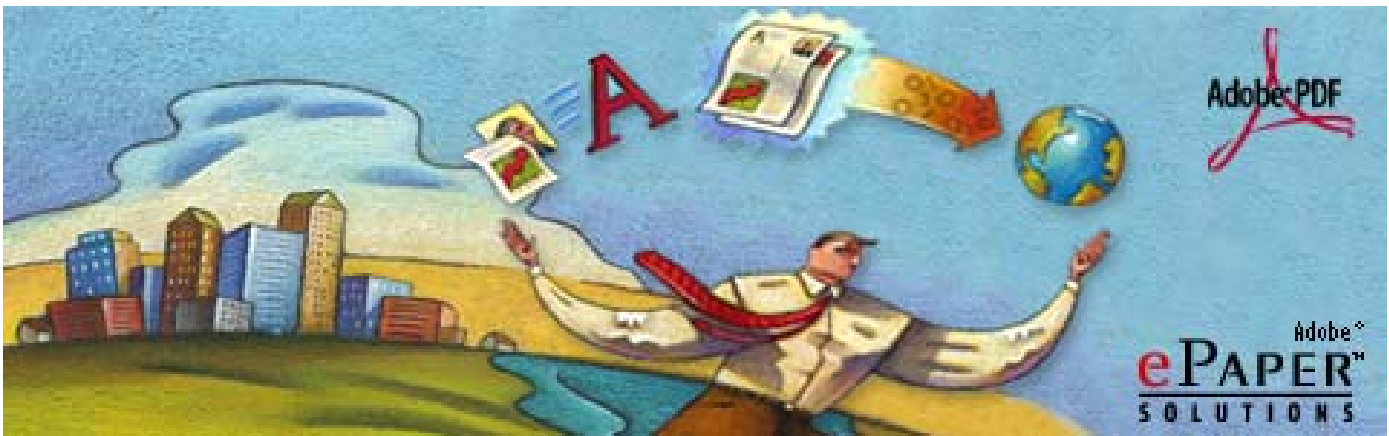




# Adobe® Acrobat®



# Forms System Implementation Notes

© 2001 Adobe Systems Incorporated. All rights reserved.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Adobe, the Adobe logo, Acrobat, Acrobat Capture, Acrobat Exchange, and Distiller are trademarks of Adobe Systems Incorporated. Microsoft and Windows are registered trademarks of Microsoft in the U.S. and other countries. Netscape and Netscape Communicator are trademarks of Netscape Communications Corporation. All other products or name brands are trademarks of their respective holders.

# Forms System Implementation Notes



## 1.0 Introduction

Acrobat Forms is a group of extensions added to PDF in version 1.2, used by the Acrobat products. These extensions allow PDF files to contain fields and buttons. These 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 3.0 or greater.

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 a variety of transmission formats for the forms data:

- URL Encoded Format. This is the format typically used by HTML-based forms. MIME Type: application/x-www-form-urlencoded
- Multipart/form-data. This is also used by HTML-based forms, whenever uploading complete files to the server as part of the submission. MIME Type: multipart/form-data
- Forms Data Format (FDF). This is an Acrobat-specific encoding of forms data similar in syntax to PDF. MIME Type: application/vnd.fdf
- XFDF format. This is an XML-encoded form of FDF. It allows for hierarchical representation of field data but is still not as rich as FDF in terms of the type of data that can be transmitted. MIME Type: application/vnd.adobe.xfdf
- PDF format. There is also the option of submitting the PDF file in its entirety. This is particularly useful when the document contains digital signatures. MIME Type: application/pdf

This document provides an overview of the various architectures for Acrobat Forms applications. It also compares using HTML versus FDF or XFDF as the file format for Acrobat Forms applications.

## 1.1 Prerequisites

Readers are assumed to know the following:

- The process of creating form fields and buttons, and modifying their properties. The Acrobat on-line documentation covers these.

- 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.
- To take full advantage of the capabilities of Acrobat Forms, a knowledge of FDF is required. FDF is described in the Portable Document Format Reference Manual, which is available at <http://partners.adobe.com/asn/developer/acrosdk/DOCS/pdfspect.pdf>.

## 1.2 Other Sources of Information

- A number of sample forms, pointers to good reference material, and third parties that use Acrobat forms can be found at <http://www.adobe.com/products/acrobat/formsresources.html>
- The FDF toolkit is a software library that allows a server process to easily generate and parse FDF files. The FDF toolkit page is located at <http://partners.adobe.com/asn/developer/acrosdk/forms.html>
- Forms can be more “intelligent” by embedding JavaScript commands in the document. This allows for client side formatting, validation, and calculation of data. Information about the JavaScript interpreter embedded in Acrobat can be found by selecting [Help->Acrobat JavaScript Guide](#) in the full product.

## 2.0 The Basics

After a user has filled in an Acrobat form, they must click on a button whose action is a submit form action in order to submit the data to a server. The format of the submitted data may be either HTML-compatible (urlencoded or multipart/form-data), FDF, or XFDF. The selection of which format to use is made at the time the form is created, in the same dialog box in which the form’s creator enters the URL to which the data is submitted.

If HTML is selected, the format is identical to and compatible with an existing HTML form. Existing CGI scripts for HTML forms may be used to parse data in this format.

If FDF is selected, the data format is, not surprisingly, FDF. There is a server library to help parse and generate FDF files. See <http://partners.adobe.com/asn/developer/acrosdk/forms.html>.

If XFDF is selected, the data format is XML. The form data can then be parsed using any generic XML parser.

The URL to submit to is not restricted to the http scheme. It can also be the mailto scheme, e.g. <mailto:someuser@somecompany.com>

---

**Note:** To see the format of the FDF or XFDF 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 Exchange.

---

## 3.0 Choosing the Output Format

This section describes how to choose between HTML and FDF or XFDF 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.
- Appearances (the visual look of a button). This allows you to change the way buttons appear. 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 reprogrammed (including new JavaScripts, change the URL for a submit form action, etc.); field properties can be changed, such as hidden, read-only, required, don't print; listboxes and comboboxes can be populated with different choices, fields can be dynamically created, etc.
- 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 and may be visible in its source PDF document or hidden.
- The FDF can contain digital signature information which makes it possible to recreate the signed document on the server through a simple concatenation process.

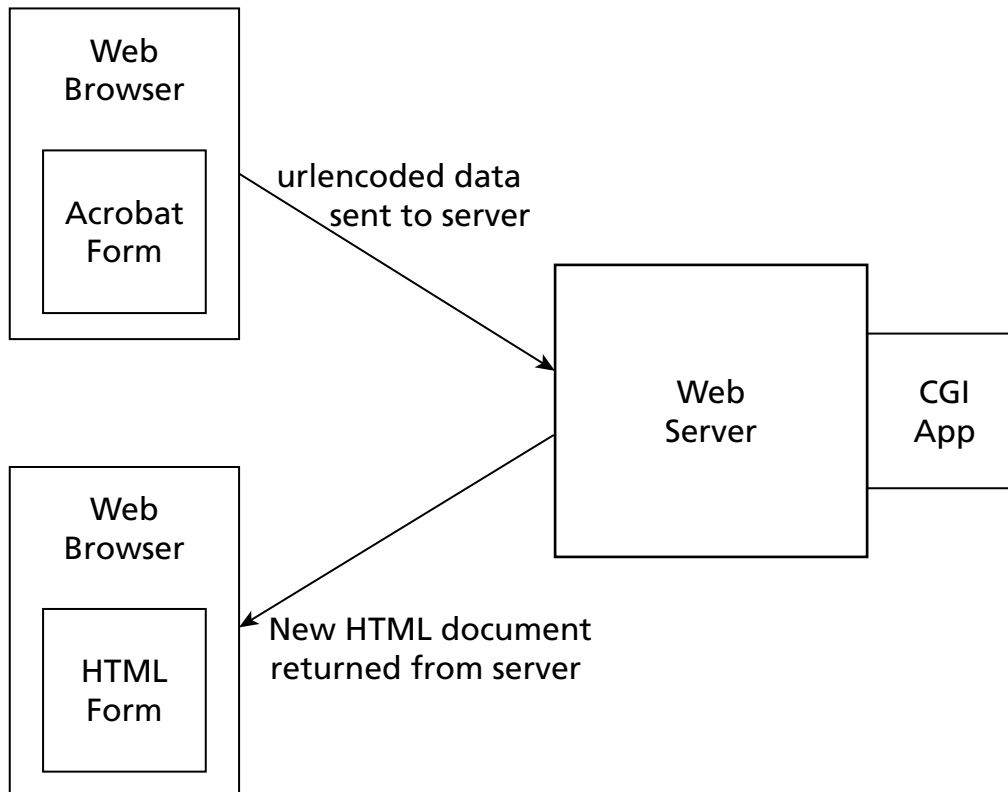
XFDF should be used if an XML-based approach is desired.

## 4.0 Replacing an HTML form with an Acrobat Form

[Figure 1](#) shows this simplest case, which permits existing CGI applications to be used without modification. To use Acrobat forms in such a system, you simply 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).

**Figure 1** Using HTML with Acrobat forms



## 5.0 Acrobat Form with Results Returned in the Same Form

[Figure 2](#) shows a system in which the form data is returned into the same form as that from which it was submitted. This is a very commonly desired situation. In such a system, the data sent to the server may be either FDF or HTML format, while the data returned from the server *must* be in FDF (or XFDF) format and have a MIME type of application/vnd.fdf (or application/vnd.adobe.xfdf, respectively)

---

**Note:** When the server returns data in FDF (or XFDF) format, the URL to submit to *must* end in #FDF, e.g. <http://yourserver.com/cgi-bin/yourscript#FDF>

---

Existing CGI applications must be modified to return FDF instead of HTML documents. [Example 1](#) shows a simple FDF-generating Active Server Page (ASP) that works with the Microsoft Internet

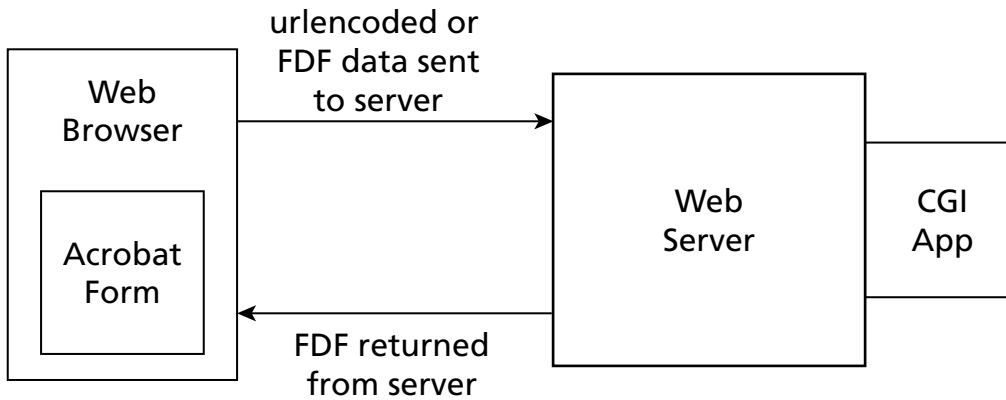
Information Server 3.0. For anything more complicated than this example, the use of the FDF toolkit is highly recommended. See also [1.2 Other Sources of Information](#).

---

**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. The same applies to XFDF.

---

**Figure 2** Returning FDF into the same form



## Example 1 Simple ASP for generating FDF

```
<%@ LANGUAGE = VBScript%>
<% Response.ContentType = "application/vnd.fdf" %>%FDF-1.2

Rem *****
Rem * Parse the inbound data *
Rem *****

Rem *****
Rem * Build FDF Stream *
Rem *****

1 0 obj
<<
/FDF << /Fields [
<< /T (status)/V (Hello, World!) >>
] >>
>>
endobj
trailer
<<
/Root 1 0 R
>>
%%EOF
```

## 6.0 Acrobat Form with Results Returned in a New Form

In some cases, you may wish to return data into a different form, not the form from which it was submitted. [Figure 3](#) shows such a case. The modified CGI application needed to implement such an

application is almost identical to that needed for 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 equivalent functionality is also available in XFDF). 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.

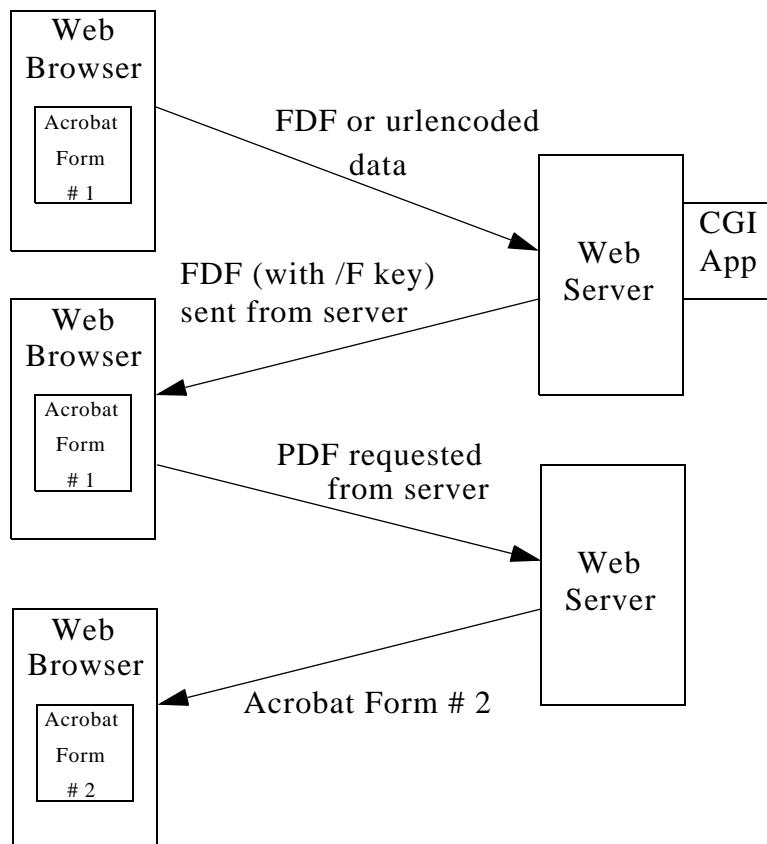
---

**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 HTML format. The data returned from the server *must* be in FDF format and have a MIME type of application/vnd.fdf.

**Figure 3** Returning FDF into a different form



## 7.0 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 gives as value the *absolute* URL of an Acrobat form. The specified form is retrieved (from the server) and populated with the forms data. [Figure 4](#) 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 (e.g. Communicator 4.0x), then 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 C:\Program Files\Adobe\Acrobat 5.0\Acrobat\AcroExch.exe “%1” (or 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. Normally, all of the above is done automatically by the Acrobat installer but if a user’s system is not working you should ensure that the above conditions are true.

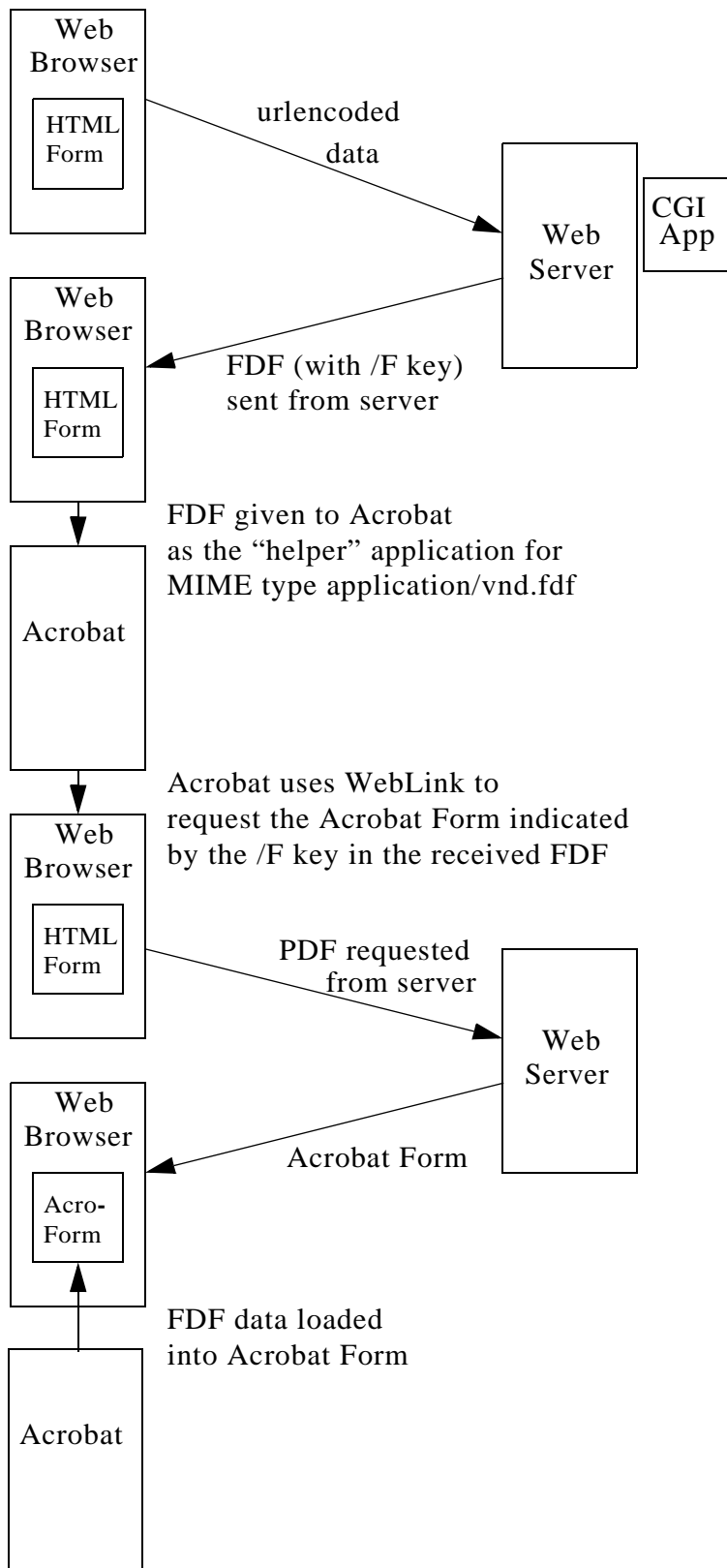
---

---

**Note:** For the case being discussed in this section the URL to submit to does not need to end in #FDF, unlike the cases described before in [5.0 Acrobat Form with Results Returned in the Same Form](#) and [6.0 Acrobat Form with Results Returned in a New Form](#)

---

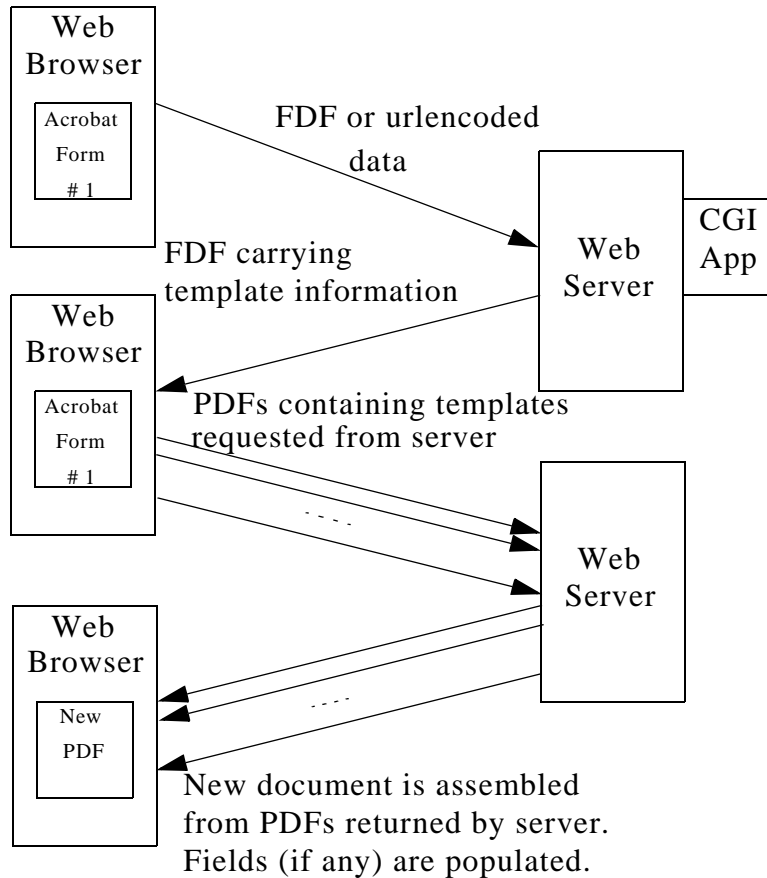
**Figure 4** Start from HTML, end in an Acrobat form



## 8.0 Templates

A recent addition to Acrobat Forms is 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. [Figure 5](#) shows such a system.

**Figure 5** Dynamic creation of a PDF from templates



## 9.0 XFDF

The XFDF (XML-based FDF) format is a semi-transliteration of FDF to XML. There are a number of excellent public domain XML parsers that can be used to read and write this format.

## 9.1 Schema

The schema for XFDF is detailed below:

```
<schema targetNamespace="http://ns.adobe.com/xfdf/"
  version="1.0"
  xmlns:xfdf="http://ns.adobe.com/xfdf/"
  xmlns="http://www.w3.org/TR/xmlschema-1">

  <element name="xfdf" type="xfdf:xfdfType"/>

  <type name="xfdfType">
    <element name="fields" minOccurs="0" maxOccurs="1"
      type="xfdf:fields"/>
    <element name="f" minOccurs="0" maxOccurs="1">
      <type>
        <attribute name="href" minOccurs="1" type="uri"/>
      </type>
    </element>
    <attribute name="xmlns" minOccurs="1">
      <datatype source="uri">
        <enumeration value="http://ns.adobe.com/xfdf/" />
      </datatype>
    </attribute>
    <attribute name="xml:space" minOccurs="1">
      <datatype source="string">
        <enumeration value="preserve">
          <annotation>
            <info> Especially important for multi-line text
              field values, but declare it at the root for
              compactness, to take advantage of inheritance.
            </info>
          </annotation>
        </enumeration>
      </datatype>
    </attribute>
  </type>

  <type name="fields">
    <element name="field" minOccurs="0" maxOccurs="*"
      type="xfdf:field"/>
  </type>
```

```
</type>

<type name="field">
  <element name="field" minOccurs="0" maxOccurs="*"
    type="xpdf:field"/>
  <element name="value" minOccurs="0" maxOccurs="*"
    type="string">
    <annotation>
      <info>A null value is equivalent to not
        including a value tag at all.
      </info>
    </annotation>
  </element>
  <attribute name="name" minOccurs="1">
    <datatype source="string">
      <pattern value="^[^.]+">
        <annotation>
          <info>Field names may not contain
            periods</info>
        </annotation>
      </pattern>
    </datatype>
  </attribute>
</type>
</schema>
```

## 9.2 Sample Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xpdf xmlns="http://ns.adobe.com/xpdf/" xml:space="preserve">
  <fields>
    <field name="essay">
      <value>Once upon a time, there was a king.
        He had one daughter.</value>
    </field>
    <field name="phone">
      <field name="work">
        <value>418 555-4433</value>
      </field>
    </field>
    <field name="hobbies">
      <value>biking</value>
    </field>
  </fields>
</xpdf>
```

```
        <value>hiking</value>
        <value>gardening</value>
    </field>
    <field name="comments" />
</fields>
<f href="http://foo.com/bar.pdf" />
</xfdf>
```

---

**Note:** Hierarchical field names are represented hierarchically, not by using the dot notation (i.e. there is a field "work" under a field "phone", as opposed to having a field "phone.work")

---

---

**Note:** Each newline in the content of value elements (which originate in multi-line text fields in the AcroForm), shows up as the single LINEFEED character (#xA) inside the XML text. The content of value elements is escaped as required for ampersands, angle brackets, etc.

---

## 10.0 Submitting the Complete PDF

In special situations, a method is needed for submitting to a web server changes made to a PDF document at the client, including digital signatures, such that the server can have an exact snapshot of the document displayed by the client, as it existed at the time of submission. In the case of digital signatures, it is imperative that the server can reconstruct an exact replica of the document as it existed at the time of signing, since the signature includes a hash of the (modified) document at the completion of the insertion of the signature.

With Acrobat 5.0 or greater, it is possible to submit either the entire, changed PDF, or enough data such that the server can reconstruct the PDF at its end. To submit the entire PDF, Acrobat saves a temporary copy of the document with all the changes, and submits that (MIME Type: application/pdf). To submit just the incremental changes, since PDF has the inherent capability to do “append-saves” (i.e. save changes to a document by appending the changed objects to the end of the file, without modifying the initial version), Acrobat submits those “append-saves”, embedded within an FDF. The server can then extract this data from the FDF, append it to the original file that it “served” to the client, and this way recreate the signed document at its end.