This paper was originally presented at the Great Lakes Great Database Workshop in Milwaukee, Wisconsin in November, 2002.

Integrating PDF Documents into Desktop and Web Applications

Session Number 11

Rick Borup
Information Technology Associates
701 Devonshire Dr Suite 127
Champaign IL 61820
Voice: 217.359.0918
Fax:217.359.0992
Email: rborup@ita-software.com

Overview

The PDF document format has become the de facto standard for device-independent document creation, delivery, and retrieval. With the nearly universal availability of the free PDF reader from Adobe, PDF documents can be made available on every desktop. This session focuses on the needs of developers who want to integrate PDF documents into their applications, with particular emphasis on 3rd-party PDF developer tools such as the Amyuni PDF Converter, and provides

working demonstrations of the creation and use of PDF documents in both desktop and Web applications using Visual FoxPro 7.0.

Overview of PDF (Portable Document Format)

What is PDF

PDF (Portable Document Format) is a document format developed by Adobe Systems, Inc. circa 1991. The concept was first described in a paper entitled "The Camelot Project" by John Warnock, Adobe Systems CEO. This paper is available on the Web (in PDF format, of course – see http://www.planetpdf.com/planetpdf/pdfs/warnock_camelot.pdf) and makes for interesting reading for anyone interested in the history of PDF.

Adobe's objective in creating the PDF format was to devise a method of describing all text, graphics, and fonts in a document and storing that information in a file that could be printed on any printer and viewed on any computer. Some of the key concepts and technologies behind the PDF format are

- *Self-contained file* – no external support files should be required to view and print the file;
- *Device Independence* – documents should be readable and printable on any platform; and
- *Font Rendering Engine* – the ability to reproduce text almost exactly the same as the original without requiring that the actual font be installed on the user's computer.

PDF documents store information about a font (character width, weight, size, etc.) and use this information to render the characters if the font itself is unavailable on the user's machine. This makes for a very portable and compact file. You can, however, embed an actual font in a PDF document if you want to, at the cost of a larger file. If a PDF document contains images, the image files are embedded in the document as opposed to storing a link to an external image file. Image files are stored in a compressed format to help reduce the overall size of the PDF file.

Types of Uses for PDF

PDF documents lend themselves to many types of use, which accounts for their popularity in many types of applications and business processes. The most obvious of course is the ability to view and print a PDF document on virtually any computer. Other business requirements could include the need to convert existing files, stored in other file formats, to PDF; the creation of original PDF documents from applications such as word processing and desktop publishing; and the use of PDF as a mechanism for editing and sharing documents in a group workflow process. A more recent companion technology, FDF (Forms Data Format), provides a mechanism for entering and retrieving information from a form. Of most interest to application software developers, however, is the ability to create PDF documents from within your application, and to deliver these to the user on their PC either locally or over the Web.

Choosing the Right PDF Tool for Development Work

Types of PDF Software

There is a wide range of PDF software products available, each corresponding to one or more of the various types of uses for PDF documents. The most well known and widely used of course is the Adobe Acrobat Reader. The ability of the Acrobat reader to function both as a stand-alone product on your PC or Mac, as well as a browser plug-in for use with PDF documents over the Web, makes it the universal host for PDF documents. Adobe's decision to make the Acrobat Reader available for free has probably been the principal driving factor in the huge growth in popularity of the PDF format: there is virtually no computer than cannot view and print a PDF document, and there is no cost to the user associated with doing so.

The full Adobe Acrobat product is a complete PDF document creation and management tool. If you are a publisher of large, complex documents in PDF format, such as books and manuscripts, then Adobe Acrobat is probably the tool you want to be using. If you need to convert large quantities of existing documents from some other format to PDF, then there are products available that specialize in this task.

As an application software developer, though, you are going to be most interested in products that can be described as PDF printer drivers. These tools give you and your users the ability to print a report directly to a PDF file from an application running on the desktop or on a server. Adobe Acrobat does provide one such tool, the PDFWriter, but you need to buy a license for the full Acrobat product to get it; PDFWriter does not come with the free Acrobat Reader. Partly because of its cost, which is in the range of $200 to $250, and partly because it provides so much additional functionality beyond just a PDF printer driver, the full Acrobat product is probably overkill when all you need is a PDF printer driver. Therefore, Acrobat itself may not be the tool of choice for application software developers, particular those who need to distribute their applications to lots of users, since you'd need a license for each user's machine.

Fortunately, there are many 3rd-party products on the market that specialize in providing a PDF printer driver with a programmatic interface for use by developers, and these can also be a more economical solution. Among these products are two that we'll see later in this paper, namely the activePDF Server and the Amyuni PDF Converter.

Licensing Issues and Cost Considerations

When you're choosing a PDF tool for development work, you'll be concerned with its cost and licensing alternatives. For example, for desktop use you could choose to purchase a single-user license for each desktop. This can be a cost effective solution for an application with only one or just a few users. Using this approach your cost grows linearly with the number of users, though, and at some point it becomes more economical to invest in a developer's license that allow royalty-free distribution.

For example, if a single-user single-platform license for the Amyuni PDF Converter costs $129, and a developer's multi-platform license costs $1150, then your crossover point occurs at 9 users (9 x $129 = $1161, which is greater than $1150). While there are other considerations involved,

which we'll touch on in a moment, based on cost alone the developer's license can quickly become the preferred solution.

If your application is deployed on a server you may need a developer's license or a site license to legally accommodate your needs. Read the fine print in the vendor's license agreement.

Technical Requirements

Technical requirements will also play an important part in your choice of a PDF tool for development work. Among the considerations here are things such as how your application will be deployed: desktop or server, single user or multi user. Another aspect is whether you want the PDF printer driver to be available only to your application, or whether you want it to be permanently installed on the user's machine and to be available, like any other printer, to other applications running on that machine.

You may also be concerned with advanced features beyond the simple ability to generate a single PDF file for a single VFP report. For example, you may want to be able to concatenate several reports together into one PDF file. If security is an issue, as it is on a Web sever for example, then you may need to find a product capable of generating encrypted PDF documents. Also of interest for apps deployed over the Web is the ability to do linearization, or Web optimization, which makes it possible to deliver a large PDF document one page at a time rather than all at once.

Finally, you may also need to be concerned with OS compatibility. Not all of the 3rd-party PDF printer driver products are compatible with all versions of Microsoft Windows®. If you do not have any control over the OS under which your application will be run, then you will need to select a solution that covers the whole range of Windows 95/98/Me/NT/2000/XP. This can limit your choice of products and/or affect the type of license you will need to purchase.

Now let's look at two products that are popular with application software developers, activePDF Server and Amyuni PDF Converter.

activePDF Server

If your application will be deployed on a server, and particularly if it will be used under high stress conditions—frequent hits and simultaneous multi-user access—then activePDF Sever is probably going to be your tool of choice. The activePDF Server (APS) is designed for developers and provides a COM interface with a set of properties and methods that makes it easy to use from VB, VFP, ASP, etc. In addition, the APS is a multi-threaded component and therefore well suited for use in this kind of environment. The basic product comes with two PDF-generation threads, and a three-thread upgrade can be acquired for an additional cost. One limitation is that APS runs only on Windows NT/2000 server, but then if you're running a server what else would you be using? <s> APS is licensed on a per server basis with prices starting at $975 for a 1-server license. While not an inexpensive solution, it is probably one of the best for this kind of use.

Amyuni PDF Converter

The Amyuni PDF Converter, on the other hand, is a great choice for developers writing desktop applications and also for server applications working under light to moderate load conditions. The

Amyuni PDF Converter is designed for developers and provides a COM (DLL and ActiveX) interface for ease of use from VB, VFP, etc. It also comes with an FLL component specifically provided for VFP developers, although you do not have to use the FLL to use Amyuni from a VFP application. The Amyuni PDF Converter runs on Windows 95/98/Me as well as on NT/2000/XP, so it's a good choice for situations where the developer cannot control which OS the user will have.

Although the Amyuni PDF Converter can certainly be run on a server, it is not multi-threaded in the same sense that the activePDF Sever product is. Therefore, when it comes to use with a server-based application, the Amyuni PDF Converter is probably best suited for use in a low stress environment. If you use Amyuni PDF Converter on a server where concurrent multi-user access is a possibility—in other words, if you need to deal with situations where two or more processes serving different Web requests would need to create PDF files at the same time—you may find that the way the Amyuni PDF Converter handles this limits its suitability for this kind of heavier use.

Licensing Amyuni PDF Converter

There is a wide variety of licensing options available with the Amyuni PDF Converter. These are distinguished by three factors:

- Single-user vs. developer
- Single platform vs. multi-platform
- Standard version or Pro version

There is a correspondingly wide range of prices depending on the particular license you choose (see http://www.amyuni.com/en/products/pdf_converter/pricing.php for current information from Amyuni). Prices range from a low of $129 for a single-user single platform Standard version license to $1,322 for a developer's multi-platform Pro version license. Site licenses and licenses for suites of Amyuni products are also available. The Standard version lacks support for encryption and Web optimization, so if these are important to you then you will want to go with the Pro version. As of version 2.0, a developer's license is required for use on a server.

Installing Amyuni PDF Converter on Your PC

Installing the Amyuni PDF Converter on your PC is very straightforward. Simply run the EXE you received when you purchased the product (for example, PDFSUPRO.EXE for the single-user Pro version). This will extract the files to the specified directory and, as of version 2, it also registers the Common Driver Interface component CDINTF.DLL. The CDI is the component that you will use when configuring and running the Amyuni PDF Converter from your VFP application, as we will see shortly.

Optionally, you can install a permanent PDF printer driver on your machine by running the INSTALL.EXE which comes with the product. This file can be found in the directory to which you installed the Amyuni files. By default, the permanent printer driver is installed with the name "Amyuni PDF Converter". This printer name will appear in the list of installed printers on your

machine, and can be selected and used to print to a PDF file from Word, Excel, or any other application on your computer.

Whether or not you install a permanent printer driver, once Amyuni PDF Converter is installed on your computer and the CDI component is registered, you can create, configure, and use a PDF printer driver on demand from your application. We'll see several examples of this below.

Deploying the Amyuni PDF Converter With Your Application

If you are going to distribute the Amyuni PDF Converter with your application and want to install it along with your app on the end user's machine, you have several options. The first is to decide whether to go with the developer's license, which allows royalty-free distribution of the necessary components, or to require your users to acquire their own license. If you're going to have each user acquire their own license, then you (or they) will probably install the Amyuni PDF Converter on their machine separately from your application, following the procedures described above.

If you have chosen to acquire the developer's license, you can distribute the Amyuni files with your app and make installation seamless to the user. In this case, you will need to include the necessary Amyuni files with your application's installation package. The files you need to distribute will depend on the operating system that the application is being installed on. There is a difference between what's required on Windows 95/98/Me vs. NT/2000/XP. The required files are documented in the DISTRIBS.PDF file that comes with the developer's version.

One thing to remember when installing the Amyuni PDF Converter under NT/2000 is that the logged on user needs to have sufficient rights to add a printer. If the end user does not have sufficient rights to do this, then the install should be done by an administrator.

There are basically three ways you can install the Amyuni PDF Converter printer driver on the end user's computer.

1. As part of your application's installation routine, run the Amyuni install utility that comes with the product. It may be a good idea to install with a name other than the default "Amyuni PDF Converter" name so as to avoid a conflict with other applications that might also be using Amyuni. To do this you can specify a different name when you run the install utility, like this:

   ```
   INSTALL "My Amyuni PDF Converter" /s
   ```

   The /s switch tells Amyuni to do a silent install. This is optional.

2. As part of your application's installation routine, register the CDINTF.DLL component and call PDFDriverInit("My Amyuni PDF Converter") followed by a call to the DriverInit("My Amyuni PDF Converter"). The second call is necessary to avoid having the driver uninstalled when the installation process ends. This concept also applies when you're using the Amyuni PDF Converter from within your application, by the way.

3. Register the CDINTFL.DLL as part of your application's install procedure, then call PDFDriverInit("My Amyuni PDF Converter") at the beginning of your application to install the PDF printer driver. Call the DriverEnd() method to remove the printer driver at

the end of your app. Remember that if your user does not have sufficient rights to add a printer, this approach will not work.

As is the case when installing it on your own computer, once the required Amyuni files are present and CDINTF.DLL has been registered you're ready to use Amyuni PDF Converter from your application.

Configuring the Amyuni PDF Converter

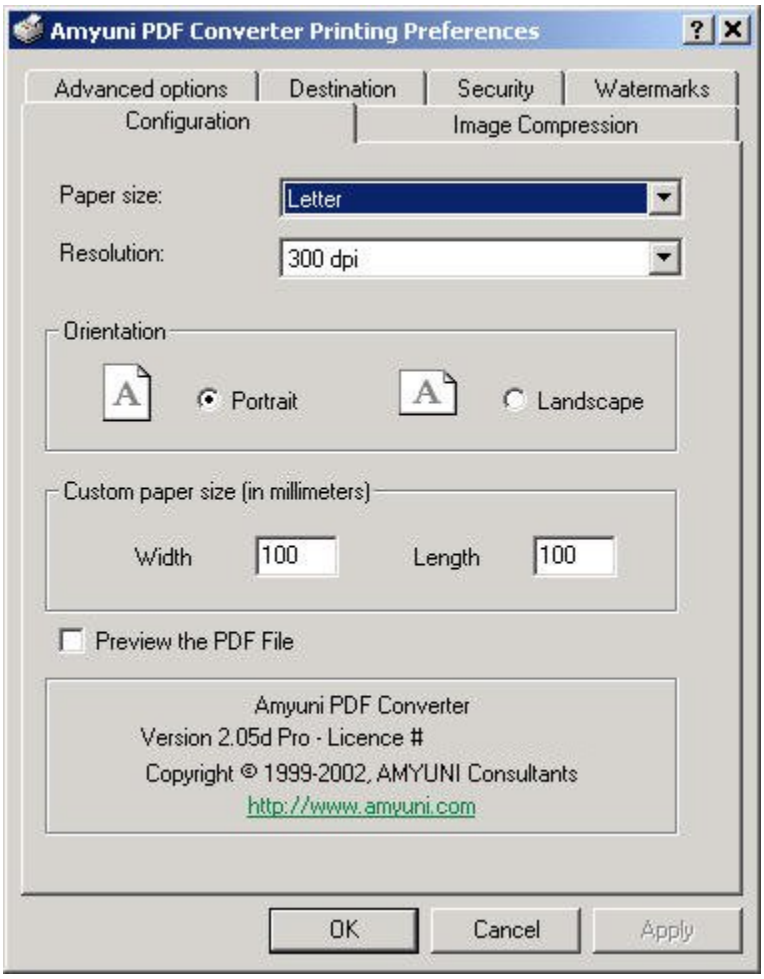
Once installed on the computer, the Amyuni PDF Converter can easily be configured and used just like any other printer. Configuration choices include such conventional items as paper size orientation, and margins, but being a PDF printer driver there are also configuration choices that you don't see with physical printers. These include things like:

- Resolution
- Content compression
- Font embedding
- Image compression
- Destination file type
- and others

Manual Configuration

If a permanent Amyuni PDF Converter printer driver has been installed on your computer, you will find it listed in the Printers window and you can configure it from its properties sheet just like any other printer. (Note that the developer's version does not have a property sheet or a "File Save As" dialog. This is what limits its use to your application, from which you have to configure it programmatically. Without this limitation, Amyuni obviously could not allow royalty-free distribution.)

The properties sheet for the Amyuni PDF Converter 2.05 Pro version looks like this:



Note the tabs for the various groups of configuration choices. Different versions of the Amyuni PDF Converter may have different configuration choices than the ones shown above.

Programmatic Configuration

Configuring the Amyuni PDF Converter programmatically is really the same as doing it manually, except that instead of entering your choices on a property sheet you are calling methods or setting properties of the CDI (Common Driver Interface) object. For example, assuming you have instantiated the Amyuni CDI object as oPDFPrinter, then to set the resolution to 300 dpi programmatically you would write

```
oPDFPrinter.Resolution = 300
```

This is equivalent to choosing “300 dpi” from the Resolution drop-down list on the Configuration tab of the properties sheet.

Sample Code – *PrintToAmyuni.prg*

A sample program to initialize, configure, and print to the Amyuni PDF Converter from a VFP app is included in the source code for this session under the name *PrintToAmyuni.prg*. You’ll want to inspect this code – we’ll refer to it several times as we discuss the demos. As you look it

over, keep in mind that it was designed and written primarily to illustrate the concepts. Although you can use this code "as is", it's not necessarily how you'd really want to do things in a live application. Among the things that would likely be different are:

a) the sample program creates and removes the PDF printer driver on each call. In a real application you would probably want to create the driver once at the beginning of the app and remove it once at the end of the app. Creating the driver is a resource-intensive operation, and you probably do not want to incur this overhead for every report.

b) the sample program has a rudimentary error-handling routine, which does nothing more than retrieve the error message from the Amyuni CDI and display it on the screen. In a real application you would of course want to change this and integrate it with whatever error-handling mechanism your application uses.

c) the sample program is written as procedural code. This simplifies things for the purpose of demonstrating the concepts, but in a real application you would probably want to separate the pieces into functional groups (create the driver, generate the report, and terminate the driver, for example) and integrate them into your application object or your report manager object.

NOTE – I wrote the original version of this program in August of 2000 and published it in the Universal Thread FAQ's and on the FoxPro Wiki as *PrintToPDF.prg.* Some of you may have seen it there. It morphed into *PrintToAmyuni* in the summer of 2001, to distinguish it from a new version I needed to write to drive the activePDF Server product. Although there are some syntactical differences between the code to drive the Amyuni PDF Converter and the code to drive the activePDF Server, the concepts are the essentially the same and the code is very similar.

Integrating PDF into Desktop Apps

The goals when integrating PDF documents into a desktop app in VFP are simple:

1. Print a VFP report to a PDF file
2. Deliver it to the user within your VFP application

Now that you know how to install and configure the Amyuni PDF Converter, these goals are easy to accomplish. Goal #1 is as simple as telling VFP to "print" to the Amyuni PDF Converter instead of to a physical printer. Goal #2 can be accomplished by creating a VFP form inside of which you place an instance of the Acrobat Reader or Web browser control to render the PDF file on the screen.

Let's see some demos. We'll be using the Category Listing report from the TasTrade app as our sample report.

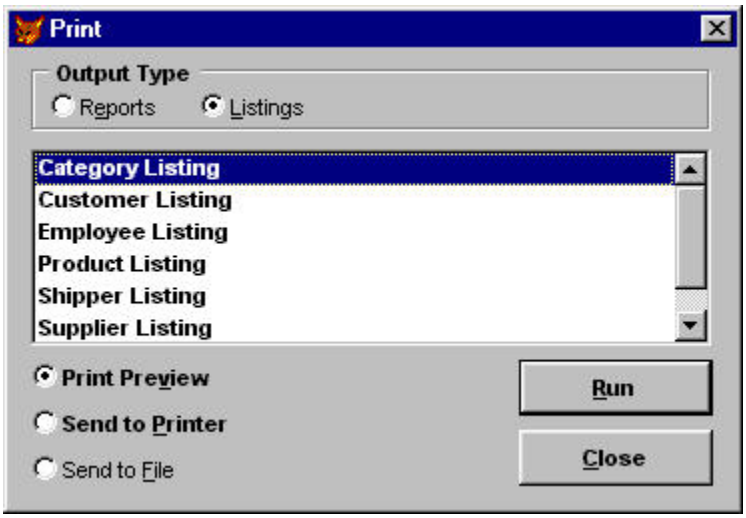
Desktop Demo 1

At the most basic level, you can print a VFP report to a PDF file and then require the user to open it in the Acrobat Reader manually. This accomplishes goal #1 but not goal #2. It works, but it is not very elegant. Nevertheless, it's a good place to start.

In this first demo we'll do everything manually. We simply run the TasTrade app

```
CD HOME(2) + “TASTRADE”
DO TASTRADE.APP
```

and use the Print Setup dialog to choose the Amyuni PDF Converter. Then we go to Print Reports on the File menu, select the Listings radio button, and select the Category Listing report.



With Print Preview selected as shown, run the report. It will appear in the VFP report preview window, which allows you to see what to expect when we look at the PDF. Now, print the report from the preview window. You’ll be prompted for a file name (if not, check your Amyuni PDF Converter properties sheet under Destination and be sure “Prompt for file name” is selected). Specify the desired directory and file name, print the report, then open the PDF you just created in Acrobat Reader. It should look the same as it did in the VFP report preview window.

Demo 2

Now we’ll automate the process a bit by running the report from code and using the PrintToAmyuni.prg program to create the PDF file (again, the PrintToAmyuni.prg program is included in the session’s source code file). The code for this demo looks like this:

```
LOCAL lcCurrDir, lcTasTradeDir, lcReportName, lcPDFFileName
lcCurrDir = SYS(2003)
lcTasTradeDir = HOME(2) + "TASTRADE"
lcReportName = "Reports/ListCat"
lcPDFFileName = ADDBS( lcCurrDir) + "ListCat.PDF"
CD ( lcTasTradeDir)
DO ADDBS( lcCurrDir) + "PrintToAmyuni.PRG" WITH lcReportName, lcPDFFileName
CD ( lcCurrDir)
RETURN
```

A lot of this is simply overhead associated with the session demo, code that identifies the location of TasTrade and returns to the demo directory when done. The important line is the third from the bottom, where the PrintToAmyuni program is called with the name of the report to run and the name of the PDF file to create being passed as parameters.

Inside PrintToAmyuni the two important lines of code are these:

```
oPDFPrinter.DefaultFileName = lcFileName
```

which sets the output file name to the lcPDFFileName we specified, and

```
REPORT FORM ( tcReportName) NOEJECT NOCONSOLE TO PRINTER
```

which prints the report (without having to actually run TasTrade.app).

After this is done, we have programmatically written the report to disk as a PDF file and told Amyuni what file name to use without having to ask the user to specify it. So we've made progress, but we're not there yet because at this point we still need to open the PDF file manually just as we did in demo #1.

Demos 3 and 4

Now we're going to bring a VFP form into the mix and use it as a container to host the Adobe Acrobat Reader. These two demos are essentially the same as one another, except that in demo 3 we are using the Adobe Acrobat Reader ActiveX control directly, whereas in demo 4 we are using the Microsoft Web Browser control and allowing it to invoke the Reader. Both of these forms have a "Load" button which will be used to open the PDF file. These two forms are included in the session source code.

The important line of code in demo 3, which uses the Adobe Acrobat Reader ActiveX control in the form named frmAcrobat, is this:

```
PROCEDURE cmd2.Click
   THISFORM.oleAcrobat.LoadFile( GETFILE( "PDF"))
ENDPROC
```

Upon selecting a PDF file, it is opened in the Acrobat Reader control inside our VFP form and is visible to the user from within VFP!

Demo 4 is basically the same thing except that it uses the Microsoft Web Browser control. The concept is the same but the syntax is a little different: Note the *Navigate* method used below as opposed to the *LoadFile* method used above.

```
PROCEDURE cmdloadpdf.Click
   WITH THISFORM
      .cPDFFileName = GETFILE( "PDF", "PDF File")
      IF NOT EMPTY( .cPDFFileName)
         .oleWebBrowser.Navigate( .cPDFFileName)
      ENDIF
   ENDWITH
ENDPROC
```

The highlighted line is where the action is. The rest of the code is just a slightly different way of getting the name of the PDF file from the user, including a validation step to detect the user canceling out of the GetFile dialog so we don't try to open an empty file name.

Adding a form like this to our app accomplishes goal #2, which is to deliver the PDF file to the user from within our VFP app. However, in both of these demos, the user still needs to load the PDF file into the form manually by clicking on the Load button. In demo 5 we'll do this automatically and dispense with the Load button entirely.

Demo 5

In this demo we complete the picture, accomplishing both goal #1 and goal #2 while also delivering the PDF file to the VFP form automatically. The user does not have to do anything

except request the report (say, from a menu item in your app) and watch it pop up on the screen in PDF format! This provides an effective replacement for the VFP report preview window. The PDF viewer also provides certain advantages over the VFP report preview window, not the least of which is a zoom feature that you can actually read.

The code for this demo can be found in frmPDFForm2 in the session source code. The key difference in this demo as opposed to demos 3 and 4 is that here we pass the name of the PDF file to the form as a parameter. The method code checks that the file exists and if so opens it automatically. This is accomplished partly in the INIT method and partly in the ShowPDFFile method.

The INIT method looks like this:

```
PROCEDURE Init
   LPARAMETERS tcPDFFileName
   IF VARTYPE( tcPDFFileName) <> "C" OR EMPTY( tcPDFFileName)
      MESSAGEBOX( "PDF file name not specified")
      RETURN .F.
   ENDIF
   WITH THISFORM
      .cPDFFileName = ALLTRIM( tcPDFFileName)
      .ShowPDFFile()
   ENDWITH
ENDPROC
```

Note that the PDF file name is stored in the cPDFFileName property of the form, and that the ShowPDFFile method is called if a file name was in fact specified.

```
PROCEDURE showpdffile
   LOCAL lcURL
   WITH THISFORM
      IF EMPTY( .cPDFFileName)
         MESSAGEBOX( "PDF file name not specified")
         RETURN .F.
      ENDIF
      IF EMPTY( JUSTEXT( .cPDFFileName))
         .cPDFFileName = .cPDFFileName + ".PDF"
      ENDIF
      IF EMPTY( JUSTPATH( .cPDFFileName))
         .cPDFFileName = SYS(5) + ADDBS( SYS(2003)) + .cPDFFileName
      ENDIF
      IF FILE( .cPDFFileName)
         *  Need to construct a full URL including "file://" or the
         *  Web browser control tries to treat it as an "http://"
         *  link and fails with an "Action canceled" error.
         lcURL = "file://" + .cPDFFileName
         .oleWebBrowser.Navigate( lcURL)
      ELSE
         MESSAGEBOX( "File " + ALLTRIM( .cPDFFileName) + " not found")
      ENDIF
   ENDWITH
ENDPROC
```

In all of these examples, when using the Web browser control in this manner, be sure to include a NODEFAULT it its Refresh method. This is avoid an OLE error that occurs if you don’t do this.

```
PROCEDURE olewebbrowser.Refresh
   *** ActiveX Control Method ***
```

```
   NODEFAULT
   *  Issue a NODEFAULT to avoide OLE "Unspecified Error" at startup.
ENDPROC
```

That's it! We now have a completely integrated and automated PDF document creation process and document viewer built into our VFP application.

Integrating PDF into Web Apps

The goals when integrating PDF documents in a Web app are essentially the same as when working with a desktop app. In both cases goal #1 is to print a VFP report to a PDF file. The difference in goal #2 is that you are going to be delivering the PDF file to the user via their Web browser instead of inside a VFP form. We'll see three demos on how to do this after we talk about some special considerations when working over the Web.

Special Considerations Working with PDF in Web Apps

Four particular areas of concern arise when you're working with PDF documents over the Web. These are

- No user interface
- Performance considerations
- Security concerns
- Scalability issues

In some cases these may also be considerations when working with PDF in desktop apps, but they take on increased importance when working over the Web.

No User Interface

It goes without saying that an application running on an unattended Web server should never raise a user interface element (messagebox, dialog window, etc.), if for no other reason than the obvious one that nobody will be there to attend to it. This means that you've got to be much more careful in designing your code so that there are no intentional or unintentional user interface requests raised by your application. In particular, you want to be sure there are no "File Save As" dialog windows generated by your PDF generation code: you've got to specify the file name programmatically. Fortunately, this is easy to do with a product such as the Amyuni PDF Converter, as we have already seen.

NOTE – One thing that can trip you up and result in an unintentional dialog window or other unwanted results is failing to clear the TAG, TAG2, and printer-specific EXPR information from the first record of the FRX file of your reports. Failing to do this can cause several problems, among them that a "File Save As" dialog box may be displayed even if you've specified a destination file name (I have seen this occur with activePDF Server, for example).

You can clear TAG and TAG2 entirely, but you will want to clear only the printer-specific information from the EXPR column entirely because if you clear it entirely you could lose important information such as page orientation.

Clearing these fields can be done manually or you can write a small program to do it. The best solution I've seen, though, is Rick Schummer's excellent *VFP Project Builder* utility, which has this tool built in. You can download this tool from http://www.rickschummer.com.

Performance Considerations

The speed with which your app can return a PDF file upon request is much more important on the Web than on the desktop. For one thing, people simply aren't going to wait as long for something to happen over the Web, particularly if they can't seen any sign that progress is happening. Slow response can lead, at best, to frustrated users and, at worst, to actual timeouts of the requested page or repeated clicks on the browser's "refresh" button. The solution to speed problems is faster hardware, more memory, finely tuned applications, and a fast PDF generator.

Reliability is also much more important on the Web, where 24x7 functionality is the norm. This means that your code must be bulletproof, and when it isn't, exceptions must be handled cleanly. Again, this means obvious things like no Messagebox( ) error messages on the server, as well as the need for a more sophisticated error handling routine. An error log should be maintained on the server, for your benefit as the developer, and an easily understandable, non-technical message should be sent back to the browser so the user knows that something happened, even if there's nothing to be done about it right away.

Security Concerns

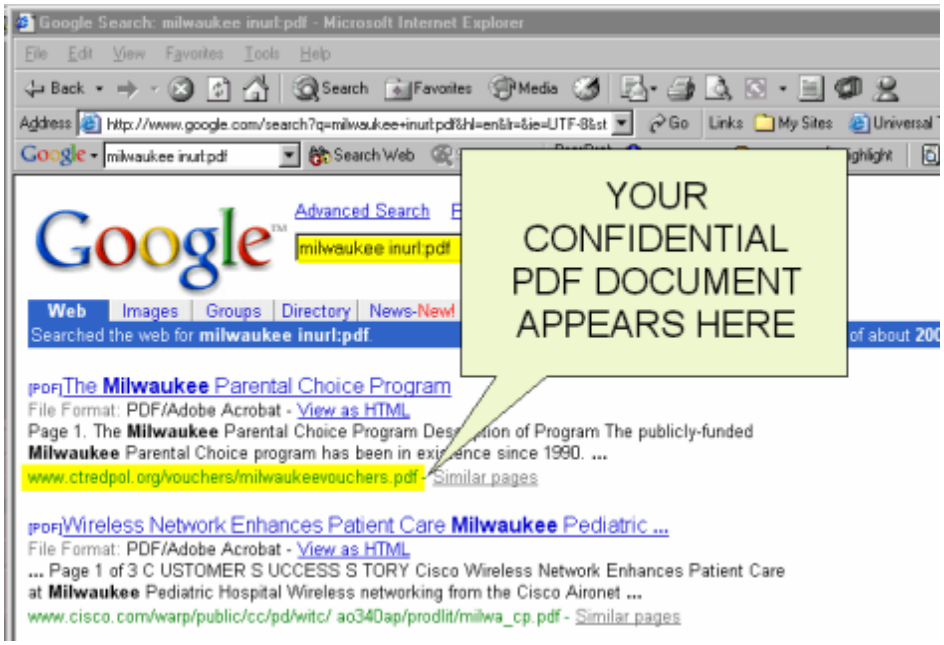
Two aspects of security are important here, both dealing with the privacy and confidentiality of the information contained in the PDF file.

The first has to do with the security of the information while it's in transit from the server to the browser. This can be addressed in one of two ways: either use a secure socket layer (SSL) on the server, and/or encrypt the PDF file before sending it to the browser. The ability to encrypt a PDF file is not available in all PDF printer driver products, but is part of the Amyuni PDF Converter Pro version. One advantage to using an encrypted PDF file is that it remains encrypted if the user stores it on their machine, which they can do from Acrobat Reader. One disadvantage is that a password is required to view it, of course.

The second aspect of security has to do with the fact that, unless you take measures to prevent it, PDF files you create and leave on your server in public directories may be found by search engines and can therefore also be found by anyone searching the Web. In case you didn't know it, you can append "inurl:pdf" to a Google search to find PDF files that match your search criteria. Naturally, this is bad if one of the PDF files that pops up is your company's payroll report or some other equally confidential information.

One way to address this issue is to create some process that's tied to a timer and that removes the PDF files from the server at a pre-determined interval, for example every 15 minutes. Another way to deal with removing the PDF files is discussed as part of demo 3 below. Also, if you place the PDF files in their own sub-directory, you can put a robots exclusion file in the root directory of your Web site that instructs search engine robots to ignore certain sub-directories. This is a simple text file you can easily create with Visual Notepad <s>. One reference to the robots exclusion technology can be found at http://www.robotstxt.org/wc/exclusion.html.

For illustration purposes, below is an example of what Google returns for a search on “Milwaukee inurl:pdf”. I don’t think the documents shown here are confidential, but the picture illustrates the point.



Scalability Issues

If you know that your Web app is going to have limited activity—perhaps it’s for private use by a small company—then you may not have to be concerned about what happens when dozens or hundreds of people start hitting your Web server at the same time. But if your app is serving the public at large, then scalability becomes a real concern and will impact both your choice of product and the design of your PDF solution. Basically, if your Web application allows multiple concurrent users then your PDF solution needs to be able to handle that, too. If not, you will at best have one user’s request waiting in line behind another user’s request, or at worst you may experience a PDF printer driver conflict when two processes try to use it at the same time. If these are concerns in your application, then a multi-threaded product such as activePDF Server is probably your best choice.

Even if multiple concurrent use is not an issue, you need to think about average loads and peak loads on the server. What is satisfactory performance when you’re only getting one or two hits a minute may be very unsatisfactory when you’re getting dozens of hits a minute, particularly if each request for a PDF file has to wait in line for the ones ahead of it to finish.

Delivering PDF to the Web Browser

We’re almost there. We know how to create a PDF file from our VFP Web app (the same as we do from our VFP desktop app), and we know some of things we should be thinking about when we deploy our solution on a Web server. All that remains is to learn how to get the PDF file back to the browser so the user can see it. There are at least three ways to do this:

- Return a page with a link to the PDF file (file based)
- Use HTTP redirect to show the PDF file directly (file based)
- Send PDF back to the browser as a string (streaming output)

Let's look at some code. Because these are demos we'll be using the same PrintToAmyuni.prg program we used with the desktop demos, but be reminded that the way some things that are done in the sample program, in particular the error handing, are not done the way you'd want to do them in actual use in a Web app. Also, these demos were created using West Wind Web Connection. While some of the syntax and constructs are specific to Web Connection, the concepts are universal and can be applied to whatever web server product you may be using.

Demo 1 – Return a page with a link to the PDF file

In this demo we create a PDF file from our VFP Web app and construct a page with a link on it that points to the PDF file on the server. When the user clicks on the link, the browser requests the PDF file from the sever, the Adobe Acrobat Reader plug-in is invoked in the browser and the file is rendered on the user's machine.

This approach requires an extra step by the user, namely clicking on the link, but it has the advantage that you can communicate other information to the user on that page as well as providing the link to the PDF file.

The code for this demo, as well as the other two Web demos, is contained in the program PDFDemoProcess.prg that is included in this session's source code. Because it's somewhat lengthy in its entirety, it's not reproduced in this paper. Please refer to "FUNCTION Demo1" in PDFDemoProcess.prg for the code that drives this demo.

The main thing to notice about this code is its similarity to the desktop demos insofar as creating the PDF file is concerned. The same basic code, as implemented in PrintToAmyuni.prg, is used in both places. The important code in FUNCTION Demo1 is the line that builds the link to the PDF file:

```
Response.Write([<p>The ] + ;
   [<a href="/] + lcPDFFileLink + [">] + ;
   lcReportDescription + [</a>] + ;
   [ report has been created.</p>])
```

By inspecting the full code you will be able to see how the value of lcPDFFileLink was created. The demo is designed to place PDF files is a sub-directory of the virtual Web directory, which is seen as C:\Inetpub\wwwroot\PDFDemo\PDF\ to the Web server's file system and as /PDFDemo/PDF/ to the browser.

Demo 2 – Use HTTP Redirect to show the PDF file directly

This approach creates a PDF file just like the first demo, but we redirect the browser to the PDF file immediately without having to ask the user to click on a link. This has the advantage of a faster result for the user with no intermediate step involved. It should be noted that if the user requests a refresh of the Web page, the same PDF file will be accessed again because that's what

the URL points to. If the PDF file has been removed (refer back to Security Concerns for reasons why you may have wanted to remove it) then this request cannot be fulfilled.

The code for this demo can be found in FUNCTION Demo2 in PDFDemoProcess.prg. The important line is the one that constructs the redirection:

```
Response.Redirect( [http://] + Request.GetServerName() + [/] + ;
   THIS.cPDFServerRelativeURLOutput + JUSTFNAME( lcPDFFileName))
```

Again, this syntax is somewhat specific to Web Connection, but the concept is universal. The variable cPDFServerRelativeURLOutput points to /PDFDemo/PDF/.

Demo 3 – Send PDF back to the browser as a string

A third alternative is to use "streaming output" to send the PDF file back to the browser as a string. You can do this by setting the HTTP header content type to "application/pdf" and using FILETOSTR to create a string from the PDF file.

The code for this demo can be found in FUNCTION Demo3 in PDFDemoProcess.prg. The important lines are the ones that build the string from the file and return it to the browser.

```
LOCAL lcPDFString
lcPDFString = FILETOSTR( lcPDFFileName)
Response.Clear()
Response.ContentTypeHeader( "application/pdf")
Response.Write( lcPDFString)
```

The apparent result to the user is the same as that achieved in demo 2 above, namely that the PDF file shows up in the browser without the user having to click on a link to get it. However, there is one useful difference between streaming output and HTTP redirect. Using streaming output like this, the URL in the browser's address bar does not change after the PDF document is rendered. Therefore it still points to whatever process was called to request the PDF file in the first place, which in turn means that if the user requests a refresh of the page they're actually requesting that the PDF file be created again. While this may involve a performance penalty as compared to pointing to a PDF file that already exists like in demo 2, it has the advantage of allowing you as the developer to remove the PDF file immediately after sending it. As we discussed under security, this is something you normally want to do pretty quickly on a Web server.

Summary

The PDF format is a great way to deliver reports to your users, both from desktop and Web applications. Although there's a learning curve involved, it's not too steep and the results are well worth it. In this session you have seen how to do the following:

1. Choose the best PDF tool for your needs by analyzing and comparing issues such as licensing, costs, and technical considerations;

2. Install and configure the Amyuni PDF Converter on your own computer;

3. Deploy a PDF printer driver to the end-user's computer with your application;

4. Create and view PDF documents from within a desktop application in VFP;

5. Analyze the special requirements for using PDF documents in a Web application, including scalability, security, and exception handling; and

6. Deliver PDF content to the browser three different ways.

Good luck and happy programming!

Resources

Below is a list of PDF resources and products for developers that I have found useful in working with PDF integration in my own apps. In addition to the specific links below, be sure to search for PDF references on the Universal Thread (www.universalthread.com) and on the FoxPro Wiki (fox.wikis.com), where you will find a lot of discussion of this topic. On the Universal Thread you will also find downloads of other privately developed PDF tools. There's a lot of interesting work being done with PDF these days by a lot of bright people, and as with everything else, the Visual FoxPro community's willingness to share information is one of our greatest resources.

Lots of good PDF information is available from

www.adobe.com

www.amyuni.com (*a new support forum is now available at www.amyuni.ca/forum*)

www.activePDF.com

www.west-wind.com

www.PlanetPDF.com

www.PDFZone.com

Other 3rd Party PDF Products (in no particular order)

PDF File Creator – www.fytek.com

pdfFactory – www.fineprint.com

Win2PDF – www.daneprairie.com

DaVince Tools – www.davince.com

PDF-Xchange – www.docu-track.com

Page Genie Pro – www.xmlcities.com

Jaws PDF Creator – www.jawspdf.com

Other PDF References

www.pdflib.com/gsmanual

www.ghostscript.com

www.cs.wisc.edu/~ghost

Biography

Rick Borup is an independent developer working from Champaign-Urbana, Illinois. He is owner and president of Information Technology Associates, a professional software development, computer services, and information systems consulting firm he founded in 1993. Rick earned BS and MBA degrees at the University of Illinois, and spent several years doing mainframe software applications development and working in information systems management positions before turning to microcomputer database development tools in the late 1980's. He began working with FoxPro in 1991 and has worked full time in FoxPro and Visual FoxPro since 1993. Rick is a Microsoft Certified Solution Developer (MCSD) and a Microsoft Certified Professional (MCP) in Visual FoxPro.

: :