Automating the Filling In Of A PDF - Reprise

Whil Hentzen

So you've got a PDF with form fields that needs to be filled in from data in your Visual FoxPro app, and you'd like to automate the process, inserting your VFP data into the form and ending up with a new copy of the PDF on disk, with the data included

Much has been written on the subject, but sadly, it's all WAY out of date. The most recent discussion I've been able to find is dated 07... while VFP hasn't changed since then, the PDF products all have.

This article discusses two different ways to accomplish this task. The first is using Windows Automation with Adobe Acrobat, replete with the licensing costs and associated issues and overhead. The second is using the free PDF Toolkit that merges PDF form data with the empty PDF and produces a filled-in PDF via a call to a simple one line batch file.

There's the story about a guy who takes his car into the shop because it's been behaving erratically. The mechanic listens to the sputtering car, opens the hood, takes a screwdriver out of his pocket, reaches in and turns a screw about a quarter turn.

Then he turns to the owner and says, "That'll be $75."

The owner angrily barks, "I saw what you did there! That didn't even take 2 minutes! Why are you charging me $75 to turn a screw?"

The mechanic responds, "I only charged you a quarter to turn the screw. But knowing which screw to turn, and how far to turn it, that cost $74.75."

Similarly, the code to fill in a PDF is nearly trivial, but getting the syntax to that point can be mind-numbingly frustrating.

Before you begin

We'll assume that you've got an electronic copy of the PDF form, without any data filled in. I'll be using the IRS's W-9 Request for Taxpayer Identification Number and Certification for our examples. A copy is included in the subscriber downloads as "w-9empty.pdf".

Tools you'll need

This article addresses the Adobe Acrobat DC application (www.adobe.com, for a 30 day trial version) and the PDF Toolkit (www.pdflabs.com, click on PDFtk Server.) We'll also use the Foxit Reader (foxitsoftware.com) at one point.

I won't even begin to guess at what all the Acrobat installation does to your machine, other than load up your task bar with lots of warnings and updater mechanisms. Eventually, though, you'll have an option for Adobe Acrobat DC in your Windows menu.

Downloading and installing pdftk_server-2.02-win-setup.exe, on the other hand, results in an 'uninstall' option in your Windows menu and a small PDFtkServer folder in Program Files.

PDF Field Names

Under the hood, a PDF file has names for the fields that the user enters data into. Regardless of which method you use, you'll need to determine those names. A PDF isn't smart enough for you to simply throw data at it and have it be inserted in the right places.

There are several ways you can determine the field names. Unfortunately, the tried and true method of reverse engineering – filling in the PDF and then comparing, via a hex editor, the empty and filled-in versions – doesn't work at all. The internals of a PDF are dark and mysterious, and while we could eventually figure it out, there are better ways to spend the next five years. Using our sample w-9.pdf, we'll include the word 'empty' in the filename so that we can distinguish it from the other files we'll end up with by the end of this article. I'm also going to assume that we're working in our root directory, to make the commands easy to read.

Now, about those field names.

PDF Toolkit

You can use a feature of the PDF Toolkit to output field information from a PDF, like so:

```
f:\> pdftk w-9empty.pdf dump_data_fields
     output w-9empty_fields.txt
```

This produces a text file that lists a variety of information about the PDF.

```
---
FieldType: Text
FieldName:
topmostSubform[0].Page1[0].f1_01_0_[0]
FieldFlags: 8388608
FieldJustification: Left
---
FieldType: Text
FieldName:
topmostSubform[0].Page1[0].f1_02_0_[0]
FieldFlags: 8388608
FieldJustification: Left
---
FieldType: Button
FieldName:
topmostSubform[0].Page1[0].FedClassification
  [0].c1_01[0]
FieldFlags: 0
FieldJustification: Left
FieldStateOption: 1
FieldStateOption: Off
---

(more fields)
```

This file, w-9empty_fields.txt, is included in the subscriber downloads. The second line in each section contains the gold we're looking for. Notice that these field names may be rather obscure, depending on what application created the PDF. I've found that different applications (Adobe Acrobat vs Adobe Live Cycle vs Foxit Phantom Creator vs Amyumi) name their fields differently. Another PDF I've worked with has an output like so:

```
---
FieldType: Text
FieldName: F[0].P1[0].Name[0]
FieldNameAlt: Name
FieldFlags: 0
FieldJustification: Left
---
FieldType: Text
FieldName: F[0].P1[0].Contact[0]
FieldNameAlt: Contact
FieldFlags: 0
FieldJustification: Left
---
FieldType: Text
FieldName: F[0].P1[0].IDnumber[0]
FieldNameAlt:  IDnumber
FieldFlags: 0
FieldJustification: Left
```

If you have any sway over how the PDF is created, you may choose to make your life easier when working with the PDF by choosing a simpler object/naming convention. But you also may just be stuck with what you're handed.

Acrobat Automation

A second method, more difficult to implement but that produces results that are easier to parse, is using Automation via Acrobat to read the PDF and spelunk through the object model. The program, acrobat_get_fields.prg, is included in the source code for this issue. Let's walk through the program.

First, assign the filename. We'll need the name to include the word 'empty' so that we can differentiate the empty version with the filled-in version. You'll see why in a minute.

```
* acrobat_get_fields.prg

lcNaEmpty = "f:\w-9empty.pdf"
```

and then make sure the file is there.

```
if !file(lcNaEmpty)
  =messagebox('Cannot locate: '+lcNaEmpty)
  lcNaEmpty=getfile('pdf','PDF File:',
    'Open',1,'Please select a pdf file.')
  if !file(lcNaEmpty)
    =messagebox('Cannot locate: '+lcNaEmpty)
    return
  endif
endif
```

We could pass the name of the PDF file in as a parameter, but typically, you're going to know what PDF you're working with, and then send a number of records through it, so hard-coding the filename is generally a reasonable choice.

Next, instantiate Acrobat. I have added a series of wait windows because it can take a little while to get it loaded.

```
wait window nowait 'Working...'
AcroExchApp = CreateObject("AcroExch.App")
```

Next, create an object for the document, and open the PDF.

```
wait window nowait 'Working..'
AcroExchAVDoc = CreateObject("AcroExch.AVDoc")
AcroExchAVDoc.Open(lcNaEmpty, "")
wait window nowait 'Working'
AcroForm = CreateObject("AFormAut.App")
```

And here comes the magic part. Create a collection of the fields in the form.

```
AcroFields = AcroForm.Fields
```

Now we can spin through the collection to determine just what the field names were. Additionally, let's echo the value in each field (which should be empty at this point.)

```
li = 0
for each Acrofield in AcroForm.Fields
  with AcroField
  li = li+1
  ? ':', li, ':', acrofield.name, ':',
    acrofield.value, ':'
```

```
  endwith
next
wait window nowait 'Done...'
return
```

The output, echoed to the screen, looks something like this:

```
:1 : topmostSubform[0] :  :
:2 : topmostSubform[0].Page1[0] :  :
:3 :
topmostSubform[0].Page1[0].Address[0] :  :
:4 :
topmostSubform[0].Page1[0].Address[0].f1_04_0_
[0] :  :
:5 :
topmostSubform[0].Page1[0].Address[0].f1_05_0_
[0] :  :

(more)

:29 : topmostSubform[0].Page1[0].f1_07_0_[0] :
:
```

Adobe Reader XML Export

A third method is to use Adobe Reader to export the fields to an XML file. After opening the PDF in Reader, select the View | Extended menu option or click on the Extended link in the upper right corner to open the Extended Features pane on the right. See **Figure 1**.

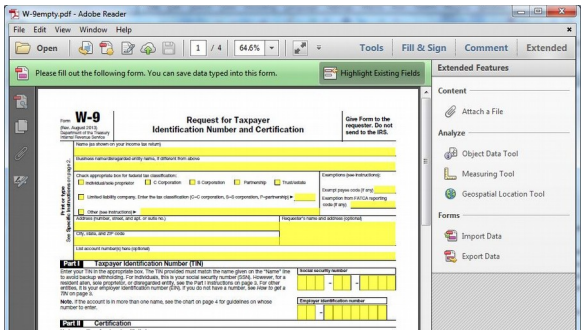

Figure 1. The Export Data option under the Extended link.

Click Export Data and name the output file as desired. The result will look something like this, in part:

```
<?xml version="1.0" encoding="UTF-8"?>
<topmostSubform
><Page1Header
xmlns:xfa="http://www.xfa.org/schema/xfa-
data/1.0/" xfa:dataNode="dataGroup"
/><f1_01_0_
/><f1_02_0_
/><FedClassification
><f1_18_0_
/><f1_50_0_
/></FedClassification
><Exemptions
><f1_100
/><f1_101
/></Exemptions
><Address
><f1_04_0_
/><f1_05_0_
/></Address

(more)
```

```
</Col1
><Col2
xmlns:xfa="http://www.xfa.org/schema/xfa-
data/1.0/" xfa:dataNode="dataGroup"
/><PrivacyActNotice
xmlns:xfa="http://www.xfa.org/schema/xfa-
data/1.0/" xfa:dataNode="dataGroup"
/><c1_01
/></topmostSubform>
```

The entire XML file, named w-9empty_data.xml, is included in the subscriber downloads.

Foxit Export to PDF

Finally, a fourth method, even more work, but at the same time, with results that are even easier to use than the last, is to export a filled-in PDF to an FDF. ('FDF' stands for 'Form Data File', and, thus, saying 'FDF file' is sort of like saying 'baud rate'. Anyways, what an FDF is and why it's a desirable goal, I'll cover in a moment.)

Unfortunately, more and more applications are shying away from the FDF file format and to XML. At this writing, the only program I've found that creates an FDF is the free Foxit Reader.

After opening the empty PDF in Foxit Reader, select the Forms | Export Form Data | To Form Data File (FDF)... menu option and name the output file as desired. The result will look something like this, in part:

```
%FDF-1.2
1 0 obj
<</FDF<</F<</Type/Filespec/F
(/F/W-9empty.pdf)/UF
(/F/W-9empty.pdf)>>/Fields[
<</T(topmostSubform[0].Page1[0].f1_01_0_[0])>>
<</T(topmostSubform[0].Page1[0].f1_02_0_[0])>>
<</T(topmostSubform[0].Page1[0].FedClassificat
ion[0].c1_01[0])/V/Off>>
<</T(topmostSubform[0].Page1[0].FedClassificat
ion[0].c1_01[1])/V/Off>>
<</T(topmostSubform[0].Page1[0].FedClassificat
ion[0].c1_01[2])/V/Off>>
<</T(topmostSubform[0].Page1[0].FedClassificat
ion[0].c1_01[3])/V/Off>>
(more)
<</T(topmostSubform[0].Page1[0].EmployerID[0].
f1_11[0])>>]>>>>
endobj
trailer
<</Root 1 0 R>>
%%EOF
```

The entire FDF, named w-9empty.fdf, is included in the subscriber downloads.

So why is this a desirable goal? Because not only do you get the field names, they're already in the format that we'll need for solution #2, using the PDF Toolkit. I'll come back to this in that section of this article.

Automation with Adobe Acrobat

If you've used Automation before, the general concept is fairly straightforward. You instantiate an Acrobat object from within VFP, create a document object from the empty PDF file, stuff values into the PDF, and save the result under a new filename. The trick, of course, is the syntax.

Let's walk through the program, acro_fillin_pdf.prg.

We've been through first segment of this program, as it's the same as acrobat_get_fields.prg, described earlier. As mentioned earlier, the name of the source PDF includes the word 'empty', so that we can distinguish it from the filled-in version we're going to create in a minute.

```
* acro_fillin_pdf.prg

lcNaEmpty = "f:\w-9empty.pdf"
if !file(lcNaEmpty)
  =messagebox('Cannot locate: '+lcNaEmpty)
  lcNaEmpty=getfile('pdf','PDF File:',
    'Open',1,'Please select a pdf file.')
  if !file(lcNaEmpty)
    =messagebox('Cannot locate: '+lcNaEmpty)
    return
  endif
endif

wait window nowait 'Working...'
AcroExchApp = CreateObject("AcroExch.App")

wait window nowait 'Working..'
AcroExchAVDoc = CreateObject("AcroExch.AVDoc")
AcroExchAVDoc.Open(lcNaEmpty, "")
wait window nowait 'Working'
AcroForm = CreateObject("AFormAut.App")

AcroFields = AcroForm.Fields
```

Now we have our collection of fields. Last time, I demonstrated how to spin through the collection to determine just what the field names were. Let's take that one step further and update the fields. We'll use the field number, first, for proof of concept, but also to help us identify which field corresponds to which object on the PDF form. It'll be fun, really!

```
li = 0
for each Acrofield in AcroForm.Fields
  with AcroField
    li = li + 1
    lcNaField = acrofield.name
    lcValue = acrofield.value
    * put the field number in the field
    acrofield.value = alltrim(str( li ))
    * display the before and after values
    ? ':', li, ':', acrofield.name,
      ': old value:', lcValue,
      ': new value:', acrofield.value, ':'
  endwith
next
```

Finally, now that we've got new values in the document object, let's create a new PDF. As mentioned, the original file includes 'empty' in the file name. We'll create a new filename that replaces the word 'empty' with 'filled', and then adds a timestamp to the end, as it's likely that we'll run this multiple times, and may want to keep track of versions produced.

```
AcroExchPDdoc = AcroExchAVdoc.GetPDdoc
lcNaOutput = strtran(lcNaEmpty,
  'empty','filled')
lcNaOutput = addbs(justpath(lcNaOutput))
  + juststem(lcNaOutput) + '_'
  + strtran(substr(ttoc(datetime()),12,8),
    ':','') + '.'+justext(lcNaOutput)
```

And now we'll actually save the file and clean up. I've added a line that will output a note to the screen that confirms that the save worked, instead of requiring all the work of switching to your file manager to see if the new PDF is there.

```
AcroExchPDdoc.save(1,lcNaOutput)
AcroExchAVDoc.Close(.T.)
AcroExchApp.Exit
? iif(file(lcNaOutput),'TRUE', 'nope')
wait clear
return
```

The result is shown in **Figure 2**.

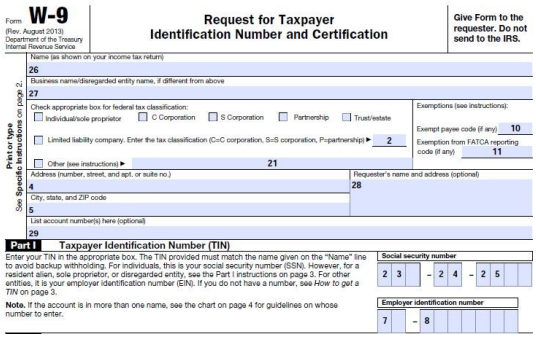

Figure 2. A test run of the filled in W-9 form.

So far, so good. But users probably aren't going to be interested in seeing their PDF form with a bunch of random numbers in the fields. We'll need to put the real VFP data in there. Now that we've identified which field number maps to which field on the form, we can add a bit of logic to stuff the right value in the right place.

First, we'll need to grab the data from the table as appropriate:

```
select ;
  name, busname, llctype, other, exempt, ;
  fatca, address, csz, accountno, requestor, ;
  ssn1, ssn2, ssn3, ein1, ein2, ;
  iid ;
  from CUST
  into array laCust
```

(Note that I group the fields in collections of five, in order help keep the array index straight, and add the unique PK at the end.)

Then, while spinning through the collection, interrogate the field number and stuff the appropriate value in the document.

```
for liRow = 1 to alen(laCust,1)

with AcroField
  li = li + 1
  lcNaField = acrofield.name
  lcValue = acrofield.value
  * put the value in the field
  do case
  case li = 26
    acrofield.value = laCust[liRow, 1]
  case li = 27
    acrofield.value = laCust[liRow, 2]
  case li = 2
    acrofield.value = laCust[liRow, 3]
  case li = 10
    acrofield.value = laCust[liRow, 4]
  case li = 11
    acrofield.value = laCust[liRow, 5]
  case li = 21
    acrofield.value = laCust[liRow, 6]
  case li = 4
    acrofield.value = laCust[liRow, 7]
  case li = 5
    acrofield.value = laCust[liRow, 8]
  case li = 29
    acrofield.value = laCust[liRow, 9]
  case li = 28
    acrofield.value = laCust[liRow, 10]
  case li = 23
    acrofield.value = laCust[liRow, 11]
  case li = 24
    acrofield.value = laCust[liRow, 12]
  case li = 25
    acrofield.value = laCust[liRow, 13]
  case li = 7
    acrofield.value = laCust[liRow, 14]
  case li = 8
    acrofield.value = laCust[liRow, 15]
  other
    acrofield.value = 'x'
  endcase
  * display the before and after values
  ? ':', li, ':', acrofield.name,
    ': old value:', lcValue,
    ': new value:', acrofield.value, ':'
endwith
```

After stuffing the value into the field, I still output the field number, name, original value and new value to the screen for verification that I got everything lined up right.

The end result is shown in **Figure 3**.

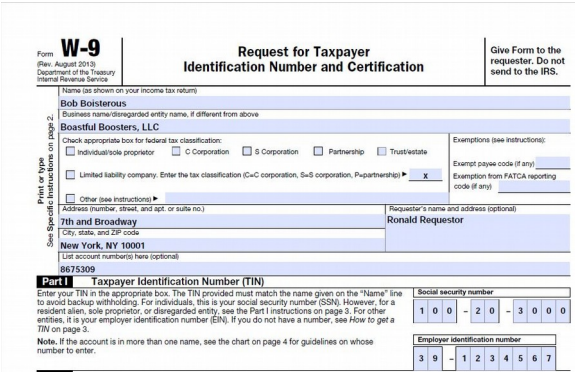

Figure 3. Filling a PDF with actual values, once the field numbers are known.

Note that it's now a trivial task to include an identifier for the record in the fieldname, so that multiple PDFs can be spit out while ripping through a table.

```
* add the PK as part of the filename
lcNaOutput = addbs(justpath(lcNaOutput)) ;
  + juststem(lcNaOutput) ;
  + '_' + allt(str(liRow)) ;
  + '.'+justext(lcNaOutput)
AcroExchPDdoc.save(1,lcNaOutput)

next && liRow
```

PDF-FDF Merge with PDF Toolkit

So for whatever reason, you've decided that you don't want to go with the Acrobat solution. That's ok, I won't judge you. The process we'll go through consists of one fairly simple VFP program that has four parts.

First, we'll stuff a variable with the PDF fields and FDF format via textmerge. Then we'll grab the VFP data and populate the textmerge variables with that data. Third, we'll execute textmerge to create a temporary FDF file, usin that VFP data and the FDF format.

Next, we'll create a text string that contains the command that calls pdftk and passes the input and output filenames. Finally, we'll execute the batch file, creating the filled-in PDF file.

The full program, pdftk_fillpdf.prg, is included in the subscriber downloads.

Stuff a textmerge variable

This is the trickiest part, because if you miss even a single format character, the process won't work. The format we're aiming for looks like this:

```
TEXT TO m.lcMergeText NOSHOW

%FDF-1.2
%âãÏÓ
1 0 obj
<</FDF <</Fields [<<
/Kids [<</Kids [<<
/V (-#lcName#-)
/T (Name[0])
>>
(more fields)
<<
/V (-#lcZip#-)
/T (Zip[0])
>>]
/T (P1[0])>>]
/T (F[0])>>]>>>>
endobj
trailer
<</Root 1 0 R>>
%%EOF

ENDTEXT
You'll notice that the construct
<<
/V (-#Name#-)
/T (Name[0])
>>
```

will be repeated for each field. The string in parens following the /V is the name of the VFP variable that contains the data to be stuffed into the field, and the string in parens following the /T is the name of the PDF field.

Looking back at our four solutions to find the fieldnames in a PDF, you'll see that the FDF produced by Foxit Reader is pretty darn close to what we're going to textmerge, so that's why I mentioned it.

Here's a hint: Before you produce an FDF, stuff the empty PDF with data placeholders, and crank out the FDF so that it contains data values. I use strings like 'AAAAAAAAAA' and 'BBBBBBBB' because the resulting FDF will then have very obvious markers for each field.

Grab the VFP data

The next step is to assign your VFP data to the variables in the FDF section.

```
select ;
  name, busname, llctype, other, exempt, ;
  fatca, address, csz, accountno, requestor, ;
  ssn1, ssn2, ssn3, ein1, ein2, ;
  iid ;
  from CUST ;
  into array laCust

for liRow = 1 to alen(laCust,1)

m.lcName = laCust[liRow,1]

(more fields)

m.lcEin2 = lcCust[laCust,15]
```

Create a temporary FDF file

We're going to merge the VFP data with the FDF format via textmerge, and create a file on disk. From there, we'll use the PDF Toolkit to merge the PDF and FDF to create the filled-in form.

```
m.lcFolder = sys(2003)
m.lcFDFFile = textmerge(m.lcMergeText, .T.,
  "-#", "#-")
strtofile(m.lcFDFFile,
  m.lcFolder + "w-9source.fdf")
```

Assemble a batch file command

Your specific needs may vary, say, if you're creating a series of PDFs, but essentially we're creating a command that calls pdftk, passes the filenames of the empty PDF, the temporary FDF, and the desired filled-in PDF.

```
m.lcBlankForm = addbs(sys(5) + sys(2003))
  + "w-9empty.pdf"
m.lcFilledForm = m.lcFolder + "w-9filled_"
  + allt(str( laCust[liRow,16] ))
  + ".pdf"
m.lcCommand = [pdftk "] + m.lcBlankForm ;
  + [" fill_form "] + m.lcFolder ;
  + [w-9source.fdf" output "] ;
  + m.lcFilledForm ;
  + [" flatten drop_xfa]
```

The last parameters take care of some housekeeping. The flatten option will make form field data a permanent part of the page. The drop_xfa option removes conflicts between older PDF formats and newer PDF technology called XFA.

Run the batch file

Since the resulting command string may extend pass the 240 character limit for RUN commands, we'll store it to a variable and execute indirectly.

```
m.lcBatchFile = m.lcFolder + "fillpdf.bat"
STRTOFILE(m.lcCommand, m.lcBatchFile)
RUN "&lcBatchFile"
```

Voila! The filled in PDF, 'w-9filled_NNNNNN.pdf', is now on disk.

Source Code Notes

The files for this article include the following:

w-9empty.pdf - Request for Taxpayer Identification Number and Certification.

w-9empty_fields.txt – output of PDF Toolkit describing the fields in the w-9empty.pdf.

acrobat_get_fields.prg – parses through a PDF and displays a list of fields on the VFP desktop.

w-9empty_data.xml – output of Acrobat Reader describing the fields in the w-9empty.pdf, in XML format.

w-9empty.fdf – FDF output of Foxit Reader.

acro_fillin_pdf.prg – fills data from a VFP table into an empty PDF and saves the result in a new file.

pdftk_fillpdf.prg – VFP program to create and merge FDF formatted data with an empty PDF and save the result in a new file.

pdftk_fillpdf.bat – temporary batch file created by pdftk_fillpdf.prg.

And a special thanks to...

Tracy Holzer for providing the head start on the Adobe Acrobat syntax and Frank Cazabon on the PDF Toolkit concepts.

Author Profile

Whil Hentzen is an independent software developer based in Milwaukee, Wisconsin (as opposed to Milwaukee, Minnesota, as many people think.) His writing has killed many trees over the years, but none since 2007. He has realized he really sort of misses it. You can reach him at whil@whilhentzen.com