FDF Toolkit Overview

Technical Note #5194

ADOBE SYSTEMS INCORPORATED

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Eastern Regional Office
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120

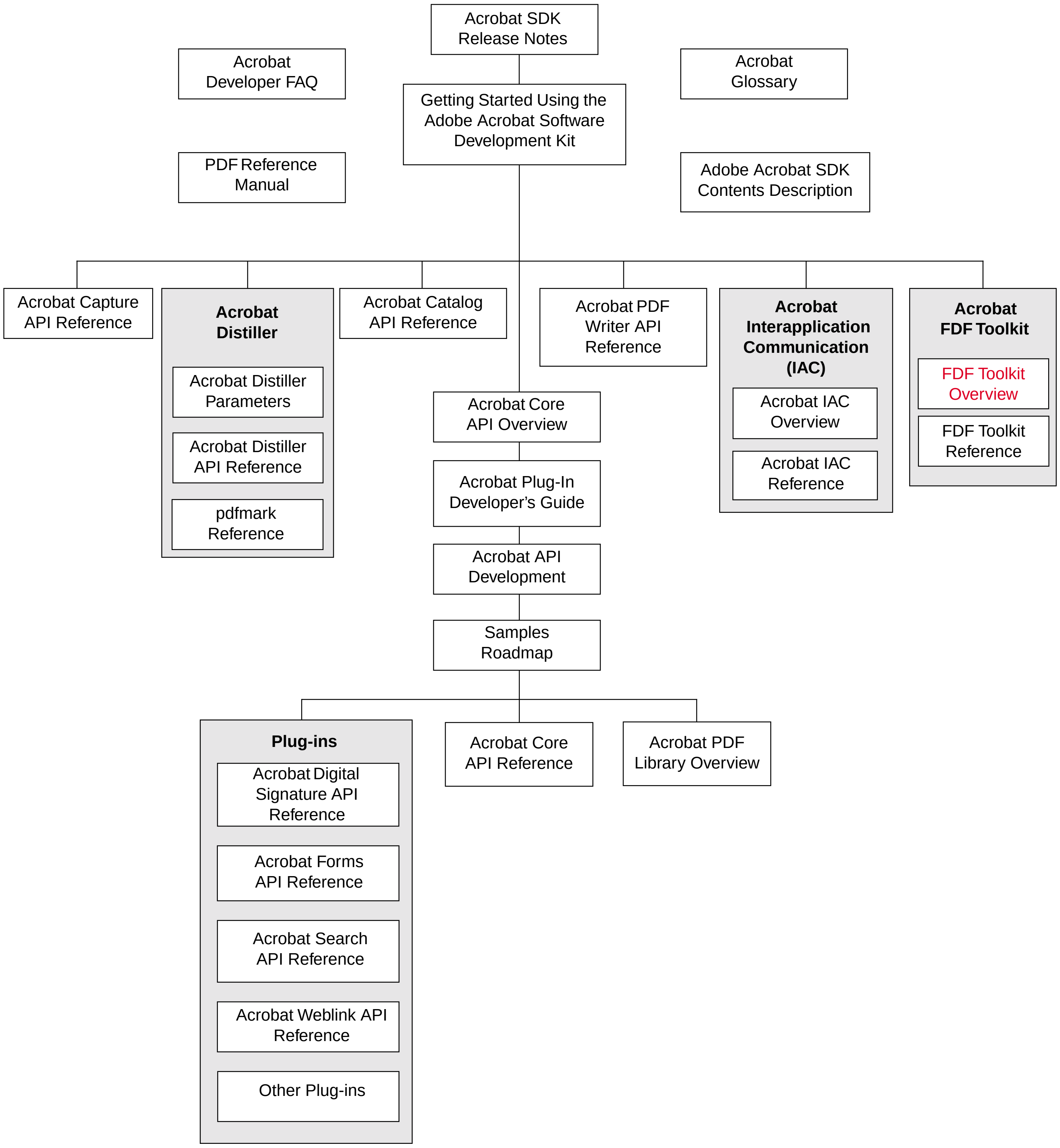
ADOBE SYSTEMS EUROPE LIMITED
Adobe House, 5 Mid New Cultins
Edinburgh EH11 4DU
Scotland, United Kingdom
+44-131-453-221

ADOBE SYSTEMS JAPAN
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150 Japan
+81-3-5423-8100

Adobe Developer Technologies

Revised: November 10, 1999

Documentation Roadmap


Acrobat SDK Release Notes
Acrobat Developer FAQ
Acrobat Glossary
Getting Started Using the Adobe Acrobat Software Development Kit
PDF Reference Manual
Adobe Acrobat SDK Contents Description
Acrobat Capture API Reference
Acrobat Distiller
Acrobat Distiller Parameters
Acrobat Distiller API Reference
pdfmark Reference
Acrobat Catalog API Reference
Acrobat PDF Writer API Reference
Acrobat Interapplication Communication (IAC)
Acrobat IAC Overview
Acrobat IAC Reference
Acrobat FDF Toolkit
FDF Toolkit Overview
FDF Toolkit Reference
Acrobat Core API Overview
Acrobat Plug-In Developer's Guide
Acrobat API Development
Samples Roadmap
Plug-ins
Acrobat Digital Signature API Reference
Acrobat Forms API Reference
Acrobat Search API Reference
Acrobat Weblink API Reference
Other Plug-ins
Acrobat Core API Reference
Acrobat PDF Library Overview

Contents

What Is in This Tech Note . . . 9
Audience . . . 9
Conventions Used in This Book . . . 10
Other Useful Documentation . . . 10
Acrobat Forms Resources . . . 10
Prerequisites . . . 11

PART I: FDF Overview

Chapter 1 Introduction to FDF . . . 13

FDF Overview . . . 13
- Acrobat Forms . . . 13
- FDF Toolkit Functionality . . . 13
- The Basics . . . 13

Description of the Forms Data Format . . . 14
- File Structure . . . 14
- The FDF Catalog Object . . . 15
- Use of FDF . . . 19
- Sample FDF . . . 19
- FDF for Annotations . . . 20

Posting Form Data to a Web Server in HTML or FDF . . . 21
- Choosing the Output Format . . . 21
- Replacing an HTML Form with Acrobat Form . . . 21

Scenarios for Using FDF . . . 22
- Acrobat Form with Results Returned in the Same Form . . . 22
- Acrobat Form with Results Returned in a New Form . . . 23
- HTML Form with Results Returned in an Acrobat Form . . . 24

Chapter 2 FDF and Web Server Connectivity . . . 29

Building Server Side Applications With FDF . . . 29
- The FDF Toolkit . . . 29
- Sample Application . . . 31
- Setting Up the Client Side PDF File . . . 33
- Populating a PDF File . . . 33
- Sample Application to Generate FDF Data . . . 37
- Debugging Tip . . . 38

PART II: Language Specific Overview

Chapter 3 FDF Toolkit for C/C++ 41

Introduction 41

Building Applications with the FDF Toolkit 41

UNIX Support 41

Microsoft Windows Support 41

FDF Toolkit Methods 42

General 43

UNIX and Macintosh Specific 43

Parsing FDF Data 43

Generating FDF Data 43

Cleaning Up and Closing FDF Files 44

Chapter 4 FDF Toolkit for ActiveX 45

Introduction 45

Building Applications with the FDF Toolkit 45

Installation 45

Using the FDF Toolkit in Visual Basic 46

Using the FDF Toolkit with Active Server Pages 46

Setting Up the NT Server to Handle FDF Data 46

Chapter 5 FDF Toolkit for Java 50

Introduction 50

Differences Between Java Implementations 50

Building Applications with the FDF Toolkit 50

FDF JNI Based Java Toolkit Class Overview 51

For the FDFTK Class 51

General Methods For the Creating and Opening FDFDoc Objects 51

FDFDoc Methods for Parsing FDF Data 51

Methods for Generating FDF Data 52

Cleaning Up and Closing FDF Files 53

FDF ActiveX Based Java Toolkit Class Overview 53

Chapter 6 FDF Toolkit for Perl 54

Perl FDF Toolkit Overview for Windows 54

Calls Not Available in the Windows Perl FDF Toolkit 54

Using the Perl FDF Toolkit 54

Sample Usage of the Windows Perl FDF Toolkit 55

FDF Toolkit Methods 55
General Methods 55
Templates 56
Generating FDF Data 56
Parsing FDF Data 56
Cleaning Up and Closing FDF Files 57
Calls Not Available in the UNIX Perl FDF Toolkit 58
Functions Only Available in the UNIX Perl FDF Toolkit 58
Supported Platforms for the FDF Toolkit Libraries 58
Using the Perl FDF Toolkit 58
Sample Usage of the Perl FDF Toolkit for UNIX 59

FDF Toolkit methods 59
General Methods 59
Templates 60
Generating FDF Data 60
Parsing FDF Data 60
Cleaning Up and Closing FDF Files 61

List of Figures

Figure 1.1 Using HTML with Acrobat Forms 22
Figure 1.2 Returning FDF into the Same Form 23
Figure 1.3 Simple ASP for generating FDF 23
Figure 1.4 Returning FDF into a Different Form 24
Figure 1.5 Start from HTML, end in an Acrobat form 25
Figure 1.6 Dynamic Creation of a PDF from Templates 26

List of Tables

Table 1.1 FDF Attributes 15
Table 1.2 Field Attributes 15
Table 1.3 Icon-fit Attributes 17
Table 1.4 Pages-object Attributes 17
Table 1.5 Templates Dictonary Attributes 18
Table 1.6 Named Page Reference Dictionary Attributes 18
Table 1.7 FDF Annotation Attributes 19

Preface

What Is in This Tech Note

This technical note provides an introduction to development using the Forms Data Format (FDF) Toolkit. It describes FDF, the FDF Toolkits, the FDF Toolkit Software Development Kit (SDK), and support options. This technical note is divided into the following chapters:

- Chapter 1 provides an introduction to the FDF toolkit, some typical scenarios for using the toolkit and introduces templates.
- Chapter 2 discusses FDF and Web Server Connectivity. It shows how to build server-side applications with FDF including parsing FDF and generating FDF.
- Chapter 3 describes the FDF Toolkit for C/C++.
- Chapter 4 describes the FDF Toolkit for ActiveX.
- Chapter 5 describes the FDF Toolkit for Java.
- Chapter 6 describes the FDF Toolkit for Perl.

This overview provides background information on the Forms Data Format, examples of use, and platform-specific information.

Audience

This technical note is intended for developers who need basic information on the FDF Toolkit and its capabilities. It assumes that you have basic knowledge of:

- Windows NT Server or UNIX
- Web authoring on either of these platforms
- PDF file format
- Adobe Acrobat and the capabilties of the Forms plug-in
- One of the following languages: Visual Basic, C/C++, Java, or Perl, and have at least one development environment for that language

Conventions Used in This Book

The Acrobat documentation uses certain text styles to identify various operators, keywords, terms, and objects.

Name	Description	Examples
ObjectName	Acrobat object types are written with initial capital letters and italics	*ASAtom*
methodName	Acrobat API methods and parameter names	*AVDestInfoDestroy* and *des-tInfo*
file names	File names	*WeblinkHFT.h*
operator	PostScript language operators, PDF operators, PDF keywords, names of keys in dictionaries, and predefined names.	**moveto**, **Tf**, **stream**, and **Type**
`code`	Examples and samples	`PDColorValue color;`
`comments`	Comments in code examples	/* disable floating windows */
documents	Other document titles	*Acrobat Core API Overview*
Web links	Hypertext links to URLs	www.apple.com
term	The first occurrence of terms and the Boolean values *true* and *false*	*true* and *false*

Other Useful Documentation

You should be familiar with the Acrobat core API and Portable Document Format (PDF). The following technical notes provide this information.

Forms Data Format Toolkit Reference, Technical Note #5193, describes how to develop server applications to parse or generate Forms Data Format (FDF) data for use with Acrobat Forms.

Portable Document Format Reference Manual, Version 1.3. Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files. It is intended for application developers who wish to produce PDF files directly.

Acrobat Forms Resources

You can find important resources that will help you get the most out of the forms capabilities in Adobe Acrobat. New resources will be made available to our Web site on a regular basis, so you may want to bookmark this page for convenience:

http://www.adobe.com/prodindex/acrobat/formsresources.html

The FDF Toolkit is part of the Adobe Solutions Network's Acrobat SDK. To download all or part of the FDF Toolkit, which includes support for writing Web applications in C/C++, Java, Perl and ActiveX, go to:

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

Prerequisites

Readers are assumed to know the following:

- The process of creating form fields and buttons, and modifying their properties. See the *Adobe Acrobat Guide*, accessible via the menu item Help -> Acrobat Guide.
- 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.

In addition, you will need to install the FDF Toolkit on a computer with the following requirements:

- Windows NT or UNIX
- Acrobat 4.0
- Administrator privileges
- Web Server, such as IIS or Apache
- A development environment for at least one of the following languages: C/C++, Visual Basic, Perl, or Java
- An e-mail account (optional, but strongly suggested)

Part I

FDF Overview

1 Introduction to FDF

FDF Overview

This chapter describes the Forms Data Format (FDF) and the FDF Toolkit for creating and/or parsing FDF-formatted text for an Acrobat form. It also compares HTML and FDF as the file format for Acrobat form-based submissions.

Acrobat Forms

Acrobat Forms are a group of extensions to PDF used by Adobe Acrobat. These extensions allow PDF files to contain fields and buttons and can be considered to be simply a new layer on top of a PDF file. The underlying PDF file may be created by any PDF producer such as the PDF Writer, the Acrobat Distiller application, or the Acrobat Capture application. The fields are subsequently added manually using Acrobat. In addition to the forms data stored in PDF files, there is a need for a file format to use for transmitting forms data between a server and a client. The Acrobat products support both the HTML form format (that is, MIME type application/x-www-form-urlencoded) and an Acrobat-specific Forms Data Format: FDF (MIME type application/vnd.fdf).

FDF Toolkit Functionality

The FDF Toolkit is a threadsafe API for writing a server application to generate FDF data or parse FDF data from a form created by the Acrobat Forms plug-in. The FDF Toolkit can be used on Microsoft Windows NT, or UNIX platforms. The FDF Toolkit supports the C/C++, Java, Perl, and Visual Basic (Active Server Pages) development languages.

The Basics

After filling in an Acrobat form, the user must click on a button whose action is a Submit form action to submit the data to a server. The format of the submitted data may be either HTML-compatible (urlencoded) or FDF. The selection of which format to use is made at the time the form is created.

- If HTML is selected, the format submitted is identical to HTML form submissions. Existing CGI scripts for HTML forms may be used to parse data in this format.
- If FDF is selected, the data format is FDF. There is a server library to help parse and generate FDF files.

The URL to submit to 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 the 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 this to the server, simply select the "Export Form Data…" menu item from the File menu of Acrobat.*

Description of the Forms Data Format

FDF is used when submitting form data to a server, receiving the response, and incorporating it into the form. It can also be used to generate (that is, “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.

FDF can also be used to control the document structure. That is, constructs within FDF allow it to control which Acrobat forms are used in the creation of a new PDF document. This functionality can be used to create complex documents dynamically.

FDF is also used to define a container for annotations that are separate from the PDF document to which the annotations apply.

File Structure

An FDF file consists of a one-line header, a body, and a trailer. It can optionally contain a cross-reference table. FDF is structured in the same way as PDF, but need only contain those elements required for Acrobat forms data export and import, which are:

- Header
- Body
- Cross Reference Table (optional)
- Trailer

Header

The first line of an FDF file specifies the version number of the PDF specification that FDF is a part of. The current version of PDF is 1.3; therefore the first line of an FDF file is:

```
%FDF-1.3
```

Body

The body consists of one Catalog object and any additional indirect objects that it may reference. The Catalog object is a dictionary with only one (required) key in it, **FDF**. Its value is a dictionary, whose entries are described in *The FDF Catalog Object*. It is legal for the body to contain additional objects, and for the Catalog object to contain additional key-value pairs. Comments can appear anywhere in the body section of an FDF file. Just as in PDF, objects in FDF can be direct or indirect.

Trailer

The trailer consists of a trailer dictionary, followed by the last line of the FDF file, containing the end-of-file marker,**%%EOF**. The trailer dictionary consists of the keyword **trailer**, followed by at least one key-value pair enclosed in double angle brackets. The only required key is **Root**, and its value is an indirect reference to the Catalog object in the FDF body.

It is legal for the trailer dictionary to contain the additional key-value pairs described in the PDF specification.

The FDF Catalog Object

The value of the **FDF** key in the Catalog object is a dictionary whose entries are described in Table 1.1.

TABLE 1.1 FDF Attributes

Key	**Type**	**Semantics**
Fields	array	*(Optional)* This array contains the root fields being exported or imported. A root field is one with no parent (it is not in the *Kids* array of another field).
Status	string	*(Optional)* A status to be displayed indicating the result of an action, typically a *SubmitForm* action. This string is encoded with *PDFDocEncoding*. **NOTE:** *The Acrobat 3.0 implementation of Forms displays the Status, if any, in an Alert Note, when importing an FDF.*
F	File specification	*(Optional)* File specification for the PDF document that this FDF data was exported from, or is meant to be imported into.
ID	array	*(Optional)* The value of the **ID** field in the trailer dictionary of the PDF document that this FDF data was exported from, or is meant to be imported into.
Pages	array	*(Optional)* This array causes new pages to be added to a PDF document. Use of the **/Pages** key precludes the use of the **/Fields** and /**Status** keys.
Encoding	name	*(Optional)* The encoding to be used for any FDF field value **(/V** key) or option **(/Opt** key) that is a string and does not begin with the Unicode prefix <FE FF>. The default is *PDFDocEncoding*. **NOTE:** *The only Encoding value supported by Acrobat 4.0 is Shift-JIS. All other values will use the default, PDFDocEncoding.*
Annot	array	*(Optional)* An array of FDF annotation dictionaries.

FDF fields

Table 1.2 describes the attributes of each field in the FDF.

TABLE 1.2 Field Attributes

Key	**Type**	**Semantics**
T	string	*(Required)* The partial field name.
Kids	array	*(Optional)* Contains the child field dictionaries.
V	various	*(Optional)* Field value.
Opt	array	*(Optional)* Options.

TABLE 1.2 Field Attributes

Key	Type	Semantics
Ff	integer	*(Optional)* Field flags. When imported into an Acrobat form, it replaces the current value of the /**Ff** key in the corresponding field inside the form. If /**SetFf** and/or /**ClrFf** are also present, they are ignored.
SetFf	integer	*(Optional)* Field flags. When imported into an Acrobat form, it is OR'ed with the current value of the /**Ff** key in the corresponding field inside the form.
ClrFf	integer	*(Optional)* Field flags. When imported into an Acrobat form, for each bit that is set to one in this value, sets the corresponding bit in the form field's /**Ff** flags to zero. If /**SetFf** is also present, /**ClrFf** is applied after /**SetFf**.
F	integer	*(Optional)* Widget annotation flags. When imported into an Acrobat form, it replaces the current value of the /**F** key in the corresponding field inside the form. If /**SetF** and/or /**ClrF** are also present, they are ignored.
SetF	integer	*(Optional)* Widget annotation flags. When imported into an Acrobat form, it is OR'ed with the current value of the /**F** key in the corresponding field inside the form.
ClrF	integer	*(Optional)* Widget annotation flags. When imported into an Acrobat form, for each bit that is set to one in this value, sets the corresponding bit in the form's /**F** flags to zero. If /**SetF** is also present, /**ClrF** is applied after /**SetF**.
AP	dictionary	*(Optional)* Dictionary containing the appearances for a Push Button field.
AS	name	*(Optional)* Appearance state.
A	dictionary	*(Optional)* Action to be performed on activation of this Widget annotation.
AA	dictionary	*(Optional)* Additional actions.
APRef	dictionary	*(Optional)* Dictionary that includes references to external PDF files containing the pages to use for the appearances for a Push Button field. The values of the N, R, and D keys must all be named page reference dictionaries, described in Table 1.6. If both an /**AP** and an /**APRef** are provided, the /**AP** is used.
IF	dictionary	(*Optional*) The Icon-fit dictionary controls how the button icon is to be manipulated within the boundaries of the button.

Icon-fit Dictionary

The Icon-fit dictionary controls how the button icon is to be manipulated within the boundaries of the button. The Icon-fit dictionary, if provided, must contain three keys, described in Table 1.3.

TABLE 1.3 Icon-fit Attributes

Key	Type	Semantics
SW (Scale When)	name	*(Required)* Indicates when the icon should be scaled inside the button. The value is one of: **B** Scale when icon is too big, or only scale down. **S** Scale when icon is too small, or only scale up. **N** Never scale. **A** Always scale. This is the default value, used if no icon-fit dictionary is provided.
S (Scaling)	name	*(Required)* Indicates how the icon should be scaled inside the button. Possible values are: **A** (Anamorphic) Always exactly fills the BBox for the button. Anamorphic scaling does not maintain the aspect ratio of the icon. **P** (Proportional) Aspect ratio is maintained. The icon is scaled until the contents fill the BBox for the field annotation. This is the default value, used if no icon-fit dictionary is provided.
A (Align)	array	*(Required)* An array of two numbers between 0 and 1 indicating the fraction of leftover space to assign to the left and bottom location of the icon. A value of [0.5 0.5] centers the icon in the BBox, and a value of [0 0] makes the icon appear in the lower left corner of the BBox. This array is not used if the icon is scaled anamorphically. If no Icon-fit dictionary is provided, a value of [0.5 0.5] is used.

FDF Pages Object

Table 1.4 presents the pages-object attributes.

TABLE 1.4 Pages-object Attributes

Key	Type	Semantics
Templates	array	*(Required)* An array of named page dictionaries that describe the named pages that serve as templates on a page. The templates dictionary attributes are described in Table 1.5.
PInfo	dictionary	(*Optional*) Contains other information about the page described by this dictionary. Currently there are no attributes defined.

Table 1.5 describes the templates dictionary attributes.

TABLE 1.5 *Templates Dictonary Attributes*

Key	Type	Semantics
TRef	dictionary	*(Required)* A named page reference dictionary that describes the location of the template. The named page reference dictionary attributes are described in Table 1.6.
Fields	array	*(Optional)* This array contains the root fields being imported. A root field is one with no parent (that is, it is not in the *Kids* array of another field). The attributes of the fields are described in Table 1.2.
Rename	boolean	*(Optional)* If *false*, prevents the fields from being renamed during FDF importing. The default value is *true*. /**Rename** affects the fields described in the same dictionary and their children. The effect of /**Rename** is described more fully below.

The /**Rename** key is used when a new page is being added to a PDF document under the control of the FDF, and a new field on the new page has the same name as an existing field. This can occur if the same template page is imported more than once, or if two different templates are imported but have fields with the same names.

If /**Rename** is *true*, fields from the document being imported are renamed to guarantee their uniqueness. If /**Rename** is *false*, the fields are not renamed, and each time the FDF provides keys for that field, all fields with that name will be updated.

NOTE: *Acrobat renames fields by prepending a page number, a template name, and an ordinal number to the field name. The template ordinal number corresponds to the order in which the template is applied to a page, with 0 being the first template specified for the page. For example, suppose the first template used on the fifth page has the name template and has /Rename set to true. In this case, the fields defined in that template will be renamed by prepending the character string **P5.Template_0**. to the field names.*

The named page reference dictionary describes the location of external templates or page elements as shown in Table 1.6.

TABLE 1.6 *Named Page Reference Dictionary Attributes*

Key	Type	Semantics
F	File specification	*(Optional)* Refers to an external PDF file. If the /**F** key is omitted, it is assumed that the /**Name** refers to a page in the document being imported into.
Name	string	*(Required)* The name of the referenced named page (template or page element) in the document specified by the /**F** key.

FDF Annotation dictionaries

Each annotation dictionary in an FDF file must have a /**Page** key indicating the page of the source document to which the annotation is attached.

TABLE 1.7 FDF Annotation Attributes

Key	Type	Semantics
Page	integer	*(Required for annotations in FDF files)* The ordinal page number on which this annotations should appear. Page 0 is the first page.

Use of FDF

For most of the keys, unless otherwise indicated in Table 1.2, importing consists of taking the value of each key as received in the FDF data, and using it to replace the value of the corresponding key in the field inside the form with the same fully qualified name.

NOTE: *Of all the possible keys shown in Table 1.2, Acrobat 3.0 will export only the /**V** key of a field when generating FDF data, and Acrobat 4.0 will export only the /**V** and /**AP** keys. It will, however, import FDF files containing fields using any of the described keys.*

NOTE: *If Acrobat is importing FDF data that has specified in the /**F** key of the /**FDF** dictionary a form that is not the current PDF, then that form is fetched first, before the FDF data gets imported.*

NOTE: *When exporting FDF data, Acrobat computes a relative path from the location the FDF data is being stored to the location the form is in, and uses that as the value of the /**F** key in the FDF dictionary.*

NOTE: *If FDF data being imported contains fields whose fully-qualified names are not present in the form, those fields will be discarded. This feature can be useful, among other cases, if FDF data containing commonly used fields (such as name, address, and so forth) is used to populate various types of Acrobat Forms, each of which does not necessarily include all the fields available in the FDF data.*

NOTE: *As shown in Table 1.2, the only required key in the field dictionary is /**T**. One possible use for exporting FDF with fields that only contain a /**T** key but no /**V** key, is as an indication to a server of which fields are desired in the FDF coming back as a response. For example, a server accessing a database might choose between sending all fields in a record, versus just some selected ones, based on the use of this feature in the request FDF. The implementation of Acrobat Forms will ignore, during import, fields in the FDF data that do not exist in the form.*

NOTE: *The Acrobat implementation of forms allows the choice for a SubmitForm action to send the data using HTML format. This is for the benefit of existing server scripts written to process such forms. However, any such existing scripts that generate new HTML forms as a response will need to be modified to generate FDF data instead.*

Sample FDF

The following example illustrates a simple FDF file.

```
%FDF-1.2
1 0 obj <<
/FDF <<
/Fields
[
<<
/T (My Children)
/V (Tali)
/Opt [(Maya) (Adam) (Tali)]
>>
]
/F (Dependents.pdf)
>>
>>
endobj
trailer
<</Root 1 0 R>>
%%EOF
```

FDF for Annotations

The following example shows an FDF file used to express a set of annotations for a separate PDF document.

```
%FDF-1.3
%\342\343\317\323
1 0 obj
/FDF <<
     /Annot
     [
       <<
        /Type /Annot
        /Subtype /Text
        /Page 0
        /Rect [88 370 359 442]
        /Contents (Converted to Word for Windows)
        /M (D:19970620115631)
        /T (Joe Carousel)
       >>
       <<
        /Type /Annot
        /Subtype /Circle
        /Page 0
        /Rect [232 282 341 444]
        /Contents (Was Macintosh version 5.1a)
        /M (D:19970616124809)
        /T (Joe Carousel)
       >>
     ]
  >>
endobj
%%EOF
```

Posting Form Data to a Web Server in HTML or FDF

This section describes how to post form data to a Web Server using HTML or FDF.

Choosing the Output Format

You can choose between HTML and FDF formats when implementing an Acrobat forms application.

- HTML support allows Acrobat forms to be used as a direct replacement for HTML forms. Choose HTML if you need compatibility with existing systems.
- FDF supports various capabilities not possible with HTML. Choose FDF if you wish to take advantage of its additional capabilities, which include:
  - When sending data back from the server, it is not necessary to re-send the form itself; the data can populate the same form that originated the data.
  - FDF allows you to change the look of buttons. Additionally, you can take advantage of this capability to send graphical information in either direction between the client and the server.
  - The form can be altered via an FDF sent back from the server: the various actions attached to buttons can be modified, field properties can be changed, and list boxes and combo boxes can be populated with different choices.
  - FDF can be used to dynamically synthesize PDF documents composed of a variable number of pages from templates found in PDF documents specified by the FDF, and to populate any fields in the “spawned” pages with data. A template is simply a page with a name attached that can be hidden.

Replacing an HTML Form with Acrobat Form

Figure 1.1 shows the simplest case, which permits existing CGI applications to be used without modification. To use Acrobat forms in such a system, you create:

1. An Acrobat form with field names that match those in the existing HTML form.

2. A button on the form whose action is a submit form action. The URL to submit to may be relative (to the URL of the form that you are submitting from).

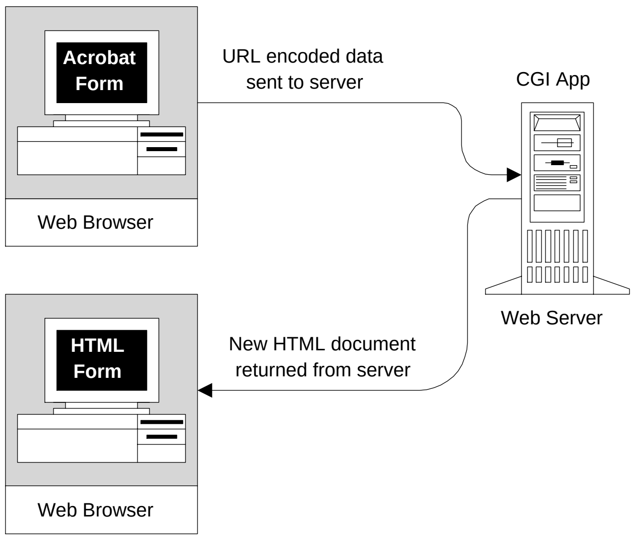

Acrobat Form
Web Browser
URL encoded data sent to server
CGI App
Web Server
HTML Form
New HTML document returned from server
Web Browser

FIGURE 1.1 Using HTML with Acrobat Forms

Scenarios for Using FDF

This section describes additional typical scenarios for using FDF.

Acrobat Form with Results Returned in the Same Form

Figure 1.2 shows a system in which the form data is returned into the same form that originally submitted the data. In such a system, the data sent to the server may be either FDF or urlencoded format, while the data returned from the server *must* be in FDF format and have a MIME type of application/vnd.fdf.

NOTE: *When the server returns data in FDF format,* ***and*** *you are submitting from an Acrobat Form, the URL to submit to must end in #FDF; for example, http://localhost.com/cgi-bin/ yourscript#FDF.*

NOTE: *If you return a static FDF file stored on the server, as opposed to one dynamically generated by a server script as in the example below, then you may have to define application/vnd.fdf as a new MIME type on your server.*

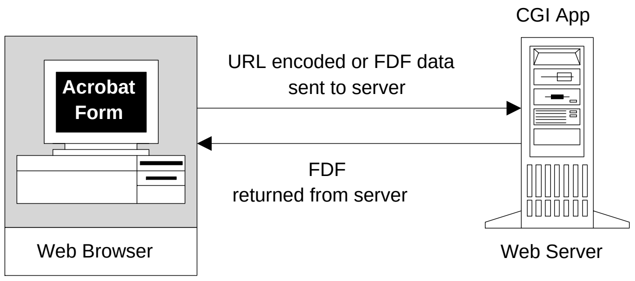


FIGURE 1.2 Returning FDF into the Same Form

Existing CGI applications must be modified to return FDF instead of HTML documents. Figure 1.3 shows a simple FDF-generating Active Server Page (ASP) that works with the Microsoft Internet Information Server 3.0. For anything more complicated than this example, the use of the FDF Toolkit is highly recommended.

FIGURE 1.3 Simple ASP for generating FDF

```
<%@ LANGUAGE = VBScript%>
<% Response.ContentType = "application/vnd.fdf " %> %FDF-1.2
1 0 obj
<<
/FDF << /fields [
<< /T (status)/V (Hello, World!) >>
] >>
>>
endobj
trailer
<<
/Root 1 0 R
>>
%%EOF
```

Acrobat Form with Results Returned in a New Form

In some cases, you may wish to return data into a form different from which it was submitted. Figure 1.4 shows such a case. The CGI application needed to implement such an application is almost identical to that used in the system described in the previous section. The only difference is that you must include an **/F** key in the FDF file when you want to populate a different form. The value of the **/F** key specifies the URL for the PDF file to populate with the forms data. This URL may be relative (to the URL of the form that you are submitting from).The specified file is retrieved (from the server) and populated with the forms data.

NOTE: *If the returned FDF is for the same form that you submitted from, it should not include the /F key.*

The data sent to the server may be either FDF or urlencoded format. The data returned from the server *must* be in FDF format and have a MIME type of application/vnd.fdf.

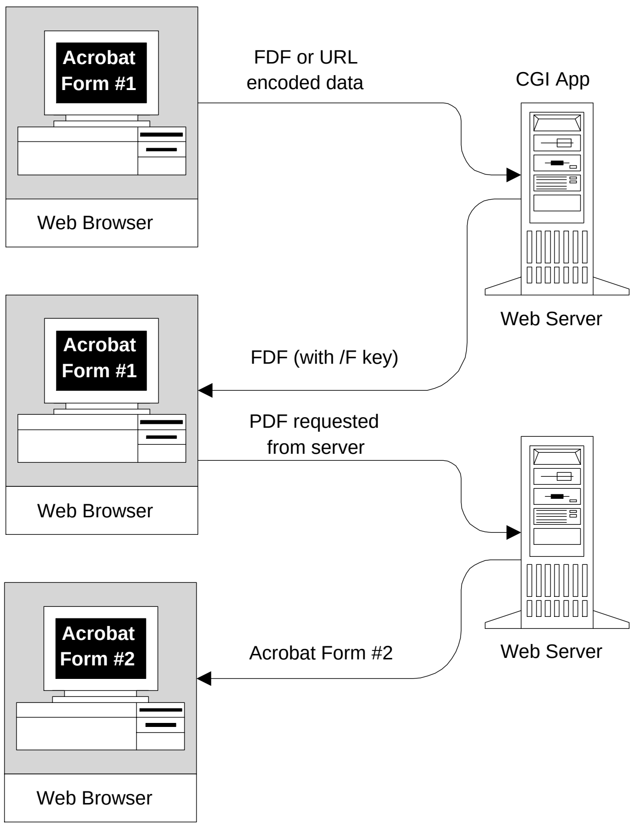


FIGURE 1.4 Returning FDF into a Different Form

HTML Form with Results Returned in an Acrobat Form

An additional possibility is starting from an HTML form, submitting to the server, and having the server return an FDF whose **/F** key is the absolute URL of an Acrobat form. The specified form is retrieved and populated with the forms data. Figure 1.5 shows such a system.

NOTE: *For this to work, it is necessary that you select Acrobat as the application that handles the MIME type "application/vnd.fdf". For example, if you are using Netscape Communicator 4.x, you do this under Preferences -> Applications. Choose Acrobat as the "helper" application. If you are using Internet Explorer (on the Microsoft Windows platform), open Windows Explorer, choose the menu item View -> Options -> File Types, and make sure there is an entry for "Adobe Acrobat Forms Document" which has the "Default Extension for Content Type:" set to "FDF" and the "Content Type (MIME):" set to "application/vnd.fdf". Additionally, the checkbox "Confirm open after download" should be unchecked. Finally, under "Actions:" there should be an entry for "open", and in its properties, "Application used to perform action" should be set to wherever Acrobat resides, and the checkbox "Use DDE" should be unchecked. It also is imperative that from within Acrobat you go to the File -> Preferences -> Weblink menu item and choose your browser. All of these settings are taken care of during Acrobat installation, if Acrobat is installed* ***after*** *the Web browser.*

NOTE: *For the case being discussed in this section the URL to submit to does not need to end in #FDF.*

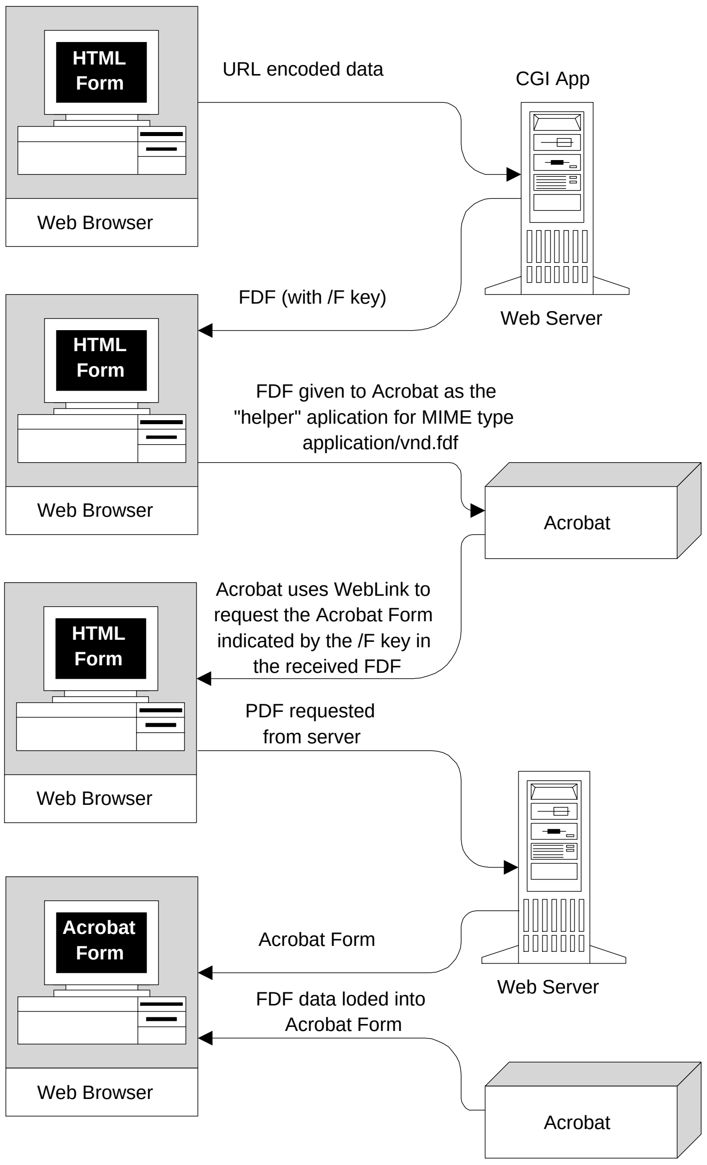

HTML Form
Web Browser
URL encoded data
CGI App
Web Server
FDF (with /F key)
HTML Form
Web Browser
FDF given to Acrobat as the "helper" aplication for MIME type application/vnd.fdf
Acrobat
HTML Form
Web Browser
Acrobat uses WebLink to request the Acrobat Form indicated by the /F key in the received FDF
PDF requested from server
Web Server
Acrobat Form
Acrobat Form
Web Browser
FDF data loded into Acrobat Form
Acrobat

FIGURE 1.5 Start from HTML, end in an Acrobat form

Templates

Acrobat Forms has the ability to dynamically create PDF documents composed of a variable number of pages from templates found in PDF documents specified by the FDF, and to populate any fields in the “spawned” pages with data carried by that FDF file. Figure 1.6 shows such a system.

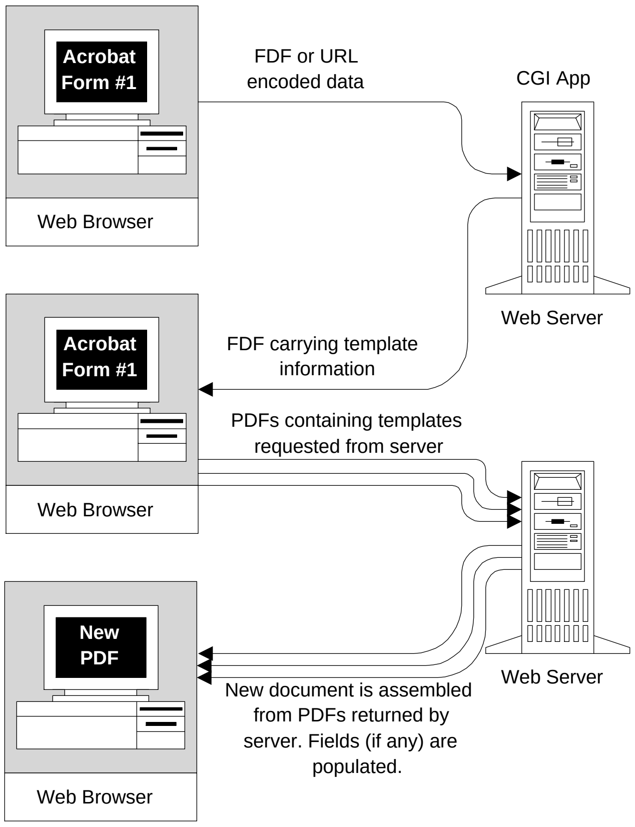

Acrobat Form #1
Web Browser
FDF or URL encoded data
CGI App
Web Server
Acrobat Form #1
Web Browser
FDF carrying template information
PDFs containing templates requested from server
Web Server
New PDF
Web Browser
New document is assembled from PDFs returned by server. Fields (if any) are populated.

FIGURE 1.6 Dynamic Creation of a PDF from Templates

Acrobat lets you define a page in your document as a template, which can then be used to dynamically generate a new form, or duplicate PDF pages on the fly. This allows you to build a form that dynamically creates another form.

Templates are useful in several ways:

- They allow the user to fill out as many form pages as needed. Additional pages (complete with new form fields) are spawned on the fly. For information on defining an action that dynamically creates new pages, choose Help -> Forms JavaScript Guide to display the *Acrobat Forms JavaScript Object Specification*.

IMPORTANT: 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.

To define a template:

1. Navigate to the page you want to use as a template, and choose Tools -> Forms -> Page Templates.
2. Enter a name for the template, and click Add. Click Yes in the confirmation dialog box.
3. Click Close to define the template and close the Document Templates dialog box.

To edit a template:

1. Choose Tools -> Forms -> Page Templates.
2. Select the desired template in the list, and do one of the following:
   - To hide the selected template page, click the eye icon to the left of the template name. To show the template, click the icon again. When you show a hidden template page, it appears appended to the end of the document. You cannot hide a template page if it is the only page in the document. If you delete a hidden template page, it is deleted from the PDF file.
   - To change the template contents to the current displayed page, click Change.
   - To remove the selected template from the list, click Delete.
   - To display the selected template page, click Goto. You cannot use Goto to display a template that is hidden.
3. Click Close to accept the template changes and close the Document.

There are two types of FDF:

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

NOTE: *You cannot add templates to a classic FDF. If you try, you will get an error code returned (for example., FDFErcIncompatibleFDF) indicating an incompatible FDF. Many of the calls in the FDF Library (for example, FDFSetFile) are incompatible with a template-based FDF (an FDF for which FDFAddTemplate has been called. Other calls (for example, FDFSetValue) work with either kind of FDF. If called with a template-based FDF, they act on the most recently added template.*

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*.

A new feature added in Acrobat 4.0 provides an alternative way of transmitting and receiving data via FDF. 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 4.0, the only possible value of the **/Encoding key** is *Shift-JIS*.

2 FDF and Web Server Connectivity

Building Server Side Applications With FDF

The last chapter defined the Form Data Format, the FDF file structure, and provided a general overview of how the Form Data Format (FDF) is used. This chapter takes a step by step look at how to use FDF in a Web environment.

One of the many features in Acrobat is the ability to electronically fill out a PDF-based form. Previously, a user would have to download the PDF file, print the PDF file to a printer, fill out the areas of the form (by either writing or typing the information onto the page), scan and convert the form back to a PDF and somehow transfer it back to the original owner. This was a time-intensive process.

Acrobat does away with such redundancies and allows the PDF file to contain annotations that represent text fields, action buttons, radio button, check boxes, list boxes, and combo boxes. The data from this PDF-based form can be exported in HTML or FDF format.

> **NOTE:** *Exporting data in FDF format has a number of advantages such as populating the same PDF file on the client with new data, sending graphical information back to a PDF file, altering a PDF file on the client, or constructing a PDF file with templates. These features will be detailed later in this chapter. 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.*

The FDF file can be parsed to extract data from the individual fields. This could then be used 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.

The FDF Toolkit

To build applications that parse and/or generate FDF data on a Web server, you need the FDF Toolkit. If you have not already downloaded the FDF Toolkit, it is available from the Adobe developers Web site at:

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

The next sections discuss parsing and generating FDF data using the FDF Toolkit. These sections cover how to initialize, parse, create, save, and free resources with the Toolkit. Sample code illustrates the use of the FDF Toolkit.

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. The application can generate a response (acknowledging some action has taken place) or do some other processing on this data. Examples of this include populating a database with the incoming form data or processing the field data to be used in an order entry system. The last step involves cleaning up any open network connections and closing any open data buffers.

To illustrate this, let's say we have a company that sells office supplies. We have an online catalog that users can fill out and then submit the items they wish to purchase using an order form page in our catalog. An application program could be written to parse the FDF 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.

Initializing the FDF Toolkit

On UNIX systems, you must first call *FDFInitialize* to initialize the Toolkit. Other calls to methods in the API can be made only after the Toolkit has been initialized.

Initializing the FDF Toolkit is only needed when using the C/C++ and Java versions of the library. There is no need to initialize the Toolkit for the Perl or ActiveX versions of the library.

When developing on Microsoft Windows systems, it is not necessary to make calls to initialize and finalize the library. This takes place when the FDF data is opened by your application. Once opened, you may make calls to the rest of the API and use *FDFClose* to close any open FDF files.

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, we use the *FDFOpen* command and pass the following parameters:

- The pathname to the FDF file or, to read from *stdin*, “-” if we open the FDF file coming in from the Web client.
- The number of bytes to read. If you are opening a file, pass “0” instead.
- A pointer to an FDFDoc object. This pointer will be referenced by the rest of the API.

For example you would call *FDFOpen* like this:

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

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. This key points to the value of the field. For example:

```
%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, we reference the *FDFDoc* object passed by the *FDFOpen* method and specify the field we are interested in and the buffers that will be used to store the data. The data returned in this buffer is used by the rest of your application (for example, the address or name of a customer). Here is an example how *FDFGetValue* is used to get the field values of a field named "Customer.Name":

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

Cleaning Up and Finalizing the Library

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

Sample Application

Below is a sample application written for C that parses a simple PDF file coming in from a Web client. This application could also be written in Java, Perl, or VBScript (for use with Microsoft's Active Server Pages) by using the other flavors of the FDF Toolkit. Later chapters of this technical note discuss the other versions of the FDF Toolkit. 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.

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

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


void main()

     {

     /*
     ** Definitions
```

```
*/

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

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

/*
** FDFOpen:
*/

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

/*
** FDFGetValue:
*/

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

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

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

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

   // Your code goes here

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

```
        */

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


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

        errorCode = FDFClose(FdfInput);


}
```

Setting Up 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.

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 PDF that it is for. The PDF is automatically loaded by Acrobat upon opening the FDF. The **/F** key is set with the *FDFSetFile* method discussed in the next section.

> **IMPORTANT:** 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.

Populating a PDF File

Once you have parsed the data from the PDF file, you could use the FDF Toolkit to generate FDF data to populate a form with new data and return it to the user. The next section discusses using the FDF Toolkit to generate FDF data.

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, the *FDFDoc* object can be referenced by other methods in the Toolkit 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 and clean up and close any open buffers.

The steps to create an FDF file are:

1. Initialize the library.
2. Call methods to create the FDF data.
3. Set the values of the individual fields (the /V key in the FDF data).
4. Set actions for fields.
5. Save the FDF data.
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.

We will now detail the steps of generating FDF data with the FDF Toolkit.

Initializing the FDF Library

On UNIX systems, you must first call *FDFInitialize* to initialize the library before you can create FDF. Other calls to methods in the API can be made only after the library has been initialized.

> IMPORTANT: Initializing the FDF library is only needed when using the C/C++ and Java versions of the FDF Toolkits. It is not done when using the Perl or ActiveX versions of the Toolkit.

When developing for Microsoft Windows systems, it is not necessary to make calls to initialize and finalize the library. This takes place when the FDF data is created by your application. Once the FDF data is created, you may make calls to the rest of the API and use *FDFClose* to close any open FDF files.

Creating FDF with the Toolkit

To build FDF data to be used in a file or by a buffer, you use the *FDFCreate* method. One of the parameters you pass is a pointer to an *FDFDoc* object. The *FDFDoc* object represents the Forms Data Format data used in a form field. This object is referenced by other methods in the library.

In C, you would create an FDF object by calling the *FDFCreate* method like this:

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

Setting Field Values

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

FDFSetValue takes as arguments:

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

For example, in C you would use *FDFSetValue* like this:

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

In this example, we set the field value of the text field "Customer.Address" to the value of "12 Saratoga Ave".

The last parameter passed tells Acrobat viewers below Acrobat version 3.01 (and that do not have the Acrobat Forms version 3.5 Update) to convert the new value of the field (in this case "12 Saratoga Ave") to a PDF string. If *true* was passed, it would be converted to a PDF name before making it the value of the field. In previous versions of the Acrobat Forms plug-in, radio buttons and check boxes took PDF names as values. For PDF names, the new value passed for radio buttons or check boxes must be "Off" or a value that was entered as the "Export Value" when defining the properties of the form field. The other field types, such as text boxes, take PDF strings.

If the clients are running Acrobat 4.0, or Acrobat 3.01 or above with the Acrobat Forms 3.5 Update, then the application should just pass *false* in all cases, without regard to the field type. However, if there is a possibility that some clients might run Acrobat 3.0 or 3.01 without the Acrobat Forms 3.5 Update, then your application should pass the correct value (whether it be *true* or *false*) for the last parameter.

Performing Actions and Additional 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, which 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.
- *JavaScript* — Execute a JavaScript script.

For example, to change the value of a submit button whose action was to submit the FDF data from a Web server named “Alpha” to a Web server named “Beta”, you use the FDF Toolkit method *FDFSetSubmitFormAction*:

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

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

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

Targeting FDF Data at a PDF File

Because FDF data is used to populate a PDF file, you also may 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 locally to the client.

To do this, you would 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.

For example, 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* like this:

```
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 “-”.

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

The following code is a sample application that shows how you may use the FDF Toolkit to generate FDF data. This code was written on a Windows NT system with Visual C++ v5.0:

```
/*************************************************************************

 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()

{
     FDFErc retCode;
     FDFDoc FdfOutput = NULL;


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

/*
     FDFSetValue.
*/

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

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

/*
     Next we'll do three things:

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

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

     We want a record of the transaction, so save the FDF data to a
specific
     place on the hard drive.
*/
/*
     FDFSave (FdfOutput,
        "C:\\InetPub\\wwwroot\\Pdf\\generated-data.fdf");
*/

/*
     Close the FDF File used to generate the output.
*/
     retCode = FDFClose (FdfOutput);

}
```

Debugging Tip

You can emit the MIME type “plain/text” (instead of “application/vnd.fdf”) when sending the FDF. For example:

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

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

Handling Errors

The table below lists the error handling codes used by all versions of the FDF Toolkit.

Error Codes	#	Description
FDFErcOK	0	No error.
FDFErcInternalError	1	Internal FDF Library error.
FDFErcBadParameter	2	One or more of the parameters passed to the function are invalid.
FDFErcFileSysErr	3	Error using the file system. Generally this includes “file not found” errors.
FDFErcBadFDF	4	FDF file being opened or parsed contains invalid data.
FDFErcFieldNotFound	5	The *fieldName* parameter contained field name that does not exist in the FDF.
FDFErcNoValue	6	A field with no value had its value requested.
FDFErcEnumStopped	7	Enumeration was stopped by *FDFEnumValuesProc* returning *false*.
FDFErcCantInsertField	8	The field name in *fieldName* cannott 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”. Or, if you try to insert a fieldName of “a.b.c” into an FDF file that already has “a.b”.
FDFErcNoOption	9	The requested element in a field’s **/Opt** key does not exist, or the field has no **/Opt** key.
FDFErcNoFlags	10	The field has no **/F** or **/Ff** keys.
FDFErcBadPDF	11	The PDF passed as parameter to *FDFSetAP* is invalid, or does not contain the page passed in parameter *pageNum*.
FDFErcBufTooShort	12	The buffer passed as parameter is not long enough for the data that the function is returning. Increase the buffer space allocated.
FDFErcNoAP	13	The field does not have an **/AP** key.
FDFErcIncompatibleFDF	14	Cannot mix classic and template-based FDF data.

Part II

Language Specific Overview

3 FDF Toolkit for C/C++

Introduction

This chapter describes the methods used with the C/C++ version of the Forms Data Format (FDF) Toolkit. The FDF Toolkit gives any server running UNIX or Microsoft Windows NT Server the capability to generate or parse FDF data for or from a form created by Acrobat Forms Author plug-in.

The Acrobat Forms plug-in allows anyone using Acrobat to create forms on PDF files. A user with the Acrobat viewer and the Acrobat Forms Fill-in plug-in can fill in and submit the form to a server.

Once the user fills out the PDF-based form, the data is submitted to the server for processing. The server application can then parse the data from the form fields using the FDF Toolkit for FDF data. Once the data has been processed, you can use the FDF Toolkit to generate a return response (for example, acknowledging some action has taken place) to the user.

Building Applications with the FDF Toolkit

To prepare your development environment for use with the FDF Toolkit, copy the C header file, *fdftk.h*, into your project's directory. Include the header file into your file:

```
#include “Fdftk.h”
```

This chapter contains a summary of each of the FDF Toolkit methods available for C/C++. Descriptions of these methods are also provided.

UNIX Support

On UNIX systems, you must first call *FDFInitialize* to initialize the library. Other calls to methods in the Toolkit can be made only after the library has been initialized. When done making calls to the library, call *FDFFinalize*.

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

Microsoft Windows Support

When developing on Microsoft Windows systems, it is not necessary to make calls to *FDFInitialize* and *FDFFinalize*. Make calls to the rest of the API and use *FDFClose* to close any open FDF files.

FDFTK.LIB is a Win32-specific import library. To link the library, add it to the project. Make calls to the API using the examples provided at the end of each method.

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.

FDF Toolkit Methods

This section is a brief description of the FDF Toolkit methods in the following categories:

- General methods to manipulate FDF files
- UNIX and Mac OS specific methods
- Methods for parsing FDF data from a file or a buffer
- Methods for generating FDF data from a file or a buffer
- Cleaning up and closing FDF files

General

Method	Description
FDFGetVersion	Verifies the correct version of the library.
FDFOpen	Opens existing FDF file or read stdin in C.
FDFRemoveItem	Removes key-value pairs in the FDF file.

UNIX and Macintosh Specific

Method	Description
FDFInitialize	Initializes the library.
FDFFinalize	Frees resources in the library.
FDFRegisterThreadsafeCallback	Creates threadsafe callback.

Parsing FDF Data

Method	Description
FDFEnumValues	Enumerates field values in the FDF file.
FDFGetAP	Gets the appearance of a field (/**AP**).
FDFGetEncoding	Gets the value of the FDF /**Encoding** key as a string.
FDFGetFile	Gets the value of the /**F** key.
FDFGetFlags	Gets the /**Ff** or /**F** flag.
FDFGetOpt	Gets the value of the /**Opt** key.
FDFGetStatus	Gets the value of the /**Status** key.
FDFGetValue	Gets the value of a specific field.
FDFNextFieldName	Gets field names in the FDF file.

Generating FDF Data

Method	Description
FDFAddTemplate	Adds a template to an FDF file.
FDFCreate	Creates a new FDF file.
FDFSetAP	Sets the appearance of a field (/**AP**).
FDFSetAPRef	Sets the /**APRef** key.

Method	Description
FDFSetEncoding	Sets the value of the **/Encoding** key.
FDFSetFile	Sets the value of the **/F** key.
FDFSetFileEx	Sets the value of the **/F** key.
FDFSetFlags	Sets the **/Ff** or **/F** flag.
FDFSetGoToAction	Sets **/A** or **/AA** key to type *GoTo*.
FDFSetGoToRAction	Sets **/A** or **/AA** key to type *GoToR*.
FDFSetHideAction	Sets **/A** or **/AA** key to type *Hide*.
FDFSetIF	Sets the Icon Fit attribute.
FDFSetImportDataAction	Sets **/A** or **/AA** key to type *ImportData*.
FDFSetJavaScriptAction	Sets **/A** or **/AA** key to type *JavaScript*.
FDFSetNamedAction	Sets **/A** or **/AA** key to type *Named*.
FDFSetOpt	Sets the value of the **/Opt** key.
FDFSetResetByNameAction	Sets **/A** or **/AA** key to type *ResetForm*.
FDFSetResetFormAction	Sets **/A** or **/AA** key to type *ResetForm*.
FDFSetStatus	Sets the value of the **/Status** key.
FDFSetSubmitByNameAction	Sets **/A** or **/AA** key to type *SubmitForm*.
FDFSetSubmitFormAction	Sets **/A** or **/AA** key to type *SubmitForm*.
FDFSetURIAction	Sets **/A** or **/AA** key to type *URI*.
FDFSetValue	Sets the value of a field.

Cleaning Up and Closing FDF Files

Method	Description
FDFClose	Closes an FDF file.
FDFSave	Saves FDF data.

4 FDF Toolkit for ActiveX

Introduction

This chapter describes the methods used with the ActiveX version of the Forms Data Format (FDF) Toolkit. The FDF Toolkit, of which the ActiveX component for Win32 is a part, allows you to generate FDF data or parse FDF data for or from a form created by Acrobat Forms Author plug-in.

The FDF Toolkit gives any server running UNIX or Microsoft Windows NT Server the capability to generate or parse FDF data for or from a form created by Acrobat Forms Author plug-in.

The Acrobat Forms Author plug-in allows anyone using Adobe Acrobat 4.0 or higher to create forms on PDF files. A user with the viewer and the Acrobat Forms Fill-in plug-in can fill in and submit the form to a server.

Once the user fills out the PDF-based form, the data is submitted to the server for processing. The server application can then parse the data from the form fields using the FDF Toolkit for FDF data. Once the data has been processed, you can use the FDF Toolkit to generate a return response (for example, acknowledging some action has taken place) to the user.

Building Applications with the FDF Toolkit

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

The examples used in this document are Microsoft Visual Basic or VBScript. In an ASP environment, it is assumed you are using VBScript.

Installation

FdfAcX.dll should be installed in a directory that has “execute” permissions. In the case of Microsoft’s Internet Information Server version 3 using ASP, a good location can be `\WINNT\system32\inetsrv\ASP\Cmpnts`. In the case of IIS version 4, `\WINNT\system32` can be used.

If you are running on an NTFS file system, the DLLs themselves need the proper execute permissions. *FdfAcX.dll* uses *FdfTk.dll*. Both should be placed in the same directory, or in the `C:\WINNT\System32` directory.

FdfAcX.dll needs to be registered with Windows NT. This is accomplished by using 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 type the following command at Windows NT Command Prompt window:

```
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*. From Visual Basic 5.0, you might use the object like this:

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

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

Use the methods the same as you would in VB.
```

Setting Up the NT 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 of the Windows NT Server. If your Web site includes files that are in multiple formats, your computer must have a Multipurpose Internet Mail Extension (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. See the Windows NT registry for the default MIME mappings.

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 NT as an asterisk (*). This is the default MIME type used in Windows NT when a MIME mapping does not exist.

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

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

```
HKEY_LOCAL_MACHINE
```

Search through for the following key:

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

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

Using *Regedt32.exe*, the syntax is:

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

Using *Regedit.exe*:

1. From the menu select Edit -> New -> String Value

2. Enter the value:

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

3. Reboot your system.

You should have Windows NT Server 4.0 with Service Pak 3 or later installed .

FDF Toolkit Methods

The methods for ActiveX are broken down into four groups:

- General exposed by the *FDFApp* Object
- Methods for parsing FDF data
- Methods for generating FDF data
- Methods for saving and close FDF files

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).

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

Method Name	Description
FDFCreate	Creates a new FDF file.
FDFGetVersion	Returns the version of the ActiveX component.
FDFOpenFromFile	Opens existing FDF file.
FDFOpenFromBuf	Opens an FDF from a buffer.
FDFOpenFromStr	Opens an FDF from a string.

Of these methods, *FDFCreate*, *FDFOpenFromFile*, *FDFOpenFromBuf*, and *FDFOpenFromStr* return an object of type FDFACXLib.FdfDoc which has 36 methods:

Method Name	Description
FDFAddTemplate	Adds a template to an FDF.
FDFClose	Closes an FDF file.
FDFGetAP	Gets the appearance of a field (/**AP**).
FDFGetFile	Gets the value of the /**F** key.
FDFGetFlags	Gets the value of the /**Ff** or /**F** key.
FDFGetOpt	Gets the value of the /**Opt** key.
FDFGetOptNumElem	Gets the number of elements in the /**Opt** key.
FDFGetStatus	Gets the value of the /**Status** key.
FDFGetValue	Gets the value of a field.
FDFNextFieldName	Gets field names in the FDF data.
FDFRemoveItem	Removes key-value pairs in the FDF data.
FDFSaveToBuf	Saves FDF data to a buffer.
FDFSaveToFile	Saves FDF data to a file.
FDFSaveToStr	Saves FDF data to a string.

Method Name	Description
FDFSetAP	Sets the appearance of a field (/**AP**).
FDFSetAPRef	Sets the appearance of a field (/**APRef**).
FDFSetFile	Sets the value of the **/F** key.
FDFSetFlags	Sets the value of the **/Ff** or **/F** key.
FDFSetGoToAction	Sets **/A** or **/AA** key to type *GoTo*.
FDFSetGoToRAction	Sets **/A** or **/AA** key to type *GoToR*.
FDFSetHideAction	Sets **/A** or **/AA** key to type *Hide*.
FDFSetIF	Sets the Icon-fit attribute.
FDFSetImportDataAction	Sets **/A** or **/AA** key to type *ImportData*.
FDFSetJavaScriptAction	Sets **/A** or **/AA** key to type *JavaScript*.
FDFSetNamedAction	Sets **/A** or **/AA** key to the type named.
FDFSetOpt	Sets the value of the **/Opt** key.
FDFSetResetFormAction	Sets **/A** or **/AA** key to type *ResetForm*.
FDFSetStatus	Sets the value of the **/Status** key.
FDFSetSubmitFormAction	Sets **/A** or **/AA** key to type *SubmitForm*.
FDFSetURIAction	Sets **/A** or **/AA** key to type *URI*.
FDFSetValue	Sets the value of a field.

5 FDF Toolkit for Java

Introduction

This chapter describes how to use the Java versions of the Forms Data Format (FDF) Toolkit. This FDF Toolkit gives any server running a standard servlet environment the capability to generate or parse FDF data for/from a form created by Acrobat Forms Author plug-in.

The Acrobat Forms Author plug-in allows anyone using Acrobat 4.0 or higher to create forms on PDF files. A user with the Acrobat Reader and the Acrobat Forms Fill-in plug-in can fill in and submit the form to a server.

Once the user fills out the PDF-based form, the data is submitted to the server for processing. The server application can then parse the data from the form fields using the FDF Toolkit for FDF data. Once the data has been processed, you can use the FDF Toolkit to generate a return response (for example, acknowledging some action has taken place) to the user.

Differences Between Java Implementations

There are two Java toolkits implemented for FDF. There is a standard Java implementation and an implementation based on ActiveX. The standard implementation runs on either a Windows-based operating system or Solaris and uses a shared object library for implementing the native calls needed to communicate with Acrobat. The ActiveX version only runs on the Windows platform.

Building Applications with the FDF Toolkit

The requirements for each of the Java toolkits are:

- The JNI version of the Java FDF Toolkit requires installation of *FDFTK.jar* and *FDFTK.dll*. On Solaris, *libjFdfTk.so* should be installed. Follow the directions of your servlet runner for installation procedures. When building your code, the *FDFTK.jar* file needs to be added to the classpath for your development environment.
- The ActiveX version of the Java FDF Toolkit is based on, and requires, the ActiveX component of the FDF Toolkit. See the demo under Credit Card Demo for sample usage of this toolkit.

To install the ActiveX Java toolkit: Copy *FdfApp.class*, *FdfDoc.class*, *IFdfApp.class*, *IFdfDoc.class*, and *summary.txt* to a new folder: *%windir%\Java\TrustLib\fdfacx* (for example, *C:\WINNT\Java\TrustLib\fdfacx*).

The ActiveX Java FDF Toolkit was tested in cgi-bin applications written using VJ++ 1.0 under Internet Information Server (IIS). Use the following procedure.

1. Use *Regedt32.exe* to open *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ScriptMap*.

2. From the Edit menu, choose “Add Value”. Enter the value “.class” for the “REG_SZ” data type.
3. Use the String editor to enter **C:\WINNT\Jview.exe %s %s**.
4. Restart the WWW service.

FDF JNI Based Java Toolkit Class Overview

This section is a brief description of the common FDF Toolkit classes:

- FDFTK — for initializing and terminating the FDF toolkit.
- FDFDoc — the main class for parsing and generating FDF files.

For the FDFTK Class

Method	Description
FdfTKInit	Initializes the library.
FDFTKTerm	Frees resources in the library.
Finalize	Removes key-value pairs in the FDF file.

General Methods For the Creating and Opening FDFDoc Objects

Method	Description
FDFDoc	Constructs a new FDFDoc.
FDFDoc	Constructs a new FDFDoc from a existing String or from *stdin*.
FDFOpen	Reads an *FDFFile* into memory.

FDFDoc Methods for Parsing FDF Data

Method	Description
GetAP	Gets the appearance of a field (/**AP**).
GetFile	Gets the value of the **/F** key.
GetFlags	Gets the **/Ff** or **/F** flag.
GetOpt	Gets the value of the **/Opt** key.

Method	Description
GetStatus	Gets the value of the /**Status** key.
GetValue	Gets the value of a specific field.
NextFieldName	Get field names in the FDF data.

Methods for Generating FDF Data

Method	Description
AddTemplate	Adds a template to an FDF.
Create	Creates a new FDF file .
SetAP	Sets the appearance of a field (/**AP**).
SetAPRef	Sets the /**APRef** key.
SetFile	Sets the value of the /**F** key.
SetFileEx	Sets the value of the /**F** key.
SetFlags	Sets the /**Ff** or /**F** flag.
SetGoToAction	Sets /**A** or /**AA** key to type *GoTo*.
SetGoToRAction	Sets /**A** or /**AA** key to type *GoToR*.
SetHideAction	Sets /**A** or /**AA** key to type *Hide*.
SetIF	Sets the Icon-Fit attribute.
SetImportDataAction	Sets /**A** or /**AA** key to type *ImportData*.
SetJavaScriptAction	Sets /**A** or /**AA** key to type *JavaScript*.
SetNamedAction	Sets /**A** or /**AA** key to the type named.
SetOpt	Sets the value of the /**Opt** key.
SetResetByNameAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetResetFormAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetStatus	Sets the value of the /**Status** key.
SetSubmitByNameAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetSubmitFormAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetURIAction	Sets /**A** or /**AA** key to type *URI*.
SetValue	Sets the value of a field.

Cleaning Up and Closing FDF Files

Method	Description
Close	Closes an FDF file.
Save	Saves FDF data.

FDF ActiveX Based Java Toolkit Class Overview

Most of the calls for ActiveX are supported, except for:

- *FDFOpenFromStr*
- *FDFOpenFromBuf*

  NOTE: *Instead, save the received data to a temporary file, and use FDFOpenFromFile*

- *FDFSaveToStr*
- *FDFSaveToBuf*

  NOTE: *Use* SaveToFile *passing* "-" *as the filename in order to write to STDOUT*

- *FDFSetSubmitFormAction*
- *FDFSetResetFormAction*

You cannot pass the last parameter (which is optional), representing the names of the fields to submit (or reset), due to the lack of support in Java for the OLE type VARIANT holding an array of strings.

6 FDF Toolkit for Perl

Perl FDF Toolkit Overview for Windows

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

Calls Not Available in the Windows Perl FDF Toolkit

FDFCreate — use *new* instead. For example, `$inFdf = new Acrobat::FDF;`

FDFOpenFromFile — use *new* instead. For example, `$inFDF = new Acrobat::FDF('c:\temp\in.fdf');`

FDFOpenFromStr and *FDFOpenFromBuf* — use *NewFromBuf*. For example, `$inFdf = Acrobat::FDF::NewFromBuf($buf);`

FDFSaveToStr and *FDFSaveToBuf* — se either *SaveToBuf* (`print $outFdf->SaveToBuf();`) or *SaveToFile* (passing '-' as the filename in order to write to *stdout*). For example, `$outFdf->SaveToFile('-');`

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

FDFSetSubmitFormAction and *FDFSetResetFormAction* — you cannot pass the last parameter (which is optional), representing the names of the fields to submit (or reset), due to the lack of support in Perl for the OLE type VARIANT holding an array of strings.

Using the Perl FDF Toolkit

The library must be imported into Perl's namespace via the *use* operator in this manner:

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

See the demo under EmployeeInfoDemo for sample usage of this toolkit.

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

To install the Perl FDF Toolkit, a directory called *Acrobat* must be created in the *lib* folder of the Perl installation. *FDF.pm* should be copied into the *Acrobat* directory. *FDF.pm* was tested under Perl, version 5.003_07 http:www.ActiveState.com. This version assumes that OLE.pm is in the lib folder.

NOTE: *If you use a different version of Perl that expects ole.pm in the Win32 folder, then replace FDF.pm references to "OLE" with "Win32::OLE". For example, instead of "use OLE;" it should be "use Win32::OLE;".*

Also, replace "CreateObject OLE *FdfApp.FdfApp*" with "new Win32::OLE *FdfApp.FdfApp*".

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* (leave out the quotes and substitute the correct path, of course).
5. Restart the Web browser.

Sample Usage of the Windows Perl FDF Toolkit

See the demo called "EmployeeInfoDemo" for sample usage of the Windows version of the Perl FDF Toolkit.

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

FDF Toolkit Methods

General Methods

Method	Description
GetVersion	Verifies the correct version of the library.
Initialize	Initializes the library.
Finalize	Frees resources in the library.
RegisterThreadsafeCallback	Creates a threadsafe callback.
RemoveItem	Removes key-value pairs in the FDF file.

Create a new FDF file-Example:

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

Open existing FDF file or read stdin-Example:

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

or

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

Templates

Method	Description
AddTemplate	Adds a template to an FDF file.

Generating FDF Data

Method	Description
SetValue	Sets the value of a field.
SetStatus	Sets the value of the /**Status** key.
SetFile	Sets the value of the /**F** key.
SetFileEx	Sets the value of the /**F** key.
SetOpt	Sets the value of the /**Opt** key.
SetFlags	Sets the /**Ff** or /**F** key.
SetAP	Sets the appearance of a field (/**AP**).
SetAPRef	Sets the /**APRef** key .
SetIF	Sets the Icon-Fit attribute.
SetSubmitFormAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetSubmitByNameAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetResetFormAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetResetByNameAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetImportDataAction	Sets /**A** or /**AA** key to type *ImportData*.
SetJavaScriptAction	Sets /**A** or /**AA** key to type *JavaScript*.
SetGoToAction	Sets /**A** or /**AA** key to type *GoTo*.
SetGoToRAction	Sets /**A** or /**AA** key to type *GoToR*.
SetURIAction	Sets /**A** or /**AA** key to type *URI*.
SetNamedAction	Sets /**A** or /**AA** key to the type named.
SetHideAction	Sets /**A** or /**AA** key to type *Hide*.

Parsing FDF Data

Method	Description
NextFieldName	Gets field names in the FDF file.
EnumValues	Enumerates fields in the FDF file.

Method	Description
GetValue	Gets the value of a specific field.
GetStatus	Gets the value of the /**Status** key.
GetFile	Gets the value of the /**F** key.
GetOpt	Gets the value of the /**Opt** key.
GetFlags	Gets the /**Ff** or /**F** flag.
GetAP	Gets the appearance of a field (/**AP**).

Cleaning Up and Closing FDF Files

Method	Description
Save	Saves FDF data.
SaveToFile	Saves an FDF to a file (VB VBScript).
SaveToBuf	Saves an FDF to a buffer (VB VBScript).
SaveToStr	Saves an FDF to a string (VB VBScript).

Perl FDF Toolkit Overview for Unix

The UNIX version of the Perl FDF Toolkit is based on the C version of the FDF Toolkit. Most of the calls are similar, except that they don't have the "FDF" prefix. For example, use *GetValue* instead of *FDFGetValue*.

See the demo under EmployeeInfoDemo for sample usage of this toolkit.

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

Calls Not Available in the UNIX 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'});
```

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

Functions Only Available in the UNIX Perl FDF Toolkit

newFromBuf — For example, `$inFdf = Acrobat::FDF::newFromBuf ($buf);`

SaveToBuf — For example, `print $outFdf->SaveToBuf();`

Supported Platforms for the FDF Toolkit Libraries

The Perl version of the FDF Toolkit is supported under the same platforms as the C/C++ FDF Toolkit for UNIX.

The FDF libraries were compiled and tested under Perl, version 5.003 http://www.ActiveState.com. The libraries are not upward compatible with later versions of Perl.

Using the Perl FDF Toolkit

The library must be imported into Perl's namespace via the *use* operator in this manner:

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

See the demo under EmployeeInfoDemo for sample usage of this toolkit.

To install the Perl toolkit, the script *InstallAdvice.pl* should be executed through Perl. It gives further installation instructions based on the configuration of your system.

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

Sample Usage of the Perl FDF Toolkit for UNIX

For sample usage of the UNIX version of the Perl FDF Toolkit, see the demo called "EmployeeInfoDemo" in the "Samples" folder of the SDK.

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

FDF Toolkit methods

General Methods

Method	Description
GetVersion	Verifies the correct version of the library.
Initialize	Initializes the library.
Finalize	Frees resources in the library.
RegisterThreadsafeCallback	Creates a threadsafe callback.
RemoveItem	Removes key-value pairs in the FDF file.

Examples

Create a new FDF file-Example:

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

Open existing FDF file or read stdin-Example:

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

or

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

Templates

Method	Description
AddTemplate	Adds a template to an FDF file.

Generating FDF Data

Method	Description
SetValue	Sets the value of a field.
SetStatus	Sets the value of the /**Status** key.
SetFile	Sets the value of the /**F** key.
SetFileEx	Sets the value of the /**F** key.
SetOpt	Sets the value of the /**Opt** key.
SetFlags	Sets the /**Ff** or /**F** flag.
SetAP	Sets the appearance of a field (/**AP**).
SetAPRef	Sets the /**APRef** key.
SetIF	Sets the Icon-Fit attribute.
SetSubmitFormAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetSubmitByNameAction	Sets /**A** or /**AA** key to type *SubmitForm*.
SetResetFormAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetResetByNameAction	Sets /**A** or /**AA** key to type *ResetForm*.
SetImportDataAction	Sets /**A** or /**AA** key to type *ImportData*.
SetJavaScriptAction	Sets /**A** or /**AA** key to type *JavaScript*.
SetGoToAction	Sets /**A** or /**AA** key to type *GoTo*.
SetGoToRAction	Sets /**A** or /**AA** key to type *GoToR*.
SetURIAction	Sets /**A** or /**AA** key to type *URI*.
SetNamedAction	Sets /**A** or /**AA** key to the type named.
SetHideAction	Sets /**A** or /**AA** key to type *Hide*.

Parsing FDF Data

Method	Description
NextFieldName	Gets field names in the FDF file.
EnumValues	Enumerates fields in the FDF file.

Method	Description
GetValue	Gets the value of a specific field.
GetStatus	Gets the value of the /**Status** key.
GetFile	Gets the value of the /**F** key.
GetOpt	Gets the value of the /**Opt** key.
GetFlags	Gets the /**Ff** or /**F** flag.
GetAP	Gets the appearance of a field (/**AP**).

Cleaning Up and Closing FDF Files

Method	Description
Save	Saves FDF data.
SaveToFile	Saves an FDF to a file (VB VBScript).
SaveToBuf	Saves an FDF to a buffer (VB VBScript).
SaveToStr	Saves an FDF to a string (VB VBScript).