SetOpt

Sets the value of one element in a field's **/Opt** key. Each element represents one of the "n" potential values of the field, and is either:

- Text representing the item's export value
- An array containing two strings, the first representing the item's export value, and the second representing the item name (appearance).

If the field does not exist in the FDF, Acrobat creates it. If it does exist, and already has an **/Opt** array, Acrobat replaces the requested element in the array. If **theFDF** is template-based (**FDFAddTemplate** has been called), **FDFSetOpt** acts on the most recently added template.

An FDF file used to dynamically change the **/Opt** of a field in an Acrobat Form typically needs to also change the **/V** of that same field (use **FDFSetValue**), particularly if the old **/V** of the field is not a member of the new **/Opt** being imported into the form.

**NOTE:** If the **/Opt** includes item names and export values, **/V** should use one of the export values, not one of the item names.

### C Syntax

```
FDFErrc FDFSetOpt(
    FDFDoc theFDF,
    const char* fieldName,
    ASInt32 nElemNum,
    const char* string1,
    const char* string2);
```

### ActiveX Syntax

```
Sub FDFSetOpt(
    fieldName As String,
    nElemNum as Integer,
    string1 As String,
    string2 As String)
```

### Perl Syntax

```
$outFDF->SetOpt($fieldName, $nElemNum, $string1, $string2);
```

### Parameters

**Table 45:**

<b>fieldName</b>	String representing the fully-qualified name of the field (for example, <b>employee.name.last</b> ).
<b>nElemNum</b>	The index of the element to set in the <b>/Opt</b> array. The first element in the <b>/Opt</b> array has an index of 0 (zero).

Table 45:

<b>string1</b>	Contains the first of two strings if the <b>/Opt</b> array element is an array containing two strings, or the only string if the <b>/Opt</b> array element is a single string. In either case, this value represents the export value of the item. See <a href="#">FDFSetAS</a> for a discussion of encodings.
<b>string2</b>	Contains the second of two strings (the item name), if the <b>/Opt</b> array element to be set is an array containing two strings Otherwise, it is <b>NULL</b> .

**Errors**

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcInternalError](#)

**Related Functions**

[FDFGetOpt](#)  
[FDFGetOptNumElem](#)

**C Example**

```

/** next section of code builds the list box values on the
    MyListBox field.*/
/* set the first element in a list box */
    retcode = FDFSetOpt (theFDF, "MyListBox", 0, "X",
        "Please choose your language option");

/* set the second element in a list box */
    retcode = FDFSetOpt (theFDF, "MyListBox", 1, "E", "English");

/* set the third element in a list box */
    retcode = FDFSetOpt (theFDF, "MyListBox", 2, "S", "Spanish");

/* set the fourth element in a list box */
    retcode = FDFSetOpt (theFDF, "MyListBox", 3, "G", "German");

/* set the fifth element in a list box */
    retcode = FDFSetOpt (theFDF, "MyListBox", 4, "F", "French");

/* now use FDFSetValue to set a specific value for the field */
    retcode = FDFSetValue (theFDF, "MyListBox", "English", false);

```

**ActiveX Example**

```
objFdf.FDFSetOpt "Credit Card", 2, "Visa", ""
```

## FDFSetResetByNameAction

### (Perl) SetResetByNameAction

(C, Perl only)

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **ResetForm**, using the passed field names. This is similar to [FDFSetResetFormAction](#), except that you can specify which fields to reset.

The parameter **theFlags** determines how the **Fields** key is interpreted and thus how fields are reset. If **theFlags** bit is 0, then the array of fields passed in the **fields** parameter will be reset. If the **theFlags** is 1, then all fields in the form are reset *except* for those in the **Fields** array.

If the FDF is template-based ([FDFAddTemplate](#) has been called), **FDFSetResetByNameAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetResetByNameAction(
    FDFDoc theFDF,
    const char* fieldName,
    FDFActionTrigger whichTrigger,
    ASUns32 theFlags,
    ASInt32 nFlds,
    const char* fields[]);
```

### Perl Syntax

```
$outFDF->SetResetByNameAction ($fieldName,
                                $whichTrigger,
                                $theFlags
                                $fields[])
```

### Parameters

Table 46:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action: <b>FDFEnter</b> , <b>FDFExit</b> , <b>FDFDown</b> , <b>FDFUp</b> , <b>FDFOnFocus</b> , or <b>FDFOnBlur</b>  If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.

Table 46:

<b>theFlags</b>	Value for the <b>/Flags</b> key in the <b>ResetForm</b> action. At present, only one flag is defined: <ul style="list-style-type: none"> <li>• Bit 1 - <b>Include/exclude</b> flag: determines how the <b>fields</b> key is interpreted. If this bit is 0, then <b>fields</b> represents the individual fields to reset. If the bit is 1, then all fields in the AcroForm are reset except for those in the <b>fields</b> array.</li> </ul>
<b>nFlds</b>	An integer specifying how many field names are being passed in the <b>fields</b> parameter. If zero, this function is identical to <a href="#">FDFSetResetFormAction</a> .
<b>fields</b>	An array of <b>nFlds</b> pointers to Strings representing the fully qualified names of the fields to not reset. The fields, if they have a value, will retain that value and all other fields will be reset. These are same as the values of the <b>/Fields</b> key in the <b>ResetForm</b> action.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

## Related Functions

[FDFSetResetFormAction](#)

## C Example

```
/* only reset (or don't reset) the following fields ...*/
const char* fieldNames[]={"CustomerName", "CustomerAddress"};

/* In this example, passing 1 for the "theFlags" parameter will reset
all the fields in the form, *except* for the two passed in the "fields"
parameter */
theErc = FDFSetResetByNameAction(theFDF, "reset",  FDFUp, 1,
                                2,      fieldNames);
/* In this example, passing 0 for the "theFlags" parameter will reset
just the two fields passed in the "fields" parameter */
theErc = FDFSetResetByNameAction(theFDF, "reset",  FDFUp, 0,
                                2,      fieldNames);
```

---

## FDFSetResetFormAction

### (Perl) SetResetFormAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **ResetForm**. The created action does not include **/Fields** or **/Flags** keys.

If the field does not exist in the FDF file, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value. If the FDF file is template-based ([FDFAddTemplate](#) has been called), **FDFSetResetFormAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetResetFormAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger);
```

### ActiveX Syntax

```
Sub FDFSetResetFormAction(  
    fieldName As String,  
    whichTrigger As Integer,  
    theFlags As Long,  
    rgFields As Array)
```

### Perl Syntax

```
$outFDF->SetResetFormAction ($fieldName, $whichTrigger);
```

**Parameters****Table 47:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	<p>An <b>FDFActionTrigger</b> value indicating the event trigger for the action:</p> <ul style="list-style-type: none"> <li>● <b>FDFEnter</b></li> <li>● <b>FDFExit</b></li> <li>● <b>FDFDown</b></li> <li>● <b>FDFUp</b></li> <li>● <b>FDFOnFocus</b></li> <li>● <b>FDFOnBlur</b></li> </ul> <p>If <b>FDFUp</b>, an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.</p>
<b>rgFields</b> (ActiveX only)	Optional parameter. If not passed, then the created action will not include a <b>/Fields</b> key. Otherwise, it should be an array of strings, representing the names of the fields to reset (or exclude from resetting) when the action gets executed.

**Errors**

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**,  
**FDFErcEmbeddedFDFs**, **FDFErcInternalError**

**Related Functions**

**FDFSetResetByNameAction**

**C Example**

```
/* Write with a reset form action on FDFUp */
retcode = FDFSetResetFormAction (theFDF, "ResetButton", FDFUp);
```



## FDFSetRichValue

### (Perl) SetRichValue

Sets the rich text value of a field (the **/RV** value). If the field does not yet exist in the FDF, it is created. If it already does exist and already has a value, the old value is replaced.

If **theFDF** is a "template-based" FDF (i.e., an FDF for which **FDFAddTemplate** has been called), this function acts on the most recently added template.

**NOTE:** If an FDF contains the **/RV** key, it takes priority over the **/V** key on import.

### C Syntax

```
FDFErc FDFSetRichValue(FDFDoc theFDF,
    const char* fieldName,
    const char* newValue,
    ASBool bNotUsed);
```

### ActiveX Syntax

```
Sub SetRichValue (newValue As String)
```

### Perl Syntax

```
$outFDF->SetRichValue ($theFDF, $newValue);
```

### Parameters

**Table 48:**

<b>fieldName</b>	String representing the fully-qualified name of the field (for example, <b>employee.name.last</b> ).
<b>newValue</b>	If the FDF includes an <b>/Encoding</b> key (see <b>FDFSetEncoding</b> ), then the string you pass should be in that encoding. Otherwise, the value should be in PDFDocEncoding, or in Unicode if it cannot be represented in PDFDocEncoding (i.e. it contains double-byte characters).
<b>bNotUsed</b>	This parameter is not used by this method. You can pass <b>true</b> or <b>false</b> .

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcCantInsertField](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetRichValue](#)

## FDFSetStatus

### (Perl) SetStatus

Sets the value of the **/Status** key. If the FDF data already has a **/Status** key, Acrobat replaces its old value.

When an FDF file containing a **/Status** key is returned from a server after a submission, the value of this key is displayed in an alert box to the user.

### C Syntax

```
FDFErc FDFSetStatus(
    FDFDoc theFDF,
    const char* newStatus);
```

### ActiveX Syntax

```
Sub FDFSetStatus (newStatus As String)
```

### Perl Syntax

```
$outFDF->SetStatus ($theFDF, $newStatus);
```

### Parameters

**Table 49:**

<b>newStatus</b>	Host-encoded string to use as new status.
------------------	---

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetStatus](#)

### C Example

```
/* return the status of the submission */
retcode = FDFSetStatus (theFDF, "Your order has been
entered. Thanks for shopping at the Adobe Online Store");
```

### ActiveX Example

```
objFdf.FDFSetStatus "Your order has been entered"
```

---

## FDFSetSubmitByNameAction

### (Perl) SetSubmitByNameAction

(C, Perl only)

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **SubmitForm**, using the URL it is passed. This is similar to [FDFSetSubmitFormAction](#), except that you can specify which fields to submit.

If the FDF is template-based ([FDFAddTemplate](#) has been called), **FDFSetSubmitByNameAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetSubmitByNameAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger,  
    const char* theURL,  
    ASUns32 theFlags,  
    ASInt32 nFlds,  
    const char* fields[]);
```

### Perl Syntax

```
$outFDF->SetSubmitByNameAction (  
    $fieldName,  
    $whichTrigger,  
    $theURL,  
    $theFlags,  
    $nFlds,  
    $fields[]);
```

## Parameters

Table 50:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action: <b>FDFEnter</b> , <b>FDFExit</b> , <b>FDFDown</b> , <b>FDFUp</b> , <b>FDFOnFocus</b> , or <b>FDFOnBlur</b> If <b>FDFUp</b> , an <b>/A</b> entry used; otherwise an <b>/AA</b> entry is created.
<b>theURL</b>	String to be used as value for the <b>/F</b> key in the <b>SubmitForm</b> action.
<b>theFlags</b>	Value for the <b>/Flags</b> key in the <b>SubmitForm</b> action. By default it is "0". Other values that can be used for the <b>/Flags</b> key are described in Section 5.7.1 in the <i>PDF Reference (fourth edition, version 1.5)</i> .
<b>nFlds</b>	An integer specifying how many field names are being passed in fields parameter. If zero, this function is identical to <a href="#">FDFSetSubmitFormAction</a> .
<b>fields</b>	An array of <b>nFlds</b> pointers to Strings representing the fully qualified names of the fields to submit.  <b>NOTE:</b> In ActiveX, this parameter is provided in the <a href="#">FDFSetSubmitFormAction</a> function.

## Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcEmbeddedFDFs](#),  
[FDFErcCantInsertField](#), [FDFErcInternalError](#)

## Related Functions

[FDFSetSubmitFormAction](#)  
[FDFSetResetByNameAction](#)

## C Example

```
/* bit 2 of the flags indicates whether to submit as FDF or as HTML. By
passing 4 as the value of "theFlags" parameter, the SubmitForm action
will submit in HTML format */
theErc = FDFSetSubmitFormAction(theFDF, "Submit Button", FDFUp,
    "http://myserver/cgi-bin/myscript#FDF", 4);
```

## FDFSetSubmitFormAction

### (Perl) SetSubmitFormAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **SubmitForm**, using the passed URL. The created action will not include a **/Fields** key.

If the field does not exist in the FDF file, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value. If the FDF is template-based ([FDFAddTemplate](#) has been called), **FDFSetSubmitFormAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetSubmitFormAction (FDFDoc theFDF,
    const char* fieldName, FDFActionTrigger whichTrigger,
    const char* theURL, ASUns32 theFlags);
```

### ActiveX Syntax

```
Sub FDFSetSubmitFormAction (fieldName As String,
    whichTrigger As Integer, theURL As String,
    theFlags as Integer, fields As Array)
```

### Perl Syntax

```
$outFDF->SetSubmitFormAction ($fieldName,
    $whichTrigger, $theURL,
    $theFlags);
```

### Parameters

Table 51:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <a href="#">FDFActionTrigger</a> value indicating the event trigger for the action: <ul style="list-style-type: none"> <li>• <b>FDFEnter</b>, <b>FDFExit</b>, <b>FDFDown</b>, <b>FDFUp</b>, <b>FDFOnFocus</b>, or <b>FDFOnBlur</b></li> </ul> If <b>FDFUp</b> , an <b>/A</b> entry used, otherwise <b>/AA</b> entry is created.
<b>theURL</b>	String to be used as value for the <b>/F</b> key in the <b>SubmitForm</b> action.
<b>theFlags</b>	Value for the <b>/Flags</b> key in the <b>SubmitForm</b> action. By default it is "0". Other values that can be used for the <b>/Flags</b> key are described in Section 5.7.1 in the <i>PDF Reference (fourth edition, version 1.5)</i> .

Table 51:

<b>fields</b> (ActiveX only)	(Optional parameter) If not passed, then the created action does not include a <b>/Fields</b> key. Otherwise, it should be an array of strings, representing the names of the fields to submit (or exclude from submitting) when the action gets executed.  <b>NOTE:</b> This parameter allows ActiveX to have the same functionality as <a href="#">FDFSetSubmitByNameAction</a> .
---------------------------------	---

**Return Value**

Perl, C: Error code

**Errors**

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

**Related Functions**

[FDFSetResetByNameAction](#)  
[FDFSetSubmitByNameAction](#)

**C Example**

```
/* bit 2 of the flags indicates whether to submit as FDF or as HTML. By
passing 4 as the value of "theFlags" parameter, the SubmitForm action
will submit in HTML format */
theErc = FDFSetSubmitFormAction(theFDF, "Submit Button", FDFUp,
    "http://myserver/cgi-bin/myscript#FDF", 4);
```

**ActiveX Example**

```
Const FDFEnter = 0
Const FDFExit = 1
Const FDFDown = 2
Const FDFUp = 3
Dim rgFlds(1)
rgFlds(0) = "field1"
rgFlds(1) = "field2"
objFdf.FDFSetSubmitFormAction "my button", FDFUp, _
"http://myserver/ASPSamp/Samples/testfdf.asp#FDF", 4, rgFlds
```

## FDFSetTargetFrame

### (Perl) SetTargetFrame

Sets the value of the FDF's **/Target** key. This key is only relevant if the FDF has an **/F** key as well (see [FDFSetFile](#) or [FDFSetFileEx](#)). You can use the **/Target** key to cause the PDF (the one indicated by the **/F** key) to open in a particular browser frame. This key is equivalent to the "target" attribute in HTML.

### C Syntax

```
FDFErc FDFSetTargetFrame(  
    FDFDoc theFDF,  
    const char* cTargetFrame);
```

### ActiveX Syntax

```
Sub FDFSetTargetFrame (cTargetFrame As String)
```

### Perl Syntax

```
$outFDF->SetTargetFrame ($cTargetFrame);
```

### Parameters

Table 52:

<b>cTargetFrame</b>	Value of the <b>/Target</b> key. Should be Host-encoded.
---------------------	--

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcInternalError](#)

### Related Functions

[FDFSetFile](#)  
[FDFSetFileEx](#)

### C Example

```
FDFErc erc = FDFSetTargetFrame (theFDF, "left");
```

## FDFSetURIAction

### (Perl) SetURIAction

Sets the value of either the **/A** or **/AA** keys (actions or additional actions) of a field to an action of type **URI**. If the field does not exist in the FDF, Acrobat creates it. If it does exist, and already has an **/A** or **/AA** key, Acrobat replaces its old value.

If the FDF is template-based ([FDFAddTemplate](#) has been called), **FDFSetURIAction** acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetURIAction(  
    FDFDoc theFDF,  
    const char* fieldName,  
    FDFActionTrigger whichTrigger,  
    const char* theURI,  
    ASBool isMap);
```

### ActiveX Syntax

```
Sub FDFSetURIAction(  
    fieldName As String,  
    whichTrigger As Integer,  
    theURI As String,  
    isMap As Boolean)
```

### Perl Syntax

```
$outFDF->SetURIAction(  
    $fieldName,  
    $whichTrigger,  
    $theURI,  
    $isMap);
```



## Parameters

Table 53:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>whichTrigger</b>	An <b>FDFActionTrigger</b> value indicating the event trigger for the action: <ul style="list-style-type: none"> <li>• <b>FDFEnter</b></li> <li>• <b>FDFExit</b></li> <li>• <b>FDFDown</b></li> <li>• <b>FDFUp</b></li> <li>• <b>FDFOnFocus</b></li> <li>• <b>FDFOnBlur</b></li> </ul> If <b>FDFUp</b> , an <b>/A</b> entry is used, otherwise an <b>/AA</b> entry is created.
<b>theURI</b>	Host-encoded string to be used as value for the <b>/URI</b> key in the URI action.
<b>isMap</b>	Value for the <b>/IsMap</b> key in the URI action. If <b>false</b> , the <b>/IsMap</b> key is not added.

## Errors

**FDFErcBadParameter**, **FDFErcBadFDF**, **FDFErcCantInsertField**,  
**FDFErcEmbeddedFDFs**, **FDFErcInternalError**

## C Example

```
/* Set the new URI action to open the web page*/
retCode = FDFSetURIAction (outputFDF, "OpenWebPageButton",
    FDFUp, "http://www.adobe.com", false);
```

## ActiveX Example

```
Const FDFUp = 3
theFdf.FDFSetURIAction "my button", FDFUp, _
"http://myserver/TheForm.pdf", False
```

## FDFSetValue

### (Perl) SetValue

Sets the value of a field (the value of the **/V** key). If the field does not exist, Acrobat creates it. If it does exist, and already has a value, Acrobat replaces the old value.

If the FDF file is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetValue( FDFDoc theFDF, const char* fieldName,
                    const char* newValue, ASBool bNotUsed);
```

### ActiveX Syntax

```
Sub FDFSetValue(fieldName As String, newValue As String, ASBool
bNotUsed)
```

### Perl Syntax

```
$outFDF->SetValue ($fieldName, $newValue, $bNotUsed);
```

### Parameters

**Table 54:**

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>newValue</b>	String to use as the new value. If the FDF includes an <b>/Encoding</b> key (see <a href="#">FDFSetAS</a> ), then the string should be in that encoding. Otherwise, the value should be in either PDFDocEncoding, or in Unicode if it cannot be represented in PDFDocEncoding (in other words, it contains double-byte characters). Hint: For radio buttons and checkboxes, <b>newValue</b> must be either "Off", or a value that was entered as the "Export Value" when defining the properties of the field.
<b>bNotUsed</b>	Not used.

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

### Related Functions

[FDFGetValue](#)

### C Example

```
FDFErc erc = FDFSetValue (theFDF, "ssn", "123456789", false);
```

**ActiveX Example**

```
objFdf.FDFSetValue "employee.name.last", "Jones", False
```

## FDFSetValues

### (Perl) SetValues

Sets the value of a field to an array. If the field does not exist, Acrobat creates it. If it does exist, and already has a value, Acrobat replaces the old value. This function only works for fields of type "listbox" that have the "Multiple Selection" option enabled.

If the FDF file is template-based ([FDFAddTemplate](#) has been called), this function acts on the most recently added template.

### C Syntax

```
FDFErc FDFSetValues(
    FDFDoc theFDF,
    const char* fieldName,
    ASInt32 nValues,
    const char* newValues[]);
```

### ActiveX Syntax

```
Sub FDFSetValues (fieldName As String, newValues As Array)
```

### Perl Syntax

```
$outFDF->SetValues ($fieldName, @newValues);
```

### Parameters

Table 55:

<b>fieldName</b>	String representing the fully qualified name of the field (for example, <b>employee.name.last</b> ).
<b>nValues</b> (C only)	An integer specifying how many values are being passed in the parameter <b>newValues</b> . If you set this to 1, <b>FDFSetValues</b> behaves exactly as <a href="#">FDFSetValue</a> .
<b>newValues</b>	An array of strings. See <a href="#">FDFSetValue</a> for a discussion of encodings

### Errors

[FDFErcBadParameter](#), [FDFErcBadFDF](#), [FDFErcCantInsertField](#),  
[FDFErcInternalError](#)

### Related Functions

[FDFSetValue](#)

[FDFGetValue](#)

### C Example

```
char* myValues[2] = {"first", "second"};
```

```
FDFErc erc = FDFSetValues(theFDF, "my listbox", 2/*nValues*/,  
                           myValues);
```

## new

### (Perl only)

Creates a new FDF object. If you don't specify parameters for optional arguments (shown here in square brackets), **new** creates a new FDF. If you specify them, **new** opens the existing FDF.

### Perl Syntax

```
$FDFobj = new Acrobat::FDF([ $NameOfFile ], [ $NumOfFiles ] );
```

### Parameters

**Table 56:**

<b>\$NameOfFile</b>	The name for the new FDF object
<b>\$NumOfFiles</b>	The number of bytes to read if "-" ( <b>stdin</b> ) is passed in as <b>\$NameOfFile</b> .

### Return Value

A new FDF object.

### Related Functions

[newFromEmbedded](#)

newFromBuf

(Perl only)

Creates a new FDF object from a buffer.

Perl Syntax

```
$FDFobj = Acrobat::FDF::newFromBuf ($buffer);
```

Parameters

Table 57:

buffer	The buffer containing the FDF to be opened.
--------	---

Return Value

A new FDF object.

Related Methods

[newFromEmbedded](#)

---

## newFromEmbedded

### (Perl only)

Use this call when you discover that your FDF is actually only a container for one or more "real" FDFs (when any of the calls return the error code [FDFErcEmbeddedFDFs](#)).

### Perl Syntax

```
$FDFObj = $inFDF->OpenFromEmbedded($iWhich, $password);
```

### Parameters

**Table 58:**

<b>iWhich</b>	Zero-based index of the embedded FDF that should be opened.
<b>Password</b>	If the embedded FDF is encrypted, you need to pass in the correct password. If it is not encrypted, the system ignores this parameter.

### Return Value

The embedded FDF object.

### Perl Example

```
$FDFObj1 = new Acrobat::FDF($mytestfile);  
my $MYOBJ2;  
$MYOBJ2 = $FDFObj1->NewFromEmbedded(0, "");
```

### Related Functions

[new](#)

[FDFOpenFromEmbedded](#)



## EmbedAndClose

(Perl only)

Embeds an FDF inside another, then closes it. A container FDF can be a carrier for multiple embedded FDFs. The container FDF cannot be a templates-based FDF (if it is, the system returns [FDFErcIncompatibleFDF](#).) The FDF to embed cannot itself be a container (if it is, [FDFErcEmbeddedFDF](#) returns) or be a templates-based FDF ([FDFErcIncompatibleFDF](#) returns).

Each embedded FDF may optionally be password protected.

A container FDF can optionally include, besides the embedded FDFs, only an **/F** and/or **/ID** keys (see [FDFSetFile](#) and [FDFSetID](#).) Any other desired attributes belong in the embedded FDFs themselves. In turn, Acrobat ignores any **/F** or **/ID** keys in the embedded FDFs on FDF import, since it only looks for those keys in the container FDF.

You can use this call multiple times for the same container FDF. Each successive FDF is embedded after any previous FDFs already present in the container.

### Perl Syntax

```
FDFErc EmbedAndClose ($FDFObj, $cPassword);
```

### Parameters

Table 59:

<b>FDFObj</b>	The object that holds one or more FDFs
<b>cPassword</b>	If the embedded FDF should be stored encrypted, you need to provide the correct password. The password consists of a string, and terminates with a single <b>NULL</b> . If you pass in an empty string, the FDF embeds in an unencrypted state.

### Errors

[FDFErcBadParameter](#), [FDFErcIncompatibleFDF](#), [FDFErcEmbeddedFDFs](#), [FDFErcInternalError](#)

### Related Functions

[FDFSetIF](#)  
[FDFSetID](#)

---

## Callbacks

---

### FDFEnumValuesProc

*(C only)*

Callback for [FDFEnumValues](#). Called once for each field value.

#### C Syntax

```
typedef ASBool (*FDFEnumValuesProc)(char* bufFldName,
                                     char* bufFldVal, void* clientData);
```

#### Parameters

**Table 60:**

<b>bufFldName</b>	Buffer containing a field name. This buffer was supplied in the call to <b>FDFEnumValues</b> , and is filled by that function with a new field name for each invocation of the callback.
<b>bufFldVal</b>	Buffer containing a field value. This buffer was supplied in the call to <b>FDFEnumValues</b> , and is filled by that function with a new value for each invocation of the callback. If the FDF includes an <b>/Encoding</b> key (see <a href="#">FDFGetEncoding</a> ), then the returned value will be in that encoding. Otherwise, the value will be in either PDFDocEncoding, or in Unicode if it contains double-byte characters.
<b>clientData</b>	User-supplied data that was passed in the call to <b>FDFEnumValues</b> .

#### Return Value

**true** to continue enumeration, **false** to halt it.

#### Related Functions

[FDFEnumValues](#)

[FDFGetEncoding](#)

# ThreadsafeCallback

(C only)

Callback to threadsafe operations (lock, unlock and destroy).

## C Syntax

```
typedef void (*ThreadsafeCallback)( void* threadsafeObj );
```

## Parameters

Table 61:

threadsafeObj	The data that was passed to <a href="#">FDFRegisterThreadsafeCallbacks</a> when this callback was registered.
---------------	---

## Related Functions

[FDFRegisterThreadsafeCallbacks](#)

