Contents

Preface ................................................. 7
   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

Chapter 1  Introduction ............................... 11
   Acrobat Forms ......................................... 11
   Forms Data Format (FDF) ............................... 11
   The FDF Toolkit ....................................... 11
   Submitting Data to a Server ............................ 12
   Encoding .............................................. 12

Chapter 2  FDF and Web Server Connectivity .......... 13
   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

Chapter 3  FDF Toolkit for C/C++ ....................... 23
   Building Applications with the FDF Toolkit .......... 23
Contents

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

Chapter 4 FDF Toolkit for ActiveX ............................................. 29
  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

Chapter 5 FDF Toolkit for Perl ................................................... 35
  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

Chapter 6 FDF Toolkit For Java .................................................. 41
  Setup and Sample Usage of the Java FDF Toolkit ..................... 41
  Changes to the API .............................................................. 41
Chapter 7  FDF Toolkit Reference  

Language Differences  
Method Names  
Parameters  
Errors and Return Values  
Encoding  
Methods  
FDFAddDocJavaScript  
FDFAddTemplate  
FDFClose  
FDFCreate  
FDFEmbedAndClose  
FDFEnumValues  
FDFExtractAppendSaves  
FDFExtractAttachment  
FDFFinalize  
FDFGetAP  
FDFGetEncoding  
FDFGetFDFVersion  
FDFGetFile  
FDFGetFlags  
FDFGetID  
FDFGetNthValue  
FDFGetOpt  
FDFGetOptNumElem  
FDFGetRichValue  
FDFGetStatus  
FDFGetValue  
FDFGetVersion  
FDFInitialize  
FDFOpen  
FDFOpenFromBuf  
FDFOpenFromEmbedded
Contents

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:
Preface

What’s New In This Version Of The FDF Toolkit

- 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:
- 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


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


Conventions Used in This Book

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

<table>
<thead>
<tr>
<th>Font</th>
<th>Used for</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospaced</td>
<td>Paths and filenames</td>
<td>C:\templates\mytmpl.fm</td>
</tr>
<tr>
<td></td>
<td>Code examples set off from plain text</td>
<td>These are variable declarations:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AVMenu commandMenu, helpMenu;</td>
</tr>
<tr>
<td>monospaced bold</td>
<td>Code items within plain text</td>
<td>The GetExtensionID method ...</td>
</tr>
<tr>
<td></td>
<td>Parameter names and literal values in reference documents</td>
<td>The enumeration terminates if proc returns false.</td>
</tr>
<tr>
<td>monospaced italic</td>
<td>Pseudocode</td>
<td>ACCB1 void ACCB2 ExeProc(void) { do something }</td>
</tr>
<tr>
<td></td>
<td>Placeholders in code examples</td>
<td>AFSimple_Calculate(cFunction, cFields)</td>
</tr>
<tr>
<td>Font</td>
<td>Used for</td>
<td>Examples</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>blue</td>
<td>Live links to Web pages</td>
<td>The Acrobat Solutions Network URL is: <a href="http://partners/adobe.com/asn/">http://partners/adobe.com/asn/</a></td>
</tr>
<tr>
<td></td>
<td>Live links to sections within this document</td>
<td>See Using the SDK.</td>
</tr>
<tr>
<td></td>
<td>Live links to other Acrobat SDK documents</td>
<td>See the Acrobat Core API Overview.</td>
</tr>
<tr>
<td></td>
<td>Live links to code items within this document</td>
<td>Test whether an ASA.com exists.</td>
</tr>
<tr>
<td>bold</td>
<td>PostScript language and PDF operators, keywords, dictionary key names</td>
<td>The setpagedevice operator</td>
</tr>
<tr>
<td></td>
<td>User interface names</td>
<td>The File menu</td>
</tr>
<tr>
<td>italic</td>
<td>Document titles that are not live links</td>
<td>Acrobat Core API Overview</td>
</tr>
<tr>
<td></td>
<td>New terms</td>
<td>User space specifies coordinates for...</td>
</tr>
<tr>
<td></td>
<td>PostScript variables</td>
<td>filename deletefile</td>
</tr>
</tbody>
</table>
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](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.


---

### 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`:

```c
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**

```c
/*
 ** Copyright 1999 Adobe Systems, Inc.
 **
 ** parseFDF - A sample Windows NT application to parse PDF 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"));

/*
** FDOpen:
*/

codeError = FDOpen("-", howMany, &FdfInput);

*/

** FDFGetValue:
*/

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

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

codeError = 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 similar 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");
    **
    **
    **
    ** 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 PDF completed\n");
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.

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

    errorCode = FDFClose(FdfInput);

}
2. Call methods to create the FDF data.
3. Set the values of the individual fields (the /V 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:

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

**Step Three: Setting Field Values**

You use FDFSSetValue (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.

FDFSSetValue 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”.

```c
errorCode = FDFSSetValue (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:

```c
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:

```c
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 "Fdftk.h"

/*
 Main application
 */
void main()
FDF and Web Server Connectivity
Creating the Client Side PDF File

#define FDF_OK 0
#define FDF_ERROR -1

FDFDoc FdfOutput = NULL;

retCode = FDFCreate (&FdfOutput);
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);
retCode = FDFSetFile(FdfOutput, "http://localhost/PDF/generateFDF.pdf");
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 PDF 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, "http://localhost/PDF/generateFDF.pdf");
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.

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:

```c
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.
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 in your source code:

```c
#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 `\x{fe}` 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.
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFGetVersion</td>
<td>Gets the current version of the FDF Toolkit library.</td>
</tr>
<tr>
<td>FDFOpen</td>
<td>Opens existing FDF file or reads stdin in C.</td>
</tr>
<tr>
<td>FDFOpenFromEmbedded</td>
<td>Opens an FDF that is embedded in a container.</td>
</tr>
<tr>
<td>FDFRemoveItem</td>
<td>Removes key-value pairs from the FDF file.</td>
</tr>
<tr>
<td>FDFClose</td>
<td>Frees resources used by the FDF file.</td>
</tr>
<tr>
<td>FDFEmbedAndClose</td>
<td>Embeds FDF in another document and then closes it.</td>
</tr>
<tr>
<td>FDFSsave</td>
<td>Writes out an FDF file.</td>
</tr>
</tbody>
</table>
Methods That Parse FDF Data

The following methods parse FDF data.

**TABLE 3.1**

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

Methods That Generate FDF Data

The following methods generate FDF data.

**TABLE 3.2**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFAddDocJavaScript</td>
<td>Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.</td>
</tr>
</tbody>
</table>
### Table 3.2

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

The following methods are specific to the Unix platform.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FDFInitialize</strong></td>
<td>Initializes the FDF Toolkit library.</td>
</tr>
<tr>
<td><strong>FDFFinalize</strong></td>
<td>Frees resources in the FDF Toolkit library.</td>
</tr>
<tr>
<td><strong>FDFRegisterThreadsafeCallbacks</strong></td>
<td>Registers callbacks for thread-save operations for lock, unlock and destroy.</td>
</tr>
</tbody>
</table>
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:

```vbscript
<% 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\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\MimeMap
```

Once found, add a REG_SZ value for the MIME mapping required.

Using Regedit32.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 FDFErc. 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 FDFErcFileSysErr. 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 FDFErcOK.

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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFCreate</td>
<td>Creates a new FDF file.</td>
</tr>
<tr>
<td>FDFGetVersion</td>
<td>Returns the version of the ActiveX component.</td>
</tr>
</tbody>
</table>
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.

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

### Methods That Parse FDF Data

The following functions parse FDF data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FDFExtractAppendSaves</code></td>
<td>Extracts incremental changes submitted within an FDF, and makes a new file out of it.</td>
</tr>
<tr>
<td><code>FDFExtractAttachment</code></td>
<td>Extracts an uploaded file and creates a new file out of it.</td>
</tr>
<tr>
<td><code>FDFGetAP</code></td>
<td>Gets the appearance of a field (/AP) and creates a PDF document out of it.</td>
</tr>
<tr>
<td><code>FDFGetFile</code></td>
<td>Gets the value of the /F key.</td>
</tr>
<tr>
<td><code>FDFGetFlags</code></td>
<td>Gets the flags of a field (/F or /F keys).</td>
</tr>
</tbody>
</table>
Methods That Generate FDF Data

The following functions generate FDF data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFAddDocJavaScript</td>
<td>Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.</td>
</tr>
<tr>
<td>FDFAddTemplate</td>
<td>Adds a template to an FDF file.</td>
</tr>
<tr>
<td>FDFSetAP</td>
<td>Sets the appearance of a button field (/AP key) from a PDF document.</td>
</tr>
<tr>
<td>FDFSetAPRef</td>
<td>Sets a reference to a PDF document to use for the appearance of a field (one of the faces within the /APRef key).</td>
</tr>
<tr>
<td>FDFSetAS</td>
<td>Sets the /AS key.</td>
</tr>
<tr>
<td>FDFSetPDFVersion</td>
<td>Sets the FDF version.</td>
</tr>
<tr>
<td>FDFSetFile</td>
<td>Sets the value of the /F key (string specification).</td>
</tr>
<tr>
<td>FDFSetFlags</td>
<td>Sets the value of one of the /Ff, /F, /SetFf, /ClrFf, /SetF, or /ClrF flags of a field.</td>
</tr>
<tr>
<td>FDFSetGoToAction</td>
<td>Sets /A or /AA key to action of type GoTo.</td>
</tr>
<tr>
<td>FDFSetGoToRAction</td>
<td>Sets /A or /AA key to action of type GoToR.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FDFSetHideAction</td>
<td>Sets /A or /AA key to action of type Hide.</td>
</tr>
<tr>
<td>FDFSetID</td>
<td>Sets the value of one element in the /ID key.</td>
</tr>
<tr>
<td>FDFSetIF</td>
<td>Sets the Icon Fit attribute for the appearance of a button field (/IF key).</td>
</tr>
<tr>
<td>FDFSetImportDataAction</td>
<td>Sets /A or /AA key to action of type ImportData.</td>
</tr>
<tr>
<td>FDFSetJavaScriptAction</td>
<td>Sets /A or /AA key to action of type JavaScript.</td>
</tr>
<tr>
<td>FDFSetNamedAction</td>
<td>Sets /A or /AA key to a named action.</td>
</tr>
<tr>
<td>FDFSetOnImportJavaScript</td>
<td>Adds a script to the FDF that will execute when it is imported.</td>
</tr>
<tr>
<td>FDFSetOpt</td>
<td>Sets the value of one element in a field's /Opt key.</td>
</tr>
<tr>
<td>FDFSetResetFormAction</td>
<td>Sets /A or /AA key to type action of ResetForm.</td>
</tr>
<tr>
<td>FDFSetRichValue</td>
<td>Sets the value of the /RV key.</td>
</tr>
<tr>
<td>FDFSetStatus</td>
<td>Sets the value of the /Status key.</td>
</tr>
<tr>
<td>FDFSetSubmitFormAction</td>
<td>Sets /A or /AA key to type action of SubmitForm.</td>
</tr>
<tr>
<td>FDFSetTargetFrame</td>
<td>Sets the value of the /Target key.</td>
</tr>
<tr>
<td>FDFSetURIAction</td>
<td>Sets /A or /AA key to type action of URL.</td>
</tr>
<tr>
<td>FDFSetValue</td>
<td>Sets the value of a field (/V key).</td>
</tr>
<tr>
<td>FDFSetValue</td>
<td>Sets the value of a field to an array.</td>
</tr>
</tbody>
</table>
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 `PDFGetValue`).

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:


The Perl libraries for Windows were tested under Perl 5.6 (see [http://www.ActiveState.com](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:

```perl
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,

```perl
$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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmbedAndClose</td>
<td>Embeds FDF in another document and then closes it.</td>
</tr>
<tr>
<td>GetVersion</td>
<td>Gets the current version of the FDF Toolkit library.</td>
</tr>
<tr>
<td>RemoveItem</td>
<td>Removes a key-value pair from the FDF data.</td>
</tr>
<tr>
<td>Save</td>
<td>Writes out an FDF file.</td>
</tr>
</tbody>
</table>

**Methods that Parse FDF Data**

The following methods parse FDF data.

**TABLE 5.2**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtractAppendSaves</td>
<td>Extracts incremental changes submitted within an FDF, and makes a new file out of it.</td>
</tr>
<tr>
<td>ExtractAttachment</td>
<td>Extracts an uploaded file and creates a new file out of it.</td>
</tr>
<tr>
<td>GetAP</td>
<td>Gets the appearance of a field (\texttt{/AP}) and creates a PDF document out of it.</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>Gets the value of the FDF \texttt{/Encoding} key as a string.</td>
</tr>
<tr>
<td>GetFDFVersion</td>
<td>Gets the current version of the FDF.</td>
</tr>
<tr>
<td>GetFile</td>
<td>Gets the value of the \texttt{/F} key.</td>
</tr>
<tr>
<td>GetFlags</td>
<td>Gets the flags of a field (\texttt{/Ff} or \texttt{/F} keys).</td>
</tr>
<tr>
<td>GetID</td>
<td>Gets the value of one element in the FDF's \texttt{/ID} key.</td>
</tr>
<tr>
<td>GetNthValue</td>
<td>Gets an element from a field's value if it is an array.</td>
</tr>
<tr>
<td>GetOpt</td>
<td>Gets the value of one element in a field's \texttt{/Opt} array.</td>
</tr>
</tbody>
</table>
Methods That Generate FDF Data

The following methods generate FDF data.

**TABLE 5.3**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDocJavaScript</td>
<td>Adds a document-level JavaScript to an FDF, which is added to a document when the FDF is imported into it.</td>
</tr>
<tr>
<td>AddTemplate</td>
<td>Adds a template to an FDF file.</td>
</tr>
<tr>
<td>new</td>
<td>Creates a new FDF object.</td>
</tr>
<tr>
<td>newFromBuf</td>
<td>Creates a new FDF object from a buffer.</td>
</tr>
<tr>
<td>newFromEmbedded</td>
<td>Creates a new FDF object from FDF data embedded in a container.</td>
</tr>
<tr>
<td>SetAP</td>
<td>Sets the appearance of a button field (AP key) from a PDF document.</td>
</tr>
<tr>
<td>SetAPRef</td>
<td>Sets a reference to a PDF document to use for the appearance of a field (one of the faces within the APRef key).</td>
</tr>
<tr>
<td>SetAS</td>
<td>Sets the AS key.</td>
</tr>
<tr>
<td>SetEncoding</td>
<td>Sets the value of the Encoding key.</td>
</tr>
<tr>
<td>SetFDFVersion</td>
<td>Sets the FDF version.</td>
</tr>
<tr>
<td>SetFile</td>
<td>Sets the value of the F key (string specification).</td>
</tr>
<tr>
<td>SetFileEx</td>
<td>Sets the value of the F key (complex file specification).</td>
</tr>
</tbody>
</table>
The following example shows how to use some of the Perl methods to generate FDF:

### Table 5.3

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SetFlags</strong></td>
<td>Sets the value of one of the <code>/Ff</code>, <code>/F</code>, <code>/SetFf</code>, <code>/ClrFf</code>, <code>/SetF</code>, or <code>/ClrF</code> flags of a field.</td>
</tr>
<tr>
<td><strong>SetGoToAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>GoTo</code>.</td>
</tr>
<tr>
<td><strong>SetGoToRAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>GoToR</code>.</td>
</tr>
<tr>
<td><strong>SetHideAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>Hide</code>.</td>
</tr>
<tr>
<td><strong>SetID</strong></td>
<td>Sets the value of one element in the <code>/ID</code> key.</td>
</tr>
<tr>
<td><strong>SetIF</strong></td>
<td>Sets the Icon Fit attribute for the appearance of a button field (<code>/IF</code> key).</td>
</tr>
<tr>
<td><strong>SetImportDataAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>ImportData</code>.</td>
</tr>
<tr>
<td><strong>SetJavaScriptAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>JavaScript</code>.</td>
</tr>
<tr>
<td><strong>SetNamedAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to a named action.</td>
</tr>
<tr>
<td><strong>SetOnImportJavaScript</strong></td>
<td>Adds a script to the FDF that will execute when it is imported.</td>
</tr>
<tr>
<td><strong>SetOpt</strong></td>
<td>Sets the value of one element in a field's <code>/Opt</code> key.</td>
</tr>
<tr>
<td><strong>SetResetByNameAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to action of type <code>ResetForm</code> using field names.</td>
</tr>
<tr>
<td><strong>SetResetFormAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to type action of <code>ResetForm</code>.</td>
</tr>
<tr>
<td><strong>SetRichValue</strong></td>
<td>Sets the value of the <code>/RV</code> key.</td>
</tr>
<tr>
<td><strong>SetStatus</strong></td>
<td>Sets the value of the <code>/Status</code> key.</td>
</tr>
<tr>
<td><strong>SetSubmitByNameAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to type action of <code>SubmitForm</code> using field names.</td>
</tr>
<tr>
<td><strong>SetSubmitFormAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to type action of <code>SubmitForm</code>.</td>
</tr>
<tr>
<td><strong>SetTargetFrame</strong></td>
<td>Sets the value of the <code>/Target</code> key.</td>
</tr>
<tr>
<td><strong>SetURIAction</strong></td>
<td>Sets <code>/A</code> or <code>/AA</code> key to type action of <code>URI</code>.</td>
</tr>
<tr>
<td><strong>SetValue</strong></td>
<td>Sets the value of a field (<code>/V</code> key).</td>
</tr>
<tr>
<td><strong>SetValues</strong></td>
<td>Sets the value of a field to an array.</td>
</tr>
</tbody>
</table>
**EXAMPLE 5.1 Using Perl Methods To Generate FDF Data**

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

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

/* or */
$inPdf = new Acrobat::FDF('-','ENV{CONTENT_LENGTH}');
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.
FDF Toolkit For Java

Other Useful Documentation

- 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.
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:

```c
FDRErc FDFClose (FDFDoc theFDF);
```

```c
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 \texttt{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 \texttt{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 \texttt{GetEncoding} and \texttt{SetEncoding}.

**Data Types**

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

<table>
<thead>
<tr>
<th>Data structure/enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error Codes</strong></td>
<td>Error codes.</td>
</tr>
<tr>
<td>\texttt{FDFItem}</td>
<td>Key values.</td>
</tr>
<tr>
<td>\texttt{FDFAppFace}</td>
<td>Face values.</td>
</tr>
<tr>
<td>\texttt{FDFScaleWhen}</td>
<td>Scaling option values.</td>
</tr>
<tr>
<td>\texttt{PDFActionTrigger}</td>
<td>Action Trigger values.</td>
</tr>
<tr>
<td>\texttt{pdfFileSpec}</td>
<td>PDF file specification.</td>
</tr>
</tbody>
</table>

**Error Codes**

In C/C++ and Perl most functions return error codes of type \texttt{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.

<table>
<thead>
<tr>
<th>Error</th>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFErcOK</td>
<td>0</td>
<td>The function returned successfully.</td>
</tr>
<tr>
<td>FDFErcInternalError</td>
<td>1</td>
<td>An internal FDF Library error occurred.</td>
</tr>
<tr>
<td>FDFErcBadParameter</td>
<td>2</td>
<td>One or more of the parameters passed to the function are invalid.</td>
</tr>
<tr>
<td>FDFErcFileSysErr</td>
<td>3</td>
<td>A file system error occurred, including “file not found”.</td>
</tr>
<tr>
<td>FDFErcBadFDF</td>
<td>4</td>
<td>The FDF file being opened or parsed is invalid.</td>
</tr>
<tr>
<td>FDFErcFieldNotFound</td>
<td>5</td>
<td>The field whose name was passed in the parameter fieldName does not exist in the FDF file.</td>
</tr>
<tr>
<td>FDFErcNoValue</td>
<td>6</td>
<td>The field whose value was requested has no value.</td>
</tr>
<tr>
<td>FDFErcEnumStopped</td>
<td>7</td>
<td>Enumeration was stopped by FDFEnumValuesProc by returning false. (C only).</td>
</tr>
<tr>
<td>FDFErcCantInsertField</td>
<td>8</td>
<td>The field whose name was passed in the parameter fieldName 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”.</td>
</tr>
<tr>
<td>FDFErcNoOption</td>
<td>9</td>
<td>The requested element in a field’s /Opt key does not exist, or the field has no /Opt key.</td>
</tr>
<tr>
<td>FDFErcNoFlags</td>
<td>10</td>
<td>The field has no /F or /Ff keys.</td>
</tr>
<tr>
<td>FDFErcBadPDF</td>
<td>11</td>
<td>The PDF file passed as the parameter to FDFSetAP is invalid, or does not contain pageNum.</td>
</tr>
<tr>
<td>FDFErcBufTooShort</td>
<td>12</td>
<td>The buffer passed as a parameter is too short for the length of the data that the function wants to return.</td>
</tr>
</tbody>
</table>
## Data Types

### FDFItem

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

<table>
<thead>
<tr>
<th>C, ActiveX</th>
<th>Perl</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFvalue</td>
<td>value</td>
<td>Flag for the <code>/Value</code> key.</td>
</tr>
<tr>
<td>FDFStatus</td>
<td>Status</td>
<td>Flag for the <code>/Status</code> key.</td>
</tr>
<tr>
<td>FDFFile</td>
<td>File</td>
<td>Flag for the <code>/File</code> key.</td>
</tr>
<tr>
<td>FDFID</td>
<td>ID</td>
<td>Flag for the <code>/ID</code> key.</td>
</tr>
<tr>
<td>FDDOpt</td>
<td>Opt</td>
<td>Flag for the <code>/Opt</code> key.</td>
</tr>
<tr>
<td>FDFFf</td>
<td>Ff</td>
<td>Flag for the <code>/Ff</code> key.</td>
</tr>
<tr>
<td>FDFSethFf</td>
<td>SetFf</td>
<td>Flag for the <code>/SetFf</code> key.</td>
</tr>
</tbody>
</table>

### Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFErcNoAP</td>
<td>The field has no <code>/AP</code> key.</td>
</tr>
<tr>
<td>FDFErcIncompatibleFDF</td>
<td>An attempt to mix classic and template-based FDF files was made.</td>
</tr>
<tr>
<td>FDFErcNoAppendSaves</td>
<td>The FDF does not include a <code>/Differences</code> key.</td>
</tr>
<tr>
<td>FDFErcValueIsArray</td>
<td>The value of this field is an array. Use <code>FDFGetNthValue</code>.</td>
</tr>
<tr>
<td>FDFErcEmbeddedFDFs</td>
<td>The FDF you passed as a parameter is a container for one or more FDFs embedded within it. Use <code>FDFOpenFromEmbedded</code> to gain access to each embedded FDF.</td>
</tr>
<tr>
<td>FDFErcNoMoreFDFs</td>
<td>Returned by <code>FDFOpenFromEmbedded</code> when parameter <code>iWhich</code> &gt;= the number of embedded FDFs (including the case when the passed FDF does not contain any embedded FDFs).</td>
</tr>
<tr>
<td>FDFErcInvalidPassword</td>
<td>Returned by <code>FDFOpenFromEmbedded</code> when the embedded FDF is encrypted, and the correct password is not provided.</td>
</tr>
<tr>
<td>FDFErcLast</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>C, ActiveX</td>
<td>Perl</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FDFClearFf</td>
<td>ClearFf</td>
</tr>
<tr>
<td>FDFFlags</td>
<td>Flags</td>
</tr>
<tr>
<td>FDFSfSet</td>
<td>SetF</td>
</tr>
<tr>
<td>FDFClrF</td>
<td>ClrF</td>
</tr>
<tr>
<td>FDFAP</td>
<td>AP</td>
</tr>
<tr>
<td>FDFAS</td>
<td>AS</td>
</tr>
<tr>
<td>FDFAction</td>
<td>Action</td>
</tr>
<tr>
<td>FDFAA</td>
<td>AA</td>
</tr>
<tr>
<td>FDFAPRef</td>
<td>APRef</td>
</tr>
<tr>
<td>FDFIF</td>
<td>IF</td>
</tr>
<tr>
<td>FDFTargetFrame</td>
<td>TargetFrame</td>
</tr>
<tr>
<td>FDFEncoding</td>
<td>Encoding</td>
</tr>
<tr>
<td>FDFJavaScript</td>
<td>FDFJavaScript</td>
</tr>
<tr>
<td>FDFAppendSaves</td>
<td>AppendSaves</td>
</tr>
</tbody>
</table>

**Used in Functions**

- FDFSfGetFlags
- FDFSfRemoveItem
- FDFSfSetFlags
- FDFSfSetTargetFrame
- FDFSfGetEncoding
- FDFSfSetAS
- FDFSfExtractAppendSaves
**FDFAppFace**

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

*TABLE 7.1*

<table>
<thead>
<tr>
<th>C, ActiveX</th>
<th>Perl</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFNormalAP</td>
<td>NormalAP</td>
<td>Normal /AP face (/N key)</td>
</tr>
<tr>
<td>FDFRolloverAP</td>
<td>RolloverAP</td>
<td>Rollover /AP face (/R key)</td>
</tr>
<tr>
<td>FDFDownAP</td>
<td>DownAP</td>
<td>Down /AP face (/D key)</td>
</tr>
</tbody>
</table>

**Used in Functions**

- FDFGetAP
- FDFSsetAP
- FDFSsetAPRef

**FDFScaleWhen**

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

*TABLE 7.2*

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

**Used in Functions**

- FDFSsetIF

**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
FDF toolkit Reference

Data Types

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**

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

**Perl Syntax**

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

**ActiveX Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>cScriptName</th>
<th>Name of the script. Must be in either PDFDocEncoding or Unicode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cScript</td>
<td>The script text. Must be in either PDFDocEncoding or Unicode.</td>
</tr>
<tr>
<td></td>
<td>Hint: use &quot;\r&quot; as line separator within the script. Use &quot;\t&quot; for tabs.</td>
</tr>
</tbody>
</table>

**Errors**

FDFErcIncompatibleFDF, FDFErcEmbeddedFDFs, FDFErcInternalError

**Related Functions**

FDFSetOnImportJavaScript

**C Example**

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

**ActiveX Example**

```active
objFdf.FDFAddDocJavaScript "PlusOne", "function PlusOne(x){\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

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

ActiveX Syntax

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

Perl Syntax

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bNewPage</td>
<td>If <strong>true</strong>, the template is used to start a new page. If <strong>false</strong>, it is appended to the last page. If this is the first template added to the FDF data, this parameter is ignored. 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.</td>
</tr>
</tbody>
</table>

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

C/C++: A `pdfFileSpec` structure containing the file specification for the PDF where the template is stored. This is the value for the `/F` key within the `/TRef` entry.

If `fileSpec` is `NULL`, 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.

If `fileSpec` is not `NULL`, fill in the fields of the `pdfFileSpec` data structure that you care about with null-terminated strings, and set the other fields to `NULL`. In the most common case where only the `/F` field is used, a convenience macro, `SIMPLE_FILESPEC`, is available.

Perl, ActiveX: This value is a simply string containing the pathname. The other fields of `pdfFileSpec` are not used.

**templateName**

String containing the name of the template within the PDF file specified by `fileSpec`.

C, Perl: Should be in PDFDocEncoding or Unicode.

**bRename**

If `bRename` is `false`, a `/Rename` key whose value is `false` is added, and renaming does not occur. If `bRename` is `true`, the `/Rename` key is not added because the default value is `true`; renaming occurs. The purpose of the `/Rename` key is to provide the flexibility of renaming fields as templates get spawned. This prevents potential conflicts between field names originating in different templates.

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.

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.”

---

**Errors**

`FDFErcBadParameter`, `FDFErcBadFDF`, `FDFErcIncompatibleFDF`, `FDFErcEmbeddedFDFs`, `FDFErcInternalError`
Related Functions

**FDFSetValu**e

**C Example**

```c
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**

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

**ActiveX Syntax**

```c
Function FDFClose()
```

**Errors**

- **FDFErcBadParameter** *(C only)*

**Related Functions**

- **FDFCreate**
- **FDFOpen**
- **FDFFinalize**

**C Example**

```c
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**

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

**ActiveX Syntax**

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

**Parameters**

| pTheFDF | A pointer to the FDFDoc. Other functions in the API use the pointer created here. |

**Errors**

**FDFErcBadParameter, FDFErcInternalError**

**Related Functions**

**FDFClose**

**C Example**

```c
FDFDoc theFDF;
FDFErc fdfErC;

fdfErC = FDFCreate (&theFDF);
```

**ActiveX Example**

```vbs
'The Dim statement does not apply to VBScript,
'which only supports Variant
Dim objPdf As FDFACXLib.PdfDoc

'Create the FDF
Set objPdf = 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theContainerFDF</td>
<td>The container that holds one or more FDFs (C only)</td>
</tr>
<tr>
<td>theEmbeddedFDF</td>
<td>The FDF to embed. It is automatically closed with this call unless FDFErcBadParameter is returned.</td>
</tr>
<tr>
<td>cPassword</td>
<td>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.</td>
</tr>
</tbody>
</table>

Errors

FDFErcBadParameter, FDFErcIncompatibleFDF, FDFErcEmbeddedFDFs, FDFErcInternalError
Related Functions

FDFClose
FDFSetFile
FDFSetID

C Example

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

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

**Parameters**

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

**Errors**

- FDFErcBadParameter, FDFErcEnumStopped, FDFErcBadFDF, FDFErcIncompatibleFDF, FDFErcBufTooShort, FDFErcInternalError

**Related Functions**

- FDFNextFieldName
FDF Toolkit Reference

FDFEnumValues

C Example

```c
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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFileName</td>
<td>Host-encoded string for the pathname of the file to create with the extracted bytes.</td>
</tr>
</tbody>
</table>

**Errors**

FDFErcBadParameter, FDFErcIncompatibleFDF, FDFErcEmbeddedFDFs, FDFErcFileSysErr, FDFErcNoAppendSaves, FDFErcInternalError

**Related Functions**

FDFGetFDFVersion

**C Example**

```c
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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cFieldName</strong></td>
<td>String representing the fully qualified name of the field.</td>
</tr>
<tr>
<td><strong>cFileName</strong></td>
<td>Host-encoded string for the pathname of the file to create with the extracted attachment. If <strong>cFileName</strong> is NULL or an empty string, Acrobat saves the file in the current folder, using the filename included with the attachment. If <strong>bIsFilePath</strong> is true and <strong>cFileName</strong> is not empty, Acrobat appends the filename included with the attachment to <strong>cFileName</strong> 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>.</td>
</tr>
<tr>
<td><strong>nFileName</strong> (C only)</td>
<td>If greater than zero, and <strong>cFileName</strong> is not NULL, <strong>FDFExtractAttachment</strong> fills <strong>cFileName</strong> with the full pathname (up to <strong>nFileName</strong> bytes) where the attachment was saved (only if <strong>FDFExtractAttachment</strong> returns <strong>FDFErcOK</strong>).</td>
</tr>
</tbody>
</table>
Table 7:

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

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

```c
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**

```c
FDFFinalize (void);
```

**Errors**

None except for FDFFinalize

**Related Functions**

- FDFFinalize
- FDFClose

**C Example**

```c
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

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

ActiveX Syntax

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

Perl Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field.</td>
</tr>
<tr>
<td>whichFace</td>
<td>A value indicating which face of the /AP key is requested. See <code>PDFAppFace</code> for possible values: FDFNormalAP, FDFRolloverAP, PDFDownAP</td>
</tr>
<tr>
<td>fileName</td>
<td>Host-encoded string for the pathname of the PDF file to create.</td>
</tr>
</tbody>
</table>

Errors

- FDFErcBadParameter
- FDFErcBadFDF
- FDFErcEmbeddedFDFs
- FDFErcIncompatibleFDF
- FDFErcFieldNotFound
- FDFErcFileSizeErr
- FDFErcNoAP
- FDFErcInternalError

Related Functions

- FDFSetAP
- FDFSetAPRef

C Example

```c
/* 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

```vbscript
Const FDFNormalAP = 0
Const FDFRolloverAP = 1
Const PDFDownAP = 2
objPdf.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buffer</code> (C only)</td>
<td><em>(Filled by the function)</em> The value of the FDF’s <code>/Encoding</code> key as a string. If the FDF file contains no <code>/Encoding</code> key, <code>buffer</code> will contain the empty string. If you pass <code>NULL</code>, the function will return the length of <code>/Encoding</code> in <code>nBytes</code>.</td>
</tr>
<tr>
<td><code>bufSize</code> (C only)</td>
<td>The maximum number of bytes that can be written into <code>buffer</code>. You must pass at least the length + 1, since the routine adds a ‘\0’ terminator. If <code>bufSize</code> is too small, the function returns <code>FDFErcBufTooShort</code>. If <code>buffer</code> is <code>NULL</code>, then <code>bufSize</code> is unused.</td>
</tr>
<tr>
<td><code>nBytes</code> (C only)</td>
<td><em>(Filled by the function)</em> The length of the <code>/Encoding</code> key, not including the null terminator. If there is an error other than <code>FDFErcBufTooShort</code> or <code>FDFErcOk</code>, zero is returned.</td>
</tr>
</tbody>
</table>

**Return Value**

- C: Error code
- Perl: String containing the encoding flavor.

**Errors**

`FDFErcBadParameter, FDFErcEmbeddedFDFs, FDFErcIncompatibleFDF, FDFErcBufTooShort, FDFErcInternalError`

**Related Functions**

`FDFSetEncoding`
C Example

```c
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 (PDFDoc thePDF);

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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

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

**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

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

ActiveX Example

```c
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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
<tr>
<td><strong>whichFlags</strong></td>
<td>Value specifying which flag to get; must be either <code>FDFFf</code> or <code>FDFFlags</code> (see <code>FDFItem</code>).</td>
</tr>
<tr>
<td><strong>pTheFlags</strong></td>
<td><em>(Filled by the function)</em> if <code>FDFGetFlags</code> returns <code>FDFErcOK</code>, <code>pTheFlags</code> points to the requested flags.</td>
</tr>
</tbody>
</table>

**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

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

ActiveX Example

```c
Const FDFFf = 5
Const FDFFlags = 8
lngAnnotFlags = objPdf.FDFGetFlags ("employee.name.last", FDFFlags)
```
**FDFGetID**

*(Perl)* **GetID**

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

**C Syntax**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| nElemNum  | The element number to get in the FDF’s /ID key, either:  
* 0: the permanent ID  
* 1: the changing ID |
| buffer    | *(Filled by the function)* Buffer containing the requested ID element. If you pass NULL, the function returns the length of the ID element in nBytes.  
**Note:** This is not a null-terminated string. |
| bufSize   | *(C only)* The maximum number of bytes that can be written into buffer. If bufSize is too small, the function returns FDFErcBufTooShort.  
If buffer is NULL, then bufSize is unused. |
| nBytes    | *(C only)* *(Filled by the function)* The length of the ID element in bytes. If there is an error other than FDFErcBufTooShort or FDFErcOK, 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 (thePDF, 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>
<thead>
<tr>
<th>fieldName</th>
<th>String representing the fully qualified name of the field (for example, employee.name.last).</th>
</tr>
</thead>
<tbody>
<tr>
<td>iWhich</td>
<td>Zero-based index of the desired value. Ignored if the field's value is not an array.</td>
</tr>
<tr>
<td>buffer</td>
<td><em>(Filled by the function)</em> The value of the field. If the FDF includes an Encoding key (see FDFGetEncoding), 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 NULL, the function will return the length of the field in nBytes.</td>
</tr>
<tr>
<td>bufSize</td>
<td>The maximum number of bytes that can be written into buffer. 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 bufSize is too small, the function returns FDFErcBufTooShort. If buffer is NULL, then bufSize is unused.</td>
</tr>
<tr>
<td>nBytes</td>
<td><em>(Filled by the function)</em> The length of the value corresponding to fieldName, not including the null terminator. If there is an error other than FDFErcBufTooShort or FDFErcOK, zero is returned.</td>
</tr>
</tbody>
</table>

### 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

```c
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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
<tr>
<td><strong>nElemNum</strong></td>
<td>Integer representing the element number to get from the field's <code>/Opt</code> array. The first element in the <code>/Opt</code> array has an index of zero. If this parameter is <code>-1</code>, the function returns in <code>nRet</code> the number of elements in the field's <code>/Opt</code> array, and the parameters <code>buffer1</code>, <code>buffer2</code>, and <code>bufSize</code> remain unused.</td>
</tr>
<tr>
<td><strong>buffer1 (C only)</strong></td>
<td><em>(Filled by the function)</em> If the <code>/Opt</code> array element requested is itself composed of an array containing two strings, <code>buffer1</code> contains the first string (which is the export value). See <code>FDFGetEncoding</code> for a discussion of encodings. If the <code>/Opt</code> array element requested is a single string, <code>buffer1</code> contains that string (which is the item name). If <code>buffer1</code> is <code>NULL</code>, the function returns in <code>nRet</code> the number of bytes required.</td>
</tr>
<tr>
<td><strong>buffer2 (C only)</strong></td>
<td><em>(Filled by the function)</em> If the <code>/Opt</code> array element requested is itself composed of an array containing two strings, <code>buffer2</code> contains the second string (which is the item name). Otherwise, <code>buffer2</code> contains the empty string. If <code>buffer1</code> is <code>NULL</code>, <code>buffer2</code> remains unused.</td>
</tr>
<tr>
<td><strong>bufSize (C only)</strong></td>
<td>The maximum number of bytes written into <code>buffer1</code> and <code>buffer2</code> (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 <code>buffer1</code> is <code>NULL</code>, <code>bufSize</code> remains unused.</td>
</tr>
<tr>
<td><strong>nRet (C only)</strong></td>
<td><em>(Filled by the function)</em> If <code>nElemNum</code> is <code>-1</code>, the number of elements in the field's <code>/Opt</code> array is returned here. Otherwise, <code>nret</code> will be the number of bytes copied into <code>buffer1</code> (excluding the terminating <code>NULL</code>), if <code>buffer1</code> is not <code>NULL</code>, or the number of bytes that <code>buffer1</code> and <code>buffer2</code> need to be (if <code>buffer1</code> is <code>NULL</code> or <code>FDFGetOpt</code> returns <code>FDFErcBufTooShort</code>). If there is an error other than <code>FDFErcBufTooShort</code> or <code>FDFErcOK</code>, zero is returned.</td>
</tr>
</tbody>
</table>
Table 13:

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

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

ActiveX Example

```c
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**

| fieldName | String representing the fully-qualified name of the field (for example, `employee.name.last`) |

**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**

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

(Perl) GetRichValue

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

C Syntax

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

ActiveX Syntax

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

Perl Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Table 15:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
</tr>
<tr>
<td><strong>buffer</strong> <em>(C only)</em></td>
</tr>
<tr>
<td><strong>bufSize</strong> <em>(C only)</em></td>
</tr>
<tr>
<td><strong>nBytes</strong> <em>(C only)</em></td>
</tr>
</tbody>
</table>
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:

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

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

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

ActiveX Example

```c
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**

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

**ActiveX Syntax**

Function FDFGetValue (fieldName As String) As String

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, <em>employee.name.last</em>).</td>
</tr>
<tr>
<td>buffer</td>
<td><em>(C only)</em> <em>(Filled by the function)</em> The value of the field. If the FDF includes an /Encoding key (see <strong>FDFGetEncoding</strong>), 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 <strong>NULL</strong>, the function will return the length of the value <strong>nBytes</strong>.</td>
</tr>
<tr>
<td>bufSize</td>
<td><em>(C only)</em> The maximum number of bytes that can be written into <strong>buffer</strong>. You must pass at least the length + 2, since the routine adds a null terminator, which is 2 bytes for Unicode. If <strong>bufSize</strong> is too small, the function returns <strong>FDFErcBufTooShort</strong>. If <strong>buffer</strong> is <strong>NULL</strong>, then <strong>bufSize</strong> is unused.</td>
</tr>
<tr>
<td>nBytes</td>
<td><em>(C only)</em> <em>(Filled by the function)</em> The length of the value, not including the null terminator. If there is an error other than <strong>FDFErcBufTooShort</strong> or <strong>FDFErcOK</strong>, zero is returned.</td>
</tr>
</tbody>
</table>

**Return Value**

* C: Error code
FDF Toolkit Reference

FDFGetValue

*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**

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

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

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

**ActiveX Example**

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

*(Perl) GetVersion*

Gets the current version of the FDF Toolkit.

**C Syntax**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

None

**Return Value**

A string containing the current version of the FDF Toolkit.

**C Example**

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

**ActiveX Example**

```c
strVersion = FdfAcX.FDFGetVersion
```
FDF Toolkit Reference
FDFInitialize

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
FDFFFinalize

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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

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

**Return Value**

*C:* error code
**FDF Toolkit Reference**

**FDFNextFieldName**

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

**Errors**

- `FDFErcBadParameter`, `FDFErcBadFDF`, `FDFErcEmbeddedFDFs`, `FDFErcIncompatibleFDF`, `FDFErcFieldNotFound`, `FDFErcBufTooShort`, `FDFErcInternalError`

**Related Functions**

- `FDFEnumValues`

**C Example**

```c
FDFDoc theFDF;
FDFErc fdfErC;
char cBuf[255];
ASInt32 nBytes;

fdfErC = FDFNextFieldName (theFDF, NULL, cBuf,sizeof(cBuf), &nBytes);
```

**ActiveX Example**

```c
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**

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

**Parameters**

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

**Errors**

`FDFErcBadParameter, FDFErcFileSysErr, FDFErcBadFDF, FDFErcInternalError`

**Related Functions**

`FDFClose`

**C Examples**

```c
/* 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

```c
/* 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 `FDFA CXLib.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:**

| varFdfData   | An array of bytes holding the contents of the FDF data. |

#### Return Value

An object of type `FDFA CXLib.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 objPdf As FDFACXLib.PdfDoc

Set objPdf = PdfAcX.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

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

### ActiveX Syntax

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

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>theContainerFDF</strong></td>
<td>A container that holds one or more FDFs</td>
</tr>
<tr>
<td>(C only)</td>
<td></td>
</tr>
<tr>
<td><strong>iWhich</strong></td>
<td>Zero-based index of the embedded FDF that should be opened.</td>
</tr>
<tr>
<td><strong>cPassword</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>pTheEmbeddedFDF</strong></td>
<td>If this function returns <code>FDFErcOK</code>, <code>pTheEmbeddedFDF</code> points to the opened <code>FDFDoc</code>, which can then be used for other API calls.</td>
</tr>
<tr>
<td>(C only)</td>
<td></td>
</tr>
</tbody>
</table>

### 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

```c
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**

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

**Parameters**

| `bstrFileName` | 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bstrFdfData</td>
<td>String containing the contents of the FDF data.</td>
</tr>
</tbody>
</table>

**Return Value**

An object of type FDFACXLib.FdfDoc.

**Exceptions**

FDFErcFileSysErr, FDFErcBadFDF, FDFErcInternalError

**Related Functions**

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockProc</td>
<td><code>ThreadsafeCallback</code> used for locking the mutex operation.</td>
</tr>
<tr>
<td>unlockProc</td>
<td><code>ThreadsafeCallback</code> used for unlocking the mutex operation.</td>
</tr>
<tr>
<td>destroyProc</td>
<td><code>ThreadsafeCallback</code> used for destroying the mutex operation.</td>
</tr>
<tr>
<td>clientData</td>
<td>Data passed through to each of the operations. Typically the data is initialized by the user before calling <code>FDFRegisterThreadsafeCallbacks</code> by a call to a mutex initialize operation like Sun Solaris' <code>mutex_init</code>.</td>
</tr>
</tbody>
</table>

Errors

None except for `FDFErcOK`.

Related Functions

- `FDFInitialize`
- `FDFFinalize`

C Example

```c
/* Definitions needed only on Unix systems for multithreading. */
#define 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

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

**Errors**

FDFErcBadParameter, FDFErcBadFDF, FDFErcEmbeddedFDFs, FDFErcIncompatibleFDF, FDFErcFieldNotFound, FDFErcInternalError

**C Example**

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

**ActiveX Example**

```c
Const FDFFile = 2
thePdf.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**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>Complete pathname (in Host encoding), or &quot;-&quot; to write to <strong>stdout</strong>.</td>
</tr>
</tbody>
</table>

**Errors**

- **FDFErcBadParameter**, **FDFErcFileSysErr**, **FDFErcInternalError**

**Related Functions**

- **FDFClose**
- **FDFOpen**

**C Example**

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

*(ActiveX only)*

Returns an array of unsigned bytes (Variant of type \texttt{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**

\texttt{FDFErcFileSysErr, FDFErcInternalError}

**Related Functions**

- FDFSavetoFile
- FDFSaveToStr

**ActiveX Example**

```vbnet
Response.ContentType = "application/Vnd.fdf"
Response.BinaryWrite thePdf.FDFSaveToBuf
```
FDF SavetoFile

*(ActiveX only)*

Saves FDF data to a file.

**ActiveX Syntax**

Sub FDFSavetoFile (fileName As String)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>String containing the pathname where the FDF file should be saved. The FDF data cannot be saved to a file that was opened with <code>FDFOpenFromFile</code> and has not yet been closed with <code>FDFClose</code>.</td>
</tr>
</tbody>
</table>

**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**

```vba
Function FDFSaveToStr() As String
End Function
```

**Parameters**

None

**Return Value**

A string containing the FDF data.

**Errors**

[FDFErcFileSysErr, FDFErcInternalError, E_OUTOFMEMORY]

**Related Functions**

FDFSave

FDFSavetoFile

**ActiveX Example**

```vba
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

```
FDFErr 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

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

Table 28:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>Host-encoded string representing the fully qualified name of the field (for example, <em>employee.name.last</em>).</td>
</tr>
</tbody>
</table>
| **whichFace** | An value indicating which face is to be set. Possible values are `FDFNormalAP`, `FDFDownAP`, and `FDFRolloverAP` (see `FDFAppFace`).  
**Note:** Only the `FDFNormalAP` will be imported (and `FDFDownAP`, `FDFRolloverAP` will be ignored) unless the button that the `/AP` is being imported into has a “Highlight” of type “Push”. |
| **subAP** | String to indicate which sub-appearance is to be set. Currently, Acrobat does not import sub-appearances; therefore, you should pass `NULL` for this parameter. |
| **fileName** | String for the pathname of the PDF file representing the button’s appearance. |
| **pageNum** | 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

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

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

ActiveX Example

```activeX
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 an 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

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

Errors

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError

Related Functions

FDFSetAP

C Example

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

ActiveX Example

```c
Const FDFNormalAP = 0
Const PDFrolloverAP = 1
Const PDFDownAP = 2
objPdf.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>
<thead>
<tr>
<th>fieldName</th>
<th>String representing the fully qualified name of the field (for example, employee.name.last).</th>
</tr>
</thead>
<tbody>
<tr>
<td>newAS</td>
<td>String in PDFDocEncoding that will be converted to a PDF Name to use as the new value of /AS.</td>
</tr>
</tbody>
</table>

**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**

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

**Perl Syntax**

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

**Parameters**

**Table 31:**

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

**Errors**

PDFErcBadParameter, PDFErcEmbeddedFDFs, PDFErcInternalError

**Related Functions**

FDFGetEncoding

**C Example**

```c
PDFErc 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>cVersion</th>
<th>The FDF version that the document will be set to. Either &quot;1.2&quot;, &quot;1.3&quot;, &quot;1.4&quot;, or &quot;1.5&quot;.</th>
</tr>
</thead>
</table>

**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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>newFile</th>
<th>Host-encoded string to use as the new value of the /F key. The value of /F 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.</th>
</tr>
</thead>
</table>

**Errors**

FDFErcBadParameter, FDFErcIncompatiblePDF, FDFErcInternalError

**Related Functions**

FDFGetFile
FDFSetFileEx

**C Example**

```c
/* 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**

```c
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**

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

**Perl Syntax**

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

**Parameters**

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

**Errors**

`FDFErcBadParameter`, `FDFErcIncompatibleFDF`, `FDFErcInternalError`

**Related Functions**

`FDFGetFile`

`FDFSetFile`

**C Example**

```c
/* 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully qualified name of the field. (for example, employee.name.last).</td>
</tr>
<tr>
<td><strong>whichFlags</strong></td>
<td>An FDFItem value indicating the flag to set. Must be one of the following flags:</td>
</tr>
<tr>
<td></td>
<td>■ FDFFf sets the /Ff key.</td>
</tr>
<tr>
<td></td>
<td>■ FDFSetFf sets the /SetFf key.</td>
</tr>
<tr>
<td></td>
<td>■ FDFClearFf sets the /ClrFf key.</td>
</tr>
<tr>
<td></td>
<td>■ FDFFlags sets the /F key.</td>
</tr>
<tr>
<td></td>
<td>■ FDFSetF sets the /SetF key.</td>
</tr>
<tr>
<td></td>
<td>■ FDFClrF sets the /ClrF key.</td>
</tr>
<tr>
<td><strong>newFlags</strong></td>
<td>The new value for the flags.</td>
</tr>
</tbody>
</table>

**Errors**

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError
Related Functions
  
  FDFAddTemplate
  FDFGetFlags

C Example

```c
/*
   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 PDF 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

```vbnet
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).
objPdf.FDFSetFlags "employee.name.last", FDFSetF, 2

'Make "employee.name.last" visible
objPdf.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).
objPdf.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**

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

**ActiveX Syntax**

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

**Perl Syntax**

```perl
$outFDF->SetGoToAction($fieldName, $whichTrigger, $theDest, $isName);
```
FDF Toolkit Reference

FDFSetGoToAction

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, employee.name.last).</td>
</tr>
<tr>
<td>whichTrigger</td>
<td>An <strong>FDFActionTrigger</strong> value indicating the event trigger for the action:</td>
</tr>
<tr>
<td></td>
<td>● FDFEnter</td>
</tr>
<tr>
<td></td>
<td>● FDFExit</td>
</tr>
<tr>
<td></td>
<td>● FDFDown</td>
</tr>
<tr>
<td></td>
<td>● FDFUp</td>
</tr>
<tr>
<td></td>
<td>● FDFOnFocus</td>
</tr>
<tr>
<td></td>
<td>● FDFOnBlur</td>
</tr>
<tr>
<td></td>
<td>If FDFUp, an /A entry is used, otherwise an /AA entry is created.</td>
</tr>
<tr>
<td>theDest</td>
<td>The string, in PDFDocEncoding or Unicode, to be used as value for the /D key in the GoTo action.</td>
</tr>
<tr>
<td>isName</td>
<td>If true, Acrobat converts theDest to a PDF Name before making it the value of /D. If false, theDest is set to a PDF String.</td>
</tr>
</tbody>
</table>

Errors

-FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError

Related Functions

-FDFSetGoToRAction

C Example

```c
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 the FDF 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, employee.name.last).</td>
</tr>
<tr>
<td>whichTrigger</td>
<td>An FDFActionTrigger value indicating the event trigger for the action:</td>
</tr>
<tr>
<td></td>
<td>● FDFEnter</td>
</tr>
<tr>
<td></td>
<td>● FDFExit</td>
</tr>
<tr>
<td></td>
<td>● FDFDown</td>
</tr>
<tr>
<td></td>
<td>● FDFUp</td>
</tr>
<tr>
<td></td>
<td>● FDFOnFocus</td>
</tr>
<tr>
<td></td>
<td>● FDFOnBlur</td>
</tr>
<tr>
<td></td>
<td>If FDFUp, an /A entry is used, otherwise an /AA entry is created.</td>
</tr>
<tr>
<td>theDest</td>
<td>The string to be used as value for the /D key in the GoToR action.</td>
</tr>
<tr>
<td>isName</td>
<td>If true, theDest is converted to a PDF Name before making it the value of /D. If false, theDest is set to a PDF string.</td>
</tr>
<tr>
<td>theFile</td>
<td>The String to be used as value for the /F key in the GoToR action.</td>
</tr>
<tr>
<td>addNewWindow</td>
<td>If true, adds a /NewWindow key to the GoToR action and sets the value passed in newWindow parameter. If false, does not add it, and the newWindow parameter is ignored.</td>
</tr>
<tr>
<td>newWindow</td>
<td>Value for the /NewWindow key in the GoToR action. Only used if the addNewWindow parameter is set to true.</td>
</tr>
</tbody>
</table>

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 = FDFSetGoToRAction (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 the FDF is template-based (**FDFAddTemplate** has been called), **FDFSetHideAction** acts on the most recently added template.

**C Syntax**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

<table>
<thead>
<tr>
<th><strong>fieldName</strong></th>
<th>String representing the fully qualified name of the field (for example, employee.name.last).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>whichTrigger</strong></td>
<td>An <strong>FDFActionTrigger</strong> value indicating the event trigger for the action:</td>
</tr>
<tr>
<td></td>
<td>* FDFEnter</td>
</tr>
<tr>
<td></td>
<td>* FDFExit</td>
</tr>
<tr>
<td></td>
<td>* FDFDown</td>
</tr>
<tr>
<td></td>
<td>* FDFUp</td>
</tr>
<tr>
<td></td>
<td>* FDFOnFocus</td>
</tr>
<tr>
<td></td>
<td>* FDFOnBlur</td>
</tr>
<tr>
<td></td>
<td>If FDFUp, an /A entry is used, otherwise an /AA entry is created.</td>
</tr>
<tr>
<td><strong>theTarget</strong></td>
<td>The string in PDFDocEncoding to be used as value for the /T key in the Hide action.</td>
</tr>
<tr>
<td><strong>isHide</strong></td>
<td>This is the value of the /H key in the Hide action. If true, no /H key is added.</td>
</tr>
</tbody>
</table>

**Errors**

*FDFErrBadParameter, FDFErrBadFDF, FDFErrCantInsertField, FDFErrEmbeddedFDFs, FDFErrInternalError*

**Related Functions**

*PDFAddTemplate*

**C Example**

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

**ActiveX Example**

```c
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

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

ActiveX Syntax

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

Perl Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| nElemNum  | The element number to set in the FDF’s /ID key. Must be either:  
|           | ● 0: the permanent ID  
|           | ● 1: the changing ID  
| elem      | Buffer containing the ID element to set.  
| bufSize   | The number of bytes in elem.  

(Contonly)

Errors

FDFErcBadParameter, FDFErcIncompatibleFDF, FDFErcInternalError

Related Functions

FDFGetID

C Example

```c
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 the FDF is template-based (FDFAddTemplate has been called), this function acts on the most recently added template.

C Syntax

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

ActiveX Syntax

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

Perl Syntax

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

Table 40:

<table>
<thead>
<tr>
<th>fieldName</th>
<th>String representing the fully qualified name of the field. (for example, employee.name.last).</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaleWhen</td>
<td>A value indicating when to scale the icon to fit within the rectangle of the annotation represented by the field. See FDFSacleWhen for possible values.</td>
</tr>
<tr>
<td>bProportional</td>
<td>true if scaling should be proportional; false otherwise.</td>
</tr>
<tr>
<td>x</td>
<td>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.</td>
</tr>
<tr>
<td>y</td>
<td>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.</td>
</tr>
</tbody>
</table>

Errors

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError

Related Functions

FDFSetAP
FDFSetAPRef

C Example

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

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

ActiveX Example

```vbnet
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

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

ActiveX Syntax

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

Perl Syntax

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

Parameters

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

Errors

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError

C Example

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

Const FDFUp = 3
objPdf.FDFSetImportDataAction "my button", FDFUp, "myprof.fdf"
**FDFToolkit Reference**

**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 (*PDFAddTemplate* has been called), *FDFSetJavaScriptAction* acts on the most recently added template.

**C Syntax**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

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

**Errors**

*FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError*

**C Example**

```c
theErc = FDFSetJavaScriptAction (theFDF, "checkbox", FDFUp,  
"var valMe = event.target.value;\r"  
"var fRadio = this.getField("radio");\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
objPdf.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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
<tr>
<td>whichTrigger</td>
<td>An <code>FDFActionTrigger</code> value indicating the event trigger for the action. May be either: <code>PDFEnter</code>, <code>PDFExit</code>, <code>FDPDown</code>, <code>FDFUp</code>, <code>PDFOnFocus</code>, <code>PDFOnBlur</code> If <code>FDFUp</code>, <code>/A</code> entry is used, otherwise an <code>/AA</code> entry is created.</td>
</tr>
<tr>
<td>theName</td>
<td>String in PDFDocEncoding converted to a PDF name to use as the value of <code>/N</code> in the named action.</td>
</tr>
</tbody>
</table>

**Errors**

`FDFErcBadParameter`, `FDFErcBadFDF`, `FDFErcCantInsertField`, `FDFErcEmbeddedFDFs`, `FDFErcInternalError`

**Related Functions**

`FDFAddTemplate`

**C Example**

```c
resultCode = FDFSetNamedAction (theFDF, "Named1", FDFUp, "FirstPage");
```
**ActiveX Example**

Const FDFUp = 3
thePdf.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**

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

**Perl Syntax**

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

**ActiveX Syntax**

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

**Parameters**

**Table 44:**

<table>
<thead>
<tr>
<th>cScript</th>
<th>Must either be in PDFDocEncoding or in Unicode. Use &quot;\r&quot; as line separator within the script. Use &quot;\t&quot; for tabs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBefore</td>
<td>Pass true to have cScript execute before the data in the FDF is imported, false to indicate execution after import.</td>
</tr>
</tbody>
</table>

**Return Values**

FDFErcBadParameter, FDFErcIncompatibleFDF, FDFErcEmbeddedFDFs, FDFErcInternalError

**C Example**

```c
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 the FDF 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

```perl
$outFDF->SetOpt($fieldName, $nElemNum, $string1, $string22);
```

**Parameters**

**Table 45:**

<table>
<thead>
<tr>
<th><strong>fieldName</strong></th>
<th>String representing the fully-qualified name of the field (for example, <em>employee.name.last</em>).</th>
</tr>
</thead>
<tbody>
<tr>
<td>nElemNum</td>
<td>The index of the element to set in the <code>/Opt</code> array. The first element in the <code>/Opt</code> array has an index of 0 (zero).</td>
</tr>
</tbody>
</table>
Table 45:

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

Errors

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcInternalError

Related Functions

FDFGetOpt
FDFGetOptNumElem

C Example

```c
/*** 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

```c
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 the 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**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, employee.name.last).</td>
</tr>
<tr>
<td>whichTrigger</td>
<td>An FDFActionTrigger value indicating the event trigger for the action: FDFEnter, PDFExit, FDFDown, FDFUp, FDFOnFocus, or FDFOnBlur. If FDFUp, an /A entry is used, otherwise an /AA entry is created.</td>
</tr>
</tbody>
</table>
Table 46:

| theFlags | Value for the /Flags key in the ResetForm action. At present, only one flag is defined: ● Bit 1 - Include/exclude flag: determines how the fields key is interpreted. If this bit is 0, then fields represents the individual fields to reset. If the bit is 1, then all fields in the AcroForm are reset except for those in the fields array. |
| nFlds | An integer specifying how many field names are being passed in the fields parameter. If zero, this function is identical to FDFSetResetFormAction. |
| fields | An array of nFlds 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 /Fields key in the ResetForm 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**

```c
PDFErr FDFSetResetFormAction(
    FDFDoc theFDF,
    const char* fieldName,
    PDFActionTrigger whichTrigger);
```

**ActiveX Syntax**

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

**Perl Syntax**

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

**Table 47:**

<table>
<thead>
<tr>
<th>fieldName</th>
<th>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</th>
</tr>
</thead>
</table>
| whichTrigger | An **FDFActionTrigger** value indicating the event trigger for the action:  
  ● FDFEnter  
  ● FDFExit  
  ● FDFDown  
  ● FDFUp  
  ● FDOnFocus  
  ● FDOnBlur  
  If FDFUp, an /A entry is used, otherwise an /AA entry is created. |
| rgFields | Optional parameter. If not passed, then the created action will not include a `/Fields` 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

```c
/* 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 the *FDF* 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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

**Table 48:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully-qualified name of the field (for example, <em>employee.name.last</em>).</td>
</tr>
<tr>
<td><strong>newValue</strong></td>
<td>If the FDF includes an <em>/Encoding</em> key (see <em>FDFSetEncoding</em>), 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).</td>
</tr>
<tr>
<td><strong>bNotUsed</strong></td>
<td>This parameter is not used by this method. You can pass <em>true</em> or <em>false</em>.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newStatus</td>
<td>Host-encoded string to use as new status.</td>
</tr>
</tbody>
</table>

**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**

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

**Perl Syntax**

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

### Table 50:

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

**Note:** In ActiveX, this parameter is provided in the FDFSetSubmitFormAction function.

Errors

DFERcBadParameter, DFERcBadFDF, DFERcEmbeddedFDFs, DFERcCantInsertField, DFERcInternalError

Related Functions

FDFSetSubmitFormAction
FDFSetResetByNameAction

C Example

```c
/* 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 (**PDFAddTemplate** has been called), **FDFSetSubmitFormAction** acts on the most recently added template.

**C Syntax**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, employee.name.last).</td>
</tr>
<tr>
<td>whichTrigger</td>
<td>An <strong>FDFActionTrigger</strong> value indicating the event trigger for the action:</td>
</tr>
<tr>
<td></td>
<td>- FDFEnter, FDFExit, FDFDown, FDFUp, PDFOnFocus, or PDFOnBlur</td>
</tr>
<tr>
<td></td>
<td>If FDFUp, an /A entry used, otherwise /AA entry is created.</td>
</tr>
<tr>
<td>theURL</td>
<td>String to be used as value for the /F key in the <strong>SubmitForm</strong> action.</td>
</tr>
<tr>
<td>theFlags</td>
<td>Value for the /Flags key in the <strong>SubmitForm</strong> action. By default it is &quot;0&quot;. Other values that can be used for the /Flags key are described in Section 5.7.1 in the <strong>PDF Reference (fourth edition, version 1.5)</strong>.</td>
</tr>
</tbody>
</table>
FDF Toolkit Reference
FDFSetSubmitFormAction

Table 51:

<table>
<thead>
<tr>
<th>fields (ActiveX only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional parameter</td>
</tr>
<tr>
<td>If not passed, then the created action does not include a /Fields 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.</td>
</tr>
</tbody>
</table>

**Note:** This parameter allows ActiveX to have the same functionality as `FDFSetSubmitByNameAction`.

**Return Value**

*Perl, C:* Error code

**Errors**

- `FDFErcBadParameter`, `FDFErcBadFDF`, `FDFErcCantInsertField`, `FDFErcEmbeddedFDFs`, `FDFErcInternalError`

**Related Functions**

- `FDFSetResetByNameAction`
- `FDFSetSubmitByNameAction`

**C Example**

```c
/* 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**

```vbs
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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>cTargetFrame</th>
<th>Value of the <code>/Target</code> key. Should be Host-encoded.</th>
</tr>
</thead>
</table>

**Errors**

`FDFErcBadParameter, FDFErcIncompatibleFDF, FDFErcInternalError`

**Related Functions**

`FDFSetFile`

`FDFSetFileEx`

**C Example**

```c
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

FDFErr 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:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
</tbody>
</table>
| **whichTrigger** | An **FDFActionTrigger** value indicating the event trigger for the action:  
  - FDFEnter  
  - FDFExit  
  - FDFDown  
  - FDFUp  
  - FDFOnFocus  
  - FDFOnBlur  
  If FDFUp, an /A entry is used, otherwise an /AA entry is created. |
| **theURI** | Host-encoded string to be used as value for the /URI key in the URI action. |
| **isMap** | Value for the /IsMap key in the URI action. If false, the /IsMap key is not added. |

**Errors**

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcEmbeddedFDFs, FDFErcInternalError

**C Example**

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

**ActiveX Example**

```vbnet
Const FDFUp = 3
thePdf.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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldName</strong></td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
<tr>
<td><strong>newValue</strong></td>
<td>String to use as the new value. If the FDF includes an /Encoding key (see <em>FDFSetAS</em>), 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, <em>newValue</em> must be either &quot;Off&quot;, or a value that was entered as the &quot;Export Value&quot; when defining the properties of the field.</td>
</tr>
<tr>
<td><strong>bNotUsed</strong></td>
<td>Not used.</td>
</tr>
</tbody>
</table>

**Errors**

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

**Related Functions**

- *FDFGetValue*

**C Example**

```c
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**

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

**ActiveX Syntax**

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

**Perl Syntax**

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

**Parameters**

**Table 55:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldName</td>
<td>String representing the fully qualified name of the field (for example, <code>employee.name.last</code>).</td>
</tr>
<tr>
<td>nValues</td>
<td>(C only) An integer specifying how many values are being passed in the parameter <code>newValues</code>. If you set this to 1, FDFSetValues behaves exactly as FDFSetValue.</td>
</tr>
<tr>
<td>newValues</td>
<td>An array of strings. See FDFSetValue for a discussion of encodings.</td>
</tr>
</tbody>
</table>

**Errors**

FDFErcBadParameter, FDFErcBadFDF, FDFErcCantInsertField, FDFErcInternalError

**Related Functions**

FDFSetValue
FDFGetValue

**C Example**

```c
char* myValues[2] = {"first", "second"};
```
FDFErc erc = FDFSetValues(thePDF, "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**

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

**Parameters**

<table>
<thead>
<tr>
<th>$NameOfFile</th>
<th>The name for the new FDF object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NumOfFiles</td>
<td>The number of bytes to read if &quot;-&quot; (stdin) is passed in as $NameOfFile.</td>
</tr>
</tbody>
</table>

**Return Value**

A new FDF object.

**Related Functions**

*newFromEmbedded*
newFromBuf

*(Perl only)*

Creates a new FDF object from a buffer.

**Perl Syntax**

```perl
$FDFobj = Acrobat::FDF::newFromBuf ($buffer);
```

**Parameters**

<table>
<thead>
<tr>
<th>buffer</th>
<th>The buffer containing the FDF to be opened.</th>
</tr>
</thead>
</table>

**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**

```perl
$FDFObj = $inFDF->OpenFromEmbedded($iWhich, $password);
```

**Parameters**

**Table 58:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iWhich</code></td>
<td>Zero-based index of the embedded FDF that should be opened.</td>
</tr>
<tr>
<td><code>Password</code></td>
<td>If the embedded FDF is encrypted, you need to pass in the correct password. If it is not encrypted, the system ignores this parameter.</td>
</tr>
</tbody>
</table>

**Return Value**

The embedded FDF object.

**Perl Example**

```perl
$FDFObj1 = new Acrobat::FDF($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**

```perl
FDFErc EmbedAndClose ($FDFObj, $cPassword);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDFObj</td>
<td>The object that holds one or more FDFs</td>
</tr>
<tr>
<td>cPassword</td>
<td>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 NULL. If you pass in an empty string, the FDF embeds in an unencrypted state.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufFldName</td>
<td>Buffer containing a field name. This buffer was supplied in the call to FDFEnumValues, and is filled by that function with a new field name for each invocation of the callback.</td>
</tr>
<tr>
<td>bufFldVal</td>
<td>Buffer containing a field value. This buffer was supplied in the call to FDFEnumValues, and is filled by that function with a new value for each invocation of the callback. If the FDF includes an /Encoding key (see FDFGetEncoding), 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.</td>
</tr>
<tr>
<td>clientData</td>
<td>User-supplied data that was passed in the call to FDFEnumValues.</td>
</tr>
</tbody>
</table>

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**

```c
typedef void (*ThreadsafeCallback)( void* threadsafeObj );
```

**Parameters**

| threadsafeObj | The data that was passed to `FDFRegisterThreadsafeCallbacks` when this callback was registered. |

**Related Functions**

`FDFRegisterThreadsafeCallbacks`