filePro Developer's Journal

Volume 2, number 2
June, 2001

ISSN 1527-9847

The definitive source of information for the filePro developer.

Submissions to filePro Developer's Journal

If you would like to submit articles that deal with topics suitable for filePro Developer's Journal, we would be most happy to consider them for publication. Be sure to follow the current author's guidelines, which are available on the web at:

http://www.hvcomputer.com/fpdj/authors.html

Guide to symbols.

Each article is marked with symbols to allow you to easily determine the targeted audience level and platform requirements.

Audience level is mark using symbols that should be self-explanatory:

Beginner Intermediate Advanced

If the article requires a version of filePro later than 4.5, it will be marked using: fP4.8 fP5.0

If the article requires a specific operating system platform, one of the following will appear.

SCO Unix
MS-Windows
Linux
Perl scripts
Shell scripts
Web/Internet

Publisher	Laura Brody
Editor	Laura Brody
Layout artist	Kenneth Brody
Circulation	Laura Brody
Assistant editor	Elizabeth Winkelmann

The filePro Developer's Journal is published quarterly by Hudson Valley Computer Associates, Inc. (914)739-5004. The Journal is distributed at the subscription rate of $75 US. Entire contents copyright Hudson Valley Computer Associates, Inc, 2001. All rights reserved. No reproduction in whole or in part may be made without express permission of the publisher. A subscription entitles you to print a single copy for your own use.

How to contact Hudson Valley Computer Associates, Inc.

Mail:	filePro Developer's Journal Hudson Valley Computer Associates, Inc. PO Box 859, 120 Sixth Street Verplanck, NY 10596-0859
Voice:	(914) 739-5004
Fax:	(914) 206-4184
E-mail:	fpdj@hvcomputer.com
Web:	http://www.hvcomputer.com

How to subscribe to the internet filePro mailing list

The internet filePro mailing list is a wonderful resource for filePro developers and end-users alike, and is available free of charge to anyone with an internet e-mail account. The mailing list is comprised of a group of experienced filePro developers, newcomers, and everything in between. They help each other to understand features, solve problems, suggest strategies, point you to other places to find information, or just discuss different aspects of filePro.

To subscribe, send the following e-mail:

```
To:  majordomo@seaslug.org
Subject:  subscribe
```

and in the body of the message:

```
subscribe filepro-list
end
```

You will receive a welcome message, usually within 24 hours, describing the mailing list, including how to post and how to unsubscribe. You should save a copy of this message for future reference.

Note: although this is not run by fP Technologies, it is monitored by its management and development staff.

How to contact the makers of filePro

Mail:	fP Technologies, Inc 5744 W. 79th Street Indianapolis, IN 46278
Web:	http://www.fptechnologies.com

	Voice	Fax	E-mail
Support	(317)802-0138	(317)802-9378	support@fileproplus.com
Sales	US: (800)847-4740 International: (212)644-9781	(212)644-9745	sales@fileproplus.com
TPTB*			filepro@fileproplus.com

* "The Powers That Be", aka "management".

Exposing filePro data through ADO and OLE DB

By Robert E. Haussmann, Ph.D.

This article appears as the first in a three-article series. Part 1 describes the framework for creating an OLE DB provider for filePro data, and identifies fPSQL as an invaluable tool that will be used in the process. The second article will demonstrate the creation of a Visual Basic OLE DB provider for custom filePro recordsets. This provider will harness the power of fPSQL to expose filePro data (DOS/Network or Native Windows versions) to ADO applications (such as Internet Information Server, Visual Basic, etc.). The final installment will expand the capability of the provider from read-only queries to full read-write access, allowing applications that can access ADO recordsets to retrieve, insert, and update records in filePro databases.

Introduction

A lot of people are waiting (more or less) patiently for fPTechnologies to ODBC-enable filePro. Client-side ODBC (the ability to access ODBC data sources from within filePro processing) has been targeted for a 2001Q2 release, and should provide an incredible amount of power and flexibility to the filePro programmer when interfacing with other software. Applications should include the ability from within filePro processing to perform a live ODBC query to a SQL database, pulling current inventory numbers into a filePro report being generated through rreport. Or similarly, using updating or inserting records in the same SQL database from within `INPUT` processing in `rclerk`.

But many developers need access to filePro data from applications outside of filePro. Server-side ODBC for filePro will address this need, but as of yet fPTechnologies has not committed to a release date. While workarounds exist (such as importing/exporting ASCII files), they tend to be cumbersome and as a general rule do not allow real-time access to filePro data. Luckily, given Microsoft's push for Universal Data Access (UDA), other options do exist.

Alphabet Soup: OLE DB, UDA, and ADO.

OLE DB is the basis for Microsoft's Universal Data Access strategy. It is intended to enhance (and ultimately replace?) the ODBC layer, extending access to non-relational data. In the world of OLE DB, there are two distinct classes: providers (servers) and consumers (clients). OLE DB-compliant applications operate in the consumer class (e.g., Internet Information Server, Visual Basic Scripts). OLE DB providers, on the other hand, are responsible for producing and providing recordsets to the consumers.

Sounds complicated — what does all this mean to the average filePro developer? Well, simply put, if we could build a filePro OLE DB provider, filePro data could be exposed to the rest of the world via UDA and ADO platforms. To give a real-world example, a developer using Microsoft's Internet Information Server (IIS) Active Server Page (ASP) platform could, in just a few lines of code, create a web page that performs a query against a set of filePro databases and returns real-time data to a web-based application. For example, the ASP code in figure 1 would produce a listing of all records and all fields in the filePro `codes` database.

Getting Started

So, where do we start? We need to write an OLE DB provider for filePro. Microsoft provides the skeleton for creating a generic OLE DB provider in Visual Studio (version 6.0). In order to use this generic provider in its most basic form, we need to be able to access our filePro data on demand from within Visual Studio. Preferably, we need to be able to start with an SQL query against filePro data (issued from an OLE DB consumer, such as IIS), perform this query, and place the results into an ADO recordset for return back to the consumer. The generic provider takes care of the transfer of data between our Visual Studio application and the OLE DB interface (see figure 2), but how do we go about the formidable task of writing an SQL-parser for filePro data, let alone implement such a parser while trying to take advantage of filePro indexes, maintaining record locking, etc.?

Discovering a Little-Known Gem

fPTechnologies sells a filePro add-on known as fPSQL. This program has been around for quite some time [ed. note: fPSQL has been around since filePro version 1.2, at least as far back as 1987], yet does not seem

```
<%
Dim objRS
 querystr = "select * from codes"
 set objRS = server.CreateObject("adodb.recordset")
 objRS.Open querystr,"provider=fpOLEdb;"
    response.write "<TR>"

    for each field in objrs.fields
        response.write "<TD>"
        Response.Write field.name
        response.write "</TD>"
      next
        response.write "</TR>"

 Do until objrs.eof
   response.write "<TR>"
    for each field in objrs.fields
        response.write "<TD>"
        Response.Write field.value
        response.write "</TD>"
        next
    response.write "</TR>"
   objrs.movenext
 loop
objrs.close
set objrs=nothing
%>
```

Figure 1 – Sample ASP code

- OLE DB Consumer (IIS) request (e.g., "`SELECT * FROM CODES`")
  - Generic OLE DB wrapper (Passes query to filePro OLE DB Provider)
    - filePro OLE DB Provider: interpret SQL query
    - filePro OLE DB Provider: execute SQL query
    - filePro OLE DB Provider: create recordset with results
  - Generic OLE DB Provider (Passes recordset back to consumer)
- OLE DB Consumer utilizes OLD DB recordset

Figure 2. filePro OLE DB provider model.

- OLE DB Consumer (IIS) request (e.g., "`SELECT * FROM CODES`")
  - Generic OLE DB wrapper (Passes query to filePro OLE DB Provider)
    - filePro OLE DB Provider: interpret SQL query
      - FPSQL — perform SQL query, return ASCII file
    - filePro OLE DB Provider: create recordset with results
  - Generic OLE DB Provider (Passes recordset back to consumer)
- OLE DB Consumer utilizes OLD DB recordset

Figure 3. Revised filePro OLE DB provider model

to have gained the notoriety/respect that it deserves. FPSQL allows users to perform SQL queries on filePro tables, using a subset of the American National Standards Institute (ANSI) X3.135-1986 SQL standard.

That's right — fPTechnologies sells a tool that will allow you to perform read-only SQL queries on your existing filePro data. FPSQL can be run interactively or from the command line, by specifying an input parameter file.

A Brief fPSQL Overview

In general, fPSQL uses standard SQL query syntax. For example, if we have a filePro database called `products`, that contains four fields (`code`, `description`, `quantity`, and `price`), the following query:

```
SELECT * FROM PRODUCTS
```

would produce output of the form:

```
Code    Description    Quantity  Price
WD-RED  Red Widget     324       19.95
WD-BLU  Blue Widget    128       19.95
WD-YEL  Yellow Widget  0         19.95
```

The asterisk denotes "all fields". Because we did not include any conditional selection clases, this command would return the contents of the entire table (all fields, all records).

We can constrain this output by specifying individual fields to return and utilizing a "`WHERE`" clause, in the form:

```
SELECT code, description
FROM PRODUCTS WHERE quantity>0
```

Resulting in:

```
Code        Description
WD-RED      Red Widget
WD-BLU      Blue Widget
```

The Real Power of fPSQL

By now, those of you familiar with SQL are asking yourselves, "But can I JOIN tables with fPSQL?" The answer is yes — and with the full benefit of existing filePro automatic indexes. For example, suppose that we have two filePro files: `cust` and `salesrep`. Assume that, in addition to other fields, each database contains the following:

```
Cust Fields
Name       (30,*)     Customer name
SalesRepID (5,.0)     Sales rep number
YTDPurch   (15,.2)    Year to date pur-
chases
Active     (1,yesno)  Is customer active?
```

```
Salesrep Fields
ID          (5,.0)     Sales rep number
Name        (30,*)     Sales rep name
```

(for the sake of this example, we assume that sales rep ID is a unique value).

With fPSQL, you can perform the following query:

```
Select cust.name, salesrep.name,
cust.YTDPurch
from cust, salesrep
where cust.salesrepID = salesrep.id and
cust.active='Y'
```

This statement will select all `customer` records with the "`active`" flag set to "`Y`". For each record, fPSQL will display the customer's name, YTD purchases, and sales rep name (where this name is pulled from the `salesrep` file):

```
cust.name  salesrep.name  cust.YTDPurch
A-1 Auto   Jane Smith     1000.00
AAA Auto   John Doe       500.00
ABC Auto   John Doe       100.00
```

In effect, fPSQL utilizes a SQL join to automatically perform a lookup to the salesrep database and retrieve the appropriate sales rep name.

Differences between fPSQL and ANSI SQL

(courtesy fPSQL Quick Reference Guide 4.8, Manual version March 30, 2000)

In building our OLE DB provider, it will be important to keep in mind there are a number of differences between ANSI SQL and fPSQL, including the following:

- SELECT DISTINCT clause is not implemented
- Password security is based on the creation password
- FPSQL is case-insensitive in sorts and comparisons (as is the rest of filePro)
- SET clause added
- filePro's system-maintained fields can be used
- filePro's additional field types can be used (e.g, MDY, HMS, etc)
- Associated fields can be used
- Fields can be referenced by field number (@1, @2, etc)
- filePro's MID function added

As mentioned previously, fPSQL can be used in command-line mode, by specifying a parameter file. In addition to other system-level parameters, the `SET OUTPUT` command can be used to redirect output to a file. With these tools in hand, we are ready to design the framework of our provider.

The filePro OLE DB Provider — Basic Design

Given that we have fPSQL available, our job in creating an OLE DB provider for filePro data just became much easier. We now have a tool that will allow us to take a standard SQL query and run it against filePro datasets. We do not have to worry about record locking or trying to use indexes efficiently, because fPSQL takes care of this for us. Assuming that our provider can utilize fPSQL in command-line mode, our provider need only perform general housekeeping tasks, pass a query to fPSQL, and then interpret the results (see figure 3).

In the next installment of this article, we will create a generic OLE DB provider using Visual Basic. This provider will rely on fPSQL to perform SQL queries against filePro data. These results will then be interpreted and formatted into an ADO recordset, which is then passed back to the calling consumer. Of course, to follow along you will need to purchase your own copy of fPSQL from fPTechnologies ☺.

Robert Haussmann can be reached at
haussma@nextdimension.net

Four filePro Favorites

Beginner, Intermediate, Advanced, & Extra Credit

Part 3 (Advanced)

By John Esak

I was casting around for ideas to write about and it suddenly came to me. Why not ask the users of my programs for some of the features they like. I also want everyone at various expertise levels to be able to read and get something out of any article I write, so after getting their ideas (and the list was huge, they love filePro!), I picked a simple app, an intermediate app, an advanced one, and something which any level user might find interesting. Let's just call it the Extra Credit idea.

#3 OPENDIR/NEXTDIR – Using Template Documents for Word Processing Merges

This is probably the favorite function of most of the *nix users I know. It allows them to be standing on a filePro record, press a key that shows them the most current group of template word processing and allows them to choose one. This causes Microsoft Word on the Windows box they are using to load the chosen document for them. They can then alter this document to taste, print it, fax it, etc. When they are done and close the document, they are brought back to the same place they were on the filePro record when they launched the function. This was a huge time saver for everyone, and all by itself, it is a nifty function. Later, the ability to take filePro data from the current record and automatically *merge* it onto the chosen template document was added. Now, this becomes a truly seamless integration between the high productivity tools of Windows like Microsoft Word, and the high productivity tool of *nix, filePro.

Prerequisites

There are several prerequisites that must be in place before this code piece can work for you. It is entirely written with DECLAREd variables, so you can immediately integrate it into any of your applications, but the prerequisites must still be fulfilled. First, you must be using a terminal emulator on the Windows box that can pass sequences to Windows. I am using FacetWin and show coding that will work with it. The same thing can be done with Anzio, Ice-10 and some other emulators. You will need to adjust your code to fit these packages.

NOTE: For the FacetWin implementation to work, you must have the (O)ption "Enable Run Program Escape Sequence" checked on the Properties page of the emulator.

The next requirement is that you set aside two directories on the *nix box, one for the template documents themselves, and one for the use of this processing. This is a "tmp" directory that will hold files that will be used one time only. They will generally be overwritten as time passes. Your users will create the template documents in the first directory, adding, changing and deleting as required. The program will dynamically show them only the current templates. It copies the one they choose to the specified "tmp" directory set aside for this programming so the original template does not get modified.

The final requirement is that this temporary directory must be "mapped" as a network drive on any Windows box that needs to use this program.

The directory names and the map letter of the network drive are tunables inside the processing table. You can customize them for your system once you see how the program works. I *strongly* recommend setting up the directories as shown in this processing first, creating some test Microsoft Word documents in the template directory. You will avoid many problems if you see how things work first before you move things around.

Description

Let me describe this application a little more fully. It assumes that there are a group of people working on a *nix system, say a Customer Service department all working on standard filePro data files. They are all accessing the *nix system from Windows95/98/2K/Me/NT boxes via an emulator that supports Escape Sequences (i.e., FacetWin, Anzio, etc.) The filePro files might be an Invoice file, a Customer file etc. This is unimportant. Prepared beforehand are template documents that are often sent to people in these filePro files. The program displays the list of templates so the user can point-and-pick the one they want and immediately it appears on the screen in Microsoft Word. At this point, they can print it, fax it, email it or whatever. When they are done and close the Word document, they are right back where they started on the filePro record. (As an added feature, shown after the main discussion below, data from the filePro record can be *merged* into the template document so there is relatively little for the operator to do other than choose the document, print it and close the document.)

Here is the only processing table needed. It can be inserted into any table as is assuming you set the tunables to match your system, or make your system match the tunables as they are set already (suggested). Please note, the reason I suggest you use the table as is and make the two required directories on your *nix box, and map to the same letter on your Windows machine, is that there is no *checking* done in this table for success on any of the file I/O calls. This is because in some instances (like this), it is not necessary to check the return values of things you *know* will work. Unfortunately, I only *know* they will work as they are set up here. I can pretty safely say that if you adjust the tunables to your site, and have the directories setup correctly, things will work just as they should. But remember, there is no checking, so usually, what you will see is "nothing" happening correctly, or sometimes an error from Word. Instead of adding checking on the return values of the I/O calls, you can place "`SHOW syscmd`" statements just before each `SYSTEM` statements to see if the command about to be executed *look* right.

(See figure 1.)

Most of this code is pretty self-explanatory for an advanced programmer, but I'm sure some of you who have less programming under your belt might want to follow along and learn how this table works, so here is an explanation of the more salient lines.

Line 4 - Build an array for the listbox that will hold the template documents. I have arbitrarily chosen "`99`" as the maximum number of templates that will be on hand at any one time. This may vary for you, but "`99`" is a nice high number, and any more than this would become unfeasible I think. The counter which is defined elsewhere to step through these documents is defined as `3,.0`, so you could raise this "`99`" up to "`150`" or even "`999`" if you like.

Line 9 - This line defines the handles to be used when doing file I/O. They are not given any edit type, because I do not check these handles to see if they contain valid return values or not. The form of the I/O commands requires them, so they are needed. They don't do much else but fulfill the syntax requirements though.

Line 13 - This is perhaps the most operative line of the table. It opens the template document directory and counts the number of files with a "`.doc`" extension.

Lines 14 and 15 - This is the loop bounds checking. These lines will keep the process getting filenames from the following `NXTDIR` function until there are no more files left to get.

Line 16 - This line is even more operative than the `OPENDIR` line, since `OPENDIR` is essentially useless without `NEXTDIR`. The `NEXTDIR` function gets each successive filename and stores it in the variable `LINE`. Actually, it stores much more than just the filename. It returns a formatted string that holds the essential file information. The string is 86 characters long, formatted as shown in figure 2.

```
v4.8.01 and higher NEXTDIR() return format is:

                Position   Length     Edit
   filename       1 - 32     32      left justified
   extension     34 - 43      10      left justified
   size          45 - 58      14      right justified, with commas
   date          60 - 69      10      MDYY/
   time          71 - 79       9      HMS, followed by "A" or "P"
   fullname      81 -112      32      left justified
   (all entries are separated by a space character)
```

Figure 2

```
File Name: test                                    Processing Table: input
   1 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: end
   2 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
@keyT   If: '@keyT
      Then:
   3 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'tunables   customize these to your system
      Then: '
   4 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'define a listbox to hold the files found in template directory
      Then: dim list_of_docs[99]
   5 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'define the directory where templates are stored for communal use
      Then: declare doc_dir;  doc_dir="/tmp/docs"
   6 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'define a tmp directory we need, must be mapped to map_ltr
      Then: declare global tmp_doc_dir;  tmp_doc_dir="/u/tmp/docs"
   7 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'define the mapltr used to mount tmp directory to the Windows box
      Then: declare map_ltr; map_ltr="T:"
   8 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: '
      Then: 'end_of_tunables
   9 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: declare handle, handle_c
  10 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'number of files returned by opendir
      Then: declare num_docs(3,.0)
  11 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: 'line returned by nxtdir,  filename portion of the line
      Then: declare line(112), filename
  12 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: declare counter(3,.0);  counter=""
  13 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: num_docs=opendir("*.doc",doc_dir)
  14 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
loop_ct If: counter lt num_docs
      Then: counter=counter+"1"
  15 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: not loop_ct
      Then: handle=closedir();  goto finlist
  16 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: line=nextdir();  filename=mid(line,"81","32")
  17 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: filename co "."  'strip off any extension if in sight
      Then: filename=mid(filename,"1",(instr(filename,".")-"1"))
  18 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: list_of_docs[counter]=filename
  19 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: goto loop_ct
  20 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
finlist If: 'finlist
     Then:
  21 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: cls("22")
  22 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: declare choice(2,.0), syscmd;  declare global the_doc
  23 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: 'put the listbox up on the screen
  24 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
choose  If:
     Then: choice=listbox(list_of_docs,,counter)
  25 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If: @sk eq "BRKY"
     Then: end
  26 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If: @sk eq "SAVE"
     Then: end
  27 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: the_doc=list_of_docs[choice]
  28 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: declare doc, path_doc_wq, path_doc_woq
  29 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If: 'put back the stipped off extension
     Then: doc=the_doc{".doc"
  30 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If: 'full path to document with quotes around it + _@id + .doc
     Then: path_doc_wq="\""{tmp_doc_dir{"/"{the_doc{"_"{@id{".doc\""
  31 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If: 'full path to document without quotes around it + _@id + .doc
     Then: path_doc_woq=tmp_doc_dir{"/"{the_doc{"_"{@id{".doc"
  32 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: syscmd="cp"<"\""{doc_dir{"/"{doc{"\"" < path_doc_wq
  33 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: system noredraw syscmd
  34 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: syscmd="chmod 777"<path_doc_wq
  35 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: system noredraw syscmd
  36 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: declare esc_seq
  37 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: esc_seq=chr("27"){"[2]c:\progra~1\micros~1\office\winword.exe \""{
           map_ltr{the_doc{"_"{@id{".doc\""{chr("10"){""
  38 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
       If:
     Then: handle_c=create("/tmp/catseq_"{@id,"wr0")
```

```
 39 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      If:
    Then: handle=write(handle_c,esc_seq);  handle=close(handle_c)
 40 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      If:
    Then: syscmd="cat /tmp/catseq_"{@id
 41 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      If:
    Then: system noredraw syscmd
 42 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      If:
    Then: handle=remove("/tmp/catseq_"{@id)
 43 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      If:
    Then: end
```

Figure 1

(Just a small plug. This information was taken from "Laura's Help Files" which came with the "Laura's Quick Reference Guide" I purchased from www.hvcomputer.com. An excellent set of documentation that every filePro programmer would find extremely useful.)

Once the `LINE` variable is filled with this info, `MID` is used to parse out the "fullname" of the file.

Line 17 - This line employs `INSTR` and `MID` to remove the "`.doc`" extension. There is no reason to show this extension in the listbox of available templates. Users usually never see this extension, and besides it takes up space on the screen.

Line 18 - The array "`list_of_docs`" (which will be used as the listbox) is filled field by field with each of these doctored filenames.

Line 19 - The procedure is sent back to the loop tests.

Line 24 - This is the first time the user sees what is happening. A listbox of all the available template word documents is put up on the screen.

Lines 25 and 26 - Allow the user to press either `BREAK` or `SAVE` and cancel the function.

Line 27 - Their choice of document is stored in the variable "`THE_DOC`". (This variable is `DECLARE`d `GLOBAL` so it can be used on a call table later.)

Note: Actually, from here on in, the biggest concern of the rest of this table is to use quotes correctly. Because long filenames now support SPACES, simple commands which used to work fine, do not work. Care must be taken to surround long filenames with quotes at the appropriate times. For example, you can no longer do something like this:

```
Then: filename="/tmp/document one.doc";
system "cp" < filename < "/tmp"
```

This would translate to:

```
        cp /tmp/document one.doc /tmp
```

The number of arguments would be wrong for the "`cp`" command. This is why you will see several copies of the filename in this processing table, some with quotes, without quotes, full path name, simple filename, etc. I wish it could be easier, but the spaces have to be dealt with properly.

Lines 28 through 31 - Defines the various versions of the filename as described above.

Lines 32 and 33 - These lines copy the chosen template document to the special tmp directory (with the addition of the ID of the person running the routine). This way, the templates stay unmarred, and the user can change this copied version to their needs.

Lines 34 and 35 - Changes the mode of the file to be 777 or read-writable-executable by anyone. This bypasses all permissions problems for filePro.

Lines 36 and 37 - Define the full Escape Sequence needed to open the Word document. Note that this is the path required on our machines. Your path to the "`winword.exe`" file may vary. If you like, you can just ensure that "`winword.exe`" is in the `PATH` of the Windows box. Then, using the full pathname would not be required, you could just use "`chr("27")[2]winword.exe …`".

Lines 38 and 39 - These are the filePro I/O commands needed to create a file and fill it with the Escape Sequence just built. It is important to note that the file must be `CLOSE`d after it is loaded with the sequence.

Lines 40 and 41 - These lines build the command that will cat the file just created and then sends the command via `SYSTEM` to the Windows box. (Where it will be picked up and executed by Windows.)

Line 42 - This removes the temporary "`catseq`" file. It is not really necessary to do this, as it will be recreated over and over as needed for each user.

You will be surprised at how nicely this process works once you get it put into place. The integration between *nix and Windows is so tight that the user just sees how much easier this is than finding and opening the template manually. This process as is will make users very happy.

Merging filePro data with the template documents

If you really want to blow their minds though, you can add one small processing table, and one line to the processing table above and actually merge data from the filePro record directly onto the template document. This makes sending out personalized letters of varying kinds a very productive task run entirely from your filePro database.

On the Microsoft Word side, the only thing that has to be done is the template documents must already be built as primary Merge documents. This is easily done, most secretaries know how to do this and can usually show you the procedure. It involves a 1,2,3 step process of picking the name of the datafile that will be merged with this primary document, putting the merge fields on the document, and choosing which fields from the data file merge with which fields on the Word document.

Note: Use a TAB delimited data merge format. That has always worked fine for me, and it is what the following processing table uses to create the merge data file.

IMPORTANT: It is very important that you name the data files for the merge operation with the following convention. Give them the same name as the document itself with a `.txt` instead of `.doc` extension. If your template is called "`capabilities.doc`", call the merge data file "`capabilities.txt`" and make sure that you are pointing to the the mapped special "tmp" directory. This is where the filePro process will put the merge data, and consequently that Word will find it there, too.

The only change needed to the existing processing table is to add a `CALL` to the "`mrg`" processing table. Do this on line 35 by adding it after the `SYSTEM` statement. Like this:

```
35 ------   -   -   -   -   -
      If:
    Then: system noredraw syscmd;
          call "mrg"
```

The "mrg" processing table is shown in figure 3.

There is only one final thing to do before any of this will work. You must create a special macro in Microsoft Word that will actually perform the merge of a document and a data file from the command line and return to the command line. This may take you some time (it took me the better part of a morning to make it work the first time), but it is worth the struggle. What you are going to do is start up the macro recorder and capture the keystrokes necessary to merge the document with the specified merge data file. These steps are simple, again most secretaries can show you how to do this. The key is to name the macro specifically with the same name that will be on the processing table. I used the name "`mergemacro`". More important than this is that you add the Close macro to the end of the macro you create. Otherwise, the Word document will not close automatically and return the user to the filePro screen from which they came.

Once you have recorded and saved this macro with the name "`mergemacro`", you now have to add this as a parameter to the Escape Sequence which brings up Microsoft Word and the designated document. Do this on Line 37 by inserting it just after the document name and before the `chr("10")` as shown in figure 4.

That's it. Now the users can add template documents at will, and everyone can use them whenever they need them. I have put this little process into many different situations and I'm sure you will find many places for it yourself.

```
File Name: test                                         Processing Table: input
   1 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: gosub doexp;  gosub puthdr
   2 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: gosub doexp;  gosub mrgdata
   3 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: END
   4 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
doexp   If:
      Then: declare exp_file
   5 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: declare extern the_doc, tmp_doc_dir
   6 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: exp_file=tmp_doc_dir{"/"{the_doc{".txt"
   7 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: export ascii merge=(exp_file) r=\r f=\t
   8 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: return
   9 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
mrgdata If: 'use an array to close up blank lines if necessary
      Then: dim array[5](30);  clear array
  10 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: declare ctr(1,.0);  ctr="1"
  11 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: Name ne ""
      Then: array[ctr]=Name;  ctr=ctr+"1"
  12 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: Company ne ""
      Then: array[ctr]=Company;  ctr=ctr+"1"
  13 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: Address_1 ne ""
      Then: array[ctr]=Address_1;  ctr=ctr+"1"
  14 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: Address_2 ne ""
      Then: array[ctr]=Address_2;  ctr=ctr+"1"
  15 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If: Address_3 ne ""
      Then: array[ctr]=Address_3
  16 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(1)=array["1"]
  17 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(2)=array["2"]
  18 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(3)=array["3"]
  19 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(4)=array["4"]
  20 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(5)=array["5"]
  21 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: return
  22 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
puthdr  If:
      Then: merge(1)=fieldname(-,"1")
  23 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(2)=fieldname(-,"2")
  24 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(3)=fieldname(-,"3")
  25 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(4)=fieldname(-,"4")
  26 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: merge(5)=fieldname(-,"5")
  27 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: return
```

Figure 3

```
37 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
        If:
      Then: esc_seq=chr("27"){"[2]c:\progra~1\micros~1\office\winword.exe \""{
              map_ltr{the_doc{"_"{@id{".doc /mergemacro\""{chr("10"){""
```

Figure 4

Four filePro Favorites

Beginner, Intermediate, Advanced, & Extra Credit
Part 4 (Extra Credit)
By John Esak

I was casting around for ideas to write about and it suddenly came to me. Why not ask the users of my programs for some of the features they like. I also want everyone at various expertise levels to be able to read and get something out of any article I write, so after getting their ideas (and the list was huge, they love filePro!), I picked a simple app, an intermediate app, an advanced one, and something which any level user might find interesting. Let's just call it the Extra Credit idea.

#4 HTML - Fancy Printing Made Easy

I know that Ken Brody has been doing the definitive HTML course and reference in this journal. It has been invaluable to me. I also know that there are many die-hard Unix users who still have not gotten their feet wet with HTML yet, and most of them feel they will never have a use for it. This small article is not meant to teach you HTML like Ken's articles, but rather to entice you to want to learn a little about what the HTML filePro functions may be able to do for you. This is an extremely simple application that lets you print some fancy fonts and tables without knowing much of anything about HTML. It assumes that you have the setup described in Section #3 of this article, i.e., you are connecting to your *nix box via an emulator like FacetWin (or Anzio) that lets you capture and execute Escape Sequences on the Windows side that are sent out from your Unix processing table. If this is the case, you can generate nifty looking HTML forms, bring them immediately to the screen for any particular record, print them, fax, them, email them, etc., and then close them out to be returned to the filePro record from which you came. It's much easier than trying to do the same thing with a filePro output format, believe me.

This is what the process below will generate. We can use the same "test" file that was used in Section #2 of this article. The first record's data is shown in figure 1.

The document the processing table will bring to the screen in your Windows browser is shown in figure 2. (In the example, Internet Explorer is used.)

This rendition is actually done in Microsoft Word and not HTML, but the HTML version is very similar. Be careful, when your users see things like this on their screen, available to be printed at the touch of a button, they are very likely to have you reformat all of their forms into HTML documents!

The processing table is shown in figure 3. It is really very much shorter than this, but I have put almost everything on a line by itself so the functions will be clearer as you follow them.

Lines 1 through 12 - These lines do essentially what the Section #3 processing table did to generate an Escape Sequence to the Windows box. However, the process is sent through the subroutine "`gethtm`" which generates the HTML document first. This document is given as to Internet Explorer as the file it should open.

Line 14 - This line creates the HTML document itself and names it with the value of found in field 1.

Line 15 - This starts the body of the document.

Line 16 - This line prints in Heading 1 format the specified text and pushes field 1 up against the text.

Line 17 - This prints a "horizontal rule", a line.

Line 18 - This inserts a paragraph start.

Line 19 - This generates a table with 4 columns, 2 rows, and cell padding (the space around the text in the cells) of "10".

Line 20 - This starts the first row of the table.

Lines 21 through 23 - This puts the text "Product Type" in the first cell of row 1.

Lines 25 through 32 - These lines insert their text into successive cells in row 1 as above.

Line 33 - This line closes off the first row.

Line 34 - This starts row 2.

Lines 35 through 46 - These lines insert filePro data from various fields into the successive cells of row 2.

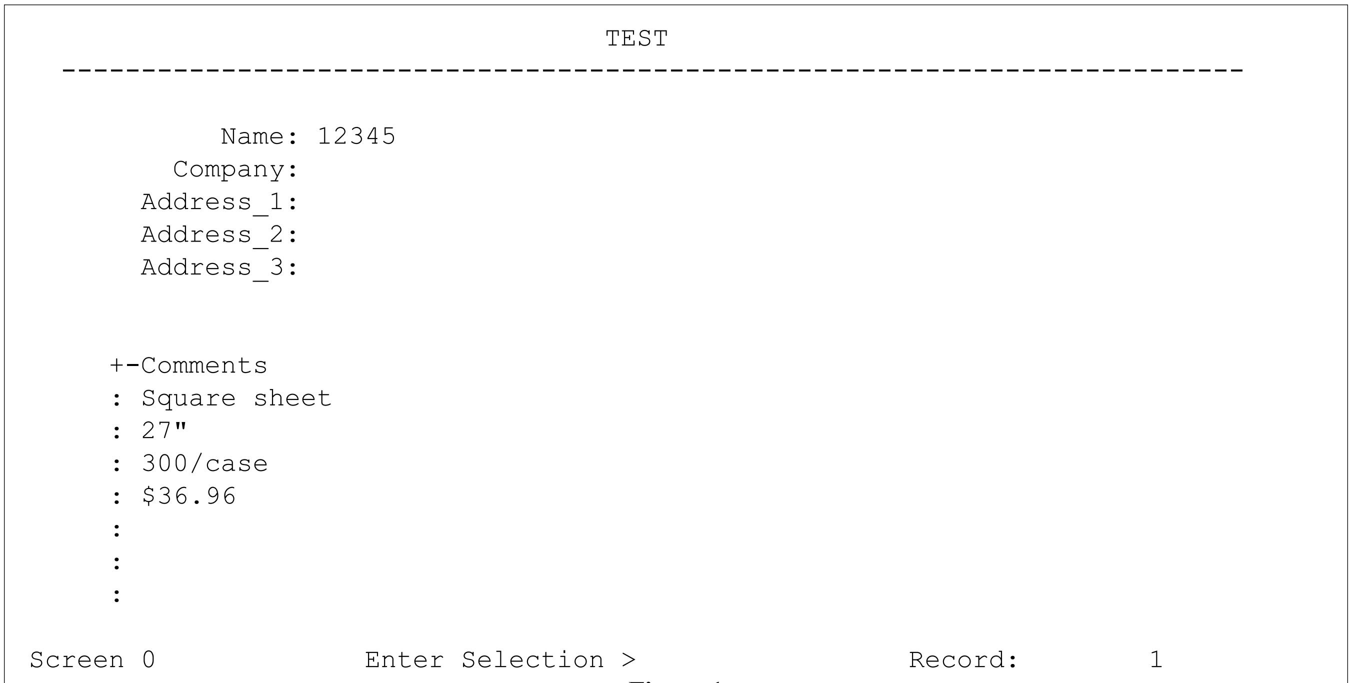
Line 47 - This closes off the second row.

Line 48 - This closes off the table.

Line 49 - This closes off the creation of the document.

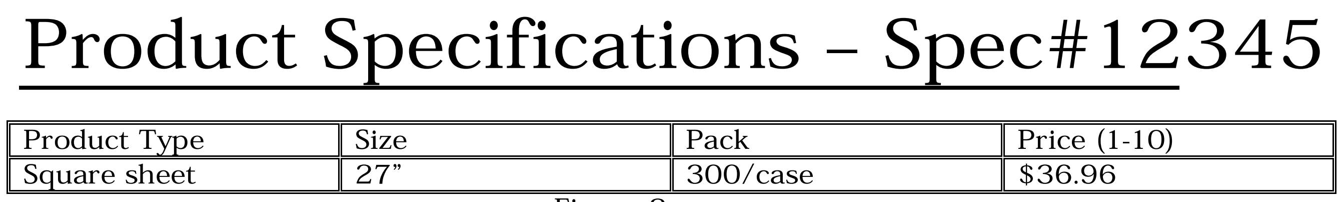
Line 50 - Returns back to the line which called this subroutine.

I hope some of these processes are useful to you, and I wish you good luck in your work with filePro.



```
                                   TEST
   --------------------------------------------------------------------------


           Name: 12345
        Company:
      Address_1:
      Address_2:
      Address_3:



    +-Comments
    : Square sheet
    : 27"
    : 300/case
    : $36.96
    :
    :
    :

Screen 0             Enter Selection >                  Record:        1
```

Figure 1



Product Specifications – Spec#12345

Product Type	Size	Pack	Price (1-10)
Square sheet	27”	300/case	$36.96

Figure 2

```
File Name: test                                      Processing Table: input
   1 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: end
   2 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
@keyT   If:
      Then: declare esc_cmd, handle, handle_c
   3 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: esc_cmd=""{chr("27"){"[2]c:/progra~1/intern~1/iexplore.exe t:/" { 1
              { ".htm"{chr("10"){""
   4 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: handle_c=create("/tmp/catseq."{@id,"wr0")
   5 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: handle=write(handle_c,esc_cmd)
   6 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: handle=close(handle_c)
   7 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: gosub gethtm
   8 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: declare runss3(100)
   9 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: runss3="chmod 777 /u/tmp/docs/" { 1 { ".htm"
  10 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: system noredraw runss3
  11 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: system "cat /tmp/catseq." { @id
  12 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: end
  13 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
gethtm  If: '*gethtm
      Then:
  14 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :CR "/u/tmp/docs/" { 1  { ".htm"
  15 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :BO
  16 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :H1 "Product Specifications - Spec#" { 1
  17 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :HR
  18 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :PA
  19 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TA :CO "4" :BO "2" :CP "10"
  20 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TR
  21 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TD
  22 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TX "Product Type"
  23 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TD-
  24 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TD
  25 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TX "Size"
  26 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :td-
  27 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
      Then: html :TD
```

```
 28 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX "Pack"
 29 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :td-
 30 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD
 31 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX "Price (1-10)"
 32 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :td-
 33 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TR-
 34 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TR
 35 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD
 36 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX 6
 37 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD-
 38 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD
 39 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX 7
 40 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :td-
 41 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD
 42 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX 8
 43 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :td-
 44 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TD
 45 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TX 9
 46 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :td-
 47 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TR-
 48 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :TA-
 49 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: html :CR-
 50 ------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    Then: return
```

Figure 3

Customized, Word-wrapped E-mail from filePro

By Kenneth Brody

Over the years, many methods of sending e-mail from a filePro application have been developed. I would guess that, in most cases, these are pre-written form letters, perhaps with some filePro-generated table in the middle. In this article, I will show how to generate form letters in which filePro data is included inline, while still keeping margins, justification, and word-wrap intact.

Note that this article depends on *nix-specific features, and will not work as-is on DOS/Windows systems.

A simple attempt

The simplest method normally used is to create a filePro form, containing everything laid out as you want, with filePro fields scattered throughout. This is generally sent to a file, and then the Unix `mail` command (or some other mail client) is called to send the file out via e-mail. A sample form is shown in figure 1a. It could be called with a few simple processing statements, such as:

```
PRINTER FILE "/tmp/email.dat"
FORM "MyEmail"
PRINTER RESET
SYSTEM "mail user@foo.com </tmp/
email.dat"
```

While this method is quick and easy, as you can see from the sample output in figure 1b, the gaps that are left around the data give it an "unfinished" look.

Using "`nroff`" for formatting

Unix includes a text formatter called "`nroff`" which we can use to format the output generated by filePro. (There are programs called "`troff`" and "`groff`" which are similar, but I will be using `nroff` throughout this article.) In its simplest form, you give it a text file where paragraphs are separated by blank lines, or started with an indent, and any single-spaced text in between is taken as a single paragraph. `nroff` will then word-wrap and justify the text within each paragraph.

See figures 2a and 2b for a simple example.

Using the same processing above, you can send your filePro output through `nroff` before e-mailing it, by changing the last line to:

```
SYSTEM "nroff /tmp/email.dat | mail
user@foo.com"
```

A modified version of the original form along with the output of this version of the command is shown in figures 1c and 1d. Note how the gaps around the filePro data have been removed, and the remaining text properly word-wrapped. Also note that the fields are no longer imbedded within each line, but rather each field ends the line. This is because multiple spaces within the text on a line (as would happen, as shown in figure 1b) would remain, and defeat the whole purpose of having run it through `nroff` in the first place. While this may look funny within the format, you can see that `nroff` has nicely re-wrapped the lines. I will discuss the "`.br`" later.

If that's all `nroff` could do, it wouldn't be much of a text formatter. In fact, it is a very powerful program that has been used to format entire books, and even the Unix "`man`" pages are usually formatted with `nroff`. While a complete tutorial on `nroff` is beyond the scope of this article (you could probably write an entire book on the subject), I will discuss some of the basics that are useful for filePro e-mail later on in this article.

Using "`sendmail`" rather than "`mail`"

The last piece of the puzzle is a program to handle sending the formatted e-mail. While you can use the `mail` command for this purpose, it was really designed as an MUA ("Mail User Agent"). That is, it provides a user interface for reading and sending mail. When you compose a message to be sent with `mail`, it generates the appropriate headers and hands it off to an MTA ("Mail Transport Agent") to do the actual transmission.

Given that there are limited options to `mail`, and the need to pass the subject, CC addresses, BCC addresses, and the recipient all on the command line, and the fact that it will simply format the headers and call an MTA, why not call the MTA directly? That's where `sendmail` comes into play.

Dealing with an MTA rather than an MUA requires that you include a properly-formed header, as well as the body of your message. While that may sound intimidating to someone unfamiliar with the internals of Internet e-mail, it's really quite simple. The

```
10        20        30        40        50        60        70        80
....:....|....:....|....:....|....:....|....:....|....:....|....:....|....:....|
..............................D.A.T.A...L.I.N.E.S.................................
Subject: Past due notice
To: *em

Dear *1     <3

It has come to our attention that your account is seriously past due.
Your balance of *5         is *6  days past due.  If we do not receive
payment within *7  days, we will be forced to take appropriate legal
action against you.

                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.

.............................E.N.D...O.F...F.O.R.M................................
```

Figure 1a

```
Subject: Past due notice
To: "John Doe" <root@linux2.hvcomputer.com>

Dear Mr. Doe

It has come to our attention that your account is seriously past due.
Your balance of    $100.00 is  90 days past due.  If we do not receive
payment within  10 days, we will be forced to take appropriate legal
action against you.

                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.
```

Figure 1b

```
Subject: Past due notice
To: “John Doe” <root@linux2.hvcomputer.com>

Dear Mr. Doe

It  has come to our attention that your account is seriously past
due.  Your balance of $100.00 is 120 days past due.  If we do not
receive  payment within 10 days, we will be forced to take appro-
priate legal action against you.


                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.
```

Figure 1d

```
         10        20        30        40        50        60        70        80
....:....|....:....|....:....|....:....|....:....|....:....|....:....|....:....|
..............................D.A.T.A...L.I.N.E.S...............................
Subject: Past due notice
.br
To: *em

Dear *1     <3

It has come to our attention that your account is seriously past due.
Your balance of <5
is <6
days past due.  If we do not receive
payment within <7
days, we will be forced to take appropriate legal action against you.


                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.

.............................E.N.D...O.F...F.O.R.M..............................
```
Figure 1c

header is simply a series of lines consisting of a keyword, colon, space, and additional information. For example, "`From: <laura@hvcomputer.com>`" or "`Subject: Call for articles`". Following the header lines is a blank line, and then the body of the message.

If you've ever looked at all of the header lines that your MUA normally hides from you, you might be wondering how you generate all of that information to send your e-mail. The answer is "you don't". In fact, the only required header line that `sendmail` needs passed to it is something to tell it the recipient. This can be any of "`To`", "`CC`", or "`BCC`". The following is a perfectly legal e-mail to send via `sendmail`:

```
      Cc: <root@localhost>

      This is a simple e-mail.
```
For our purposes we will add one more header line, "`Subject`", to make the e-mail more informative. Other header items, such as `Date` and `Message-ID` will be generated by `sendmail`.

Once you have built your e-mail with headers in place, you send it to the command "`sendmail -t`" to be handled. Using the same example we started with, this could be done by adding the "`To`" and "`Subject`" lines at the top of your form, followed by a blank line, followed by the body of your message, and then changing the `SYSTEM` command once again, to:

```
SYSTEM "cat /tmp/email.dat | sendmail -t"
```
You should note that I have used "`cat`" here rather than "`nroff`". Remember how `nroff` reformatted the text in figures 2a and 2b? Well, unless we do something special, it will also justify and word-wrap the "`To`" and "`Subject`" lines as if they were one paragraph. (And, if we put a blank line between them to force a paragraph break, there will be a blank line in the output, terminating the header section.)

Some rudimentary `nroff` formatting

Formatting in `nroff` documents is done with commands imbedded within the document itself. Commands are on lines that start with a period, and are always on lines by themselves. (As opposed to HTML, for example, where formatting is done within the text itself.)

In true Unix tradition, commands are very terse. For example, to set the length of a page to 11 inches, the command "`.pl 11i`" would be used. To set flush-left text, the command is "`.ad l`". Some conditional code (taken from the "`tmac.doc`" macro library – and no, I don't know what it does) could

```
Four score and seven years ago our fathers brought forth, upon
this continent, a new nation, conceived in liberty, and dedicated
to the proposition that “all men are created equal”

Now we are engaged in a great civil war, testing whether that
nation, or any nation so conceived, and so dedicated, can long
endure. We are met on a great battle field of that war. We come
to dedicate a portion of it, as a final resting place for those
who died here, that the nation might live. This we may, in all
propriety do. But, in a larger sense, we can not dedicate -- we
can not consecrate -- we can not hallow, this ground -- The
brave men, living and dead, who struggled here, have hallowed
it, far above our poor power to add or detract. The world will
little note, nor long remember what we say here; while it can
never forget what they did here.

It is rather for us, the living, we here be dedicated to the
great task remaining before us -- that, from these honored dead
we take increased devotion to that cause for which they here,
gave the last full measure of devotion -- that we here highly
resolve these dead shall not have died in vain; that the nation,
shall have a new birth of freedom, and that government of the
people by the people for the people, shall not perish from the
earth.
```

Figure 2a

```
Four  score  and  seven years ago our fathers brought forth, upon
this continent, a new nation, conceived in liberty, and dedicated
to the proposition that “all men are created equal”

Now we are engaged in a great civil war, testing whether that na-
tion, or any nation so conceived, and so dedicated, can long  en-
dure.  We are met on a great battle field of that war. We come to
dedicate a portion of it, as a final resting place for those  who
died  here,  that the nation might live. This we may, in all pro-
priety do. But, in a larger sense, we can not dedicate -- we  can
not  consecrate  --  we  can not hallow, this ground -- The brave
men, living and dead, who struggled here, have hallowed  it,  far
above  our  poor  power  to add or detract. The world will little
note, nor long remember what we say here; while it can never for-
get what they did here.

It  is  rather  for  us,  the living, we here be dedicated to the
great task remaining before us -- that, from these  honored  dead
we  take  increased  devotion  to that cause for which they here,
gave the last full measure of devotion -- that we here highly re-
solve  these  dead  shall not have died in vain; that the nation,
shall have a new birth of freedom, and  that  government  of  the
people  by  the  people for the people, shall not perish from the
earth.
```

Figure 2b

look like this:

```
.ie "\\$1"|" \{\
.   if "\\*(mN"Op" .ds A\\n(aC \fR\\$1\fP
.   if "\\*(mN"Ar" .ds A\\n(aC \fR\\$1\fP
.   if "\\*(mN"Fl" .ds A\\n(aC \fR\\$1\fP
.   if "\\*(mN"Cm" .ds A\\n(aC \fR\\$1\fP
.   if "\\*(mN"It" .ds A\\n(aC \fR\\$1\fP
.\}
.el .ds A\\n(aC \\$1
```

As I said, you could write an entire book on `nroff`.

For our purposes, however, we are only interested in a very limited subset of all the `nroff` commands available:

- `.nf` Disable fill
- `.fi` Enable fill
- `.ad` Adjust text (l=left, r=right, c=center, n=justify)
- `.pl` Page length

Basically, "`.nf`" and "`.fi`" disable and enable `nroff`'s functionality of combining and re-wrapping lines within a paragraph, so that line breaks are kept within the disabled sections. (Similar to HTML's `<PRE>` and `</PRE>` tags.) When filling is enabled, the "`.ad`" command tells `nroff` how to fill the lines of text, whether that be flush-left, flush-right, centered, or justified. Finally, "`.pl`" controls the length of the page being output. (Blank lines will be added if necessary to fill the page.)

Putting everything together

The final version of our form is in figures 3a and 3b. (The main paragraph is repeated 3 times, to demonstrate different adjust values.)

First, the "`.pl 1`" command sets the page length to 1 line. This effectively turns off `nroff`'s adding blank lines to fill the page. (Otherwise, you would always end up with a multiple of 66 lines in your e-mail, even if the form itself is shorter. While this is useful when sending to a printer, it is unnecessary for e-mail.)

Next, the "`.nf`" command turns off fill. This keeps each of the header lines on separate lines in the output. After the header lines, "`.fi`" re-enables fill. This could have been done by adding an explicit line break ("`.br`") between each header line, but this method makes for a cleaner format, especially if you want to include additional header lines here.

The paragraph that makes up the body of the message has been repeated three times in this example, in order to show the differences between flush-left ("`.ad l`"), justified ("`.ad n`"), and no-fill ("`.nf`") formatting options.

Finally, we include the signature at the bottom of the message. Note that these will appear on their own lines, as `nroff` takes their indentation as a sign that a new paragraph has started, without leaving a blank line between them. (This could not have been used to place the header entries on separate lines, as the header entries must start in column 1, unless they are a continuation of the previous line.)

Eliminate temp files and `SYSTEM` commands

Finally, I should note that if the intention is to have filePro generate a file which is simply going to be sent through a command pipeline for processing, you can eliminate the temporary file and `SYSTEM` command by specifying the command pipeline as the printer destination:

```
PRINTER "| nroff | sendmail -t"
FORM "MyEmail"
PRINTER RESET
```

To take this one step further, and to eliminate variations in different environments, the command pipeline can be defined in `pmaint`, rather than duplicated everywhere you need it. For example, on RedHat 6.0, `sendmail` is in the `/usr/sbin` directory, which is not normally part of the `PATH`, meaning that you would have to explicitly state "`/usr/sbin/sendmail`" in every reference. If you were to move to a different system where `sendmail` is in a different directory, you would have to change every reference. By defining it within `pmaint`, there is but a single reference that needs to be changed. I have defined a printer named "`email`", type "`nocodes`", with "`| nroff | /usr/sbin/sendmail -t`" as the destination. The processing then consists of:

```
PRINTER NAME "email"
FORM "MyEmail"
PRINTER RESET
```

Generating printed form letters

Using these same techniques, there is no reason why you need to send this information through `sendmail` and generate e-mail out of your forms. By simply eliminating the header lines, and sending `nroff`'s output to the printer rather than `sendmail`, you can just as easily generate printed form letters that are formatted just as well.

```
10        20        30        40        50        60        70        80
....:....|....:....|....:....|....:....|....:....|....:....|....:....|....:....|
..............................D.A.T.A...L.I.N.E.S...............................
.pl 1
.nf
Subject: Past due notice
To: *em
.fi

Dear *1     <3

.ad l
It has come to our attention that your account is seriously past due.
Your balance of <5
is <6
days past due.  If we do not receive payment within <7
days, we will be forced to take appropriate legal action against you.

.ad n
It has come to our attention that your account is seriously past due.
Your balance of <5
is <6
days past due.  If we do not receive payment within <7
days, we will be forced to take appropriate legal action against you.

.nf
It has come to our attention that your account is seriously past due.
Your balance of <5
is <6
days past due.  If we do not receive payment within <7
days, we will be forced to take appropriate legal action against you.
.fi

                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.

.............................E.N.D...O.F...F.O.R.M..............................
```

Figure 3a

In fact, there is no need to create two versions of the form. Instead, place the e-mail header fields (and the blank line that follows) in one form, and the body of the form in another. You can then use the `FORMM` command to combine the two (or more) for e-mail, or just use the body when printing:

```
PRINTER NAME "email"
FORMM "EmailHeaders"
FORMM "FormLetter"
FORM "MyDotSig"
PRINTER RESET

PRINTER NAME "laserjet"
FORM "FormLetter"
PRINTER RESET
```

Acknowledgments

Although I have used `nroff` on and off over the years, I am a bit rusty, and I found the `man` pages somewhat lacking while writing this article. I found <http://www.cd.pdx.edu/~trent/gnu/groff/groff_toc.html> to be a good refresher on the `groff` formatting commands.

```
Date: Sun, 31 Mar 2002 15:12:14 -0500
From: root <root@linux2.hvcomputer.com>
Message-Id: <200203312012.PAA00850@linux2.hvcomputer.com>
Subject: Past due notice
To: "John Doe" <root@linux2.hvcomputer.com>

Dear Mr. Doe

It has come to our attention that your account is seriously past
due.  Your balance of $100.00 is 120 days past due.  If we do not
receive payment within 10 days, we will be forced to take appro-
priate legal action against you.

It has come to our attention that your account is seriously  past
due.  Your balance of $100.00 is 120 days past due.  If we do not
receive payment within 10 days, we will be forced to take  appro-
priate legal action against you.

It has come to our attention that your account is seriously past due.
Your balance of $100.00
is 120
days past due.  If we do not receive payment within 10
days, we will be forced to take appropriate legal action against you.

                             Respectfully yours,

                             Harry Small
                             President
                             Small Computer Enterprises, Inc.
```

Figure 3b

Sorting Multi-Column Reports by Column Rather Than Row

By Jim Asman

Overview

A short while ago I needed to produce a simple report with an alphabetical sort on a name field. This type of output is a basic function for filePro; however, in this case the report really needed to be in a multi-column format with the sort running from top to bottom down each column across the page. This could be an enrollment listing, a voter list, or what have you. If the printed fields are collectively rather narrow, we certainly would want to print more than a single column across the page.

The Problem

While filePro generally makes the formatting of a simple report almost trivial, a multi-column report can be far more difficult. You could use a label format to generate the layout, but that would preclude a heading if desired, and further the sort of a label format runs across the rows rather than down the columns.

A workable solution might entail setting up the output as a single page form rather than a report and place all of the heading fields as well as all of the data fields on the format directly. If you had, let's say, four columns each with 55 lines, you are looking at a minimum of 220 data fields to maintain, plus whatever fields that may exist in the heading. It becomes a particularly tedious process if you need to move the data fields around a bit as well. In processing, the fields could then be populated in the desired order. There are undoubtedly other ways to approach this format, but let me present one where the PCL language makes it easy.

PCL to the Rescue

With the assistance of two different PCL print-codes, we can use a "vanilla" filePro report format and print as many columns as we can fit across the page. This technique can also be used on a label format if you prefer the sort to run down the columns rather than across the rows. In addition, PCL allows very precise positioning of the text on the label itself, and that can be very important of you are crowding the text onto the label. That is a likely scenario on "three across" labels which aren't very wide to begin with.

PCL5 vs. PCL3

There are two different PCL commands we can use to set up additional columns in the report. The "sheet offset" command will only work on PCL5 printers. This includes all standard LaserJets and a few DeskJet models, the 1200, 1600, and the 2250. Other PCL3 DeskJet printers simply do not support the offset command. There is salvation, though, for DeskJet users. Although not as flexible, the "left margin" command is supported by both PCL3 and PCL5 printers.

The basic technique is to define the report as a single column as you would normally, but long enough to contain all the records for an entire page. Then through processing, PCL codes are injected into the data stream that redefine both the left margin position and the current row position.

PCL Sheet Offset Command (`^[&l###U`, `^[&l###Z`)

The sheet offset command is a mechanism to provide an extra margin on print jobs that may require additional space on one side or the other, perhaps for binding or punching etc. The beauty of this command is that you can design the layout so the text is normally centered, and in the case where you require a binding edge, you only need to add the offset code once for the entire job. Now, consider a duplex (printed on both sides) print job that was going to eventually be put into a report folder that requires an extra half inch for binding. In a duplex job, the offset needs to be applied to the left side of the front and to the right hand side on the back of the sheet. This is precisely what the "sheet offset" command does, automatically. I bring up duplexing as it has implications if we duplex our multi-column report.

The syntax for the "sheet offset" command is `^[&l###U` where `###` represents the width of the desired offset, stated in decipoints (1/720's of an inch). Note that the command ends with "U" and in this case the "U" specifies the long edge of the page. The same command but ending in "Z" refers to the short edge of the page. Offset values represent absolute positions across or down the page independent of resolution, pitch, or user units. The default value for either command is "0".

In portrait mode, positive "U" values will shift the whole PCL page to the right, and likewise positive "Z" values will shift the PCL page down. Negative values move the print to the left and up the page respectively. `^[&l360U` will establish an extra half-inch margin on the left. The "U" and the "Z" refer to the same physical edge of the page regardless of the orientation, so if you are printing in landscape mode, the "Z" command will move the print left and right and the "U" command up and down.

If you change to landscape mode, the previous top of the sheet becomes the right hand edge. So to get a left offset in landscape mode you need to supply a negative value to the "Z" command. For example `^[&l-1440Z` would move the left margin two inches to the right of the default position on a landscape page. When any offset is applied, the entire PCL coordinate system is moved the specified amount; i.e., position 0x0y is relocated and any PCL positioning codes are referenced from the new offset location.

Row Number Command (`^[&a###R`)

The row number command directs the CAP to the row specified by the value in the command, but does not move the cursor from its current position along the row. As PCL numbers the rows beginning with "0" the argument required will be one less than the desired filePro row number. So to position the CAP on the sixth line on the output, the command `^[&a5R` would do the job. If you precede the argument with a "+" or "-" sign, then the command is interpreted as a position relative to the current row. So `^[&a-5R` will run the cursor back up the page 5 rows. The physical location where the CAP is located after the command is executed is totally dependent on the current line spacing. Row four is located at a different vertical position when printing at 6 lpi vs. 8 lpi. This is good because our filePro output formats are generally line oriented. This is the last of the commands necessary to produce the multi-column output. Now let's look at the details of making this work.

Defining the Output Format

Assume that we want to define a report format that has a four line heading plus 56 data lines. That brings us to a normal LaserJet default of sixty print lines per page. Further, our data fields permit three columns across the page allowing 168 records per page (56 * 3). The page length defined on the filePro output format is (records per page + no. of heading lines), which in this case is 172 (168 + 4). From filePro's standpoint we have a four-line heading followed by 168 lines of data. Define both lines per page and lines to print per page as 172. From the printer's standpoint we have but 60 lines per page.

In processing the task is to relocate the CAP back to the first data line after we print a full column of data. At that same time we want to change the offset to position new print data on the next column to the right. Once the last column has been filled, we let the printer do its terminating linefeed and eject the page. Figure 1 shows how the 172 lines are output by filePro with the attendant printcodes. Figure 2 then shows how the 172 lines are interpreted by the

```
Heading 1
Heading 2
Heading 3
Heading 4
Data   1^[&l0L
Data   2
Data   3
....
....
Data  56^[&l1920U^[&a-56R
Data  57
Data  58
....
....
Data 112^[&l3840U^[&a-56R
Data 113
Data 114
....
....
Data 168^[&l0U^[&l1L
```

Figure 1

```
Heading 1
Heading 2
Heading 3
Heading 4
Data   1        Data  56        Data 113
Data   2        Data  57        Data 114
Data   3        Data  58        Data 115
........        ........        ........
........        ........        ........
Data  56        Data 112        Data 168
```

Figure 2

printer and are assembled onto the 60 print lines on the page.

Define the heading section as you would on any report format. The data format line would be something like the following.

```
*1                *2             *pc
```

Field `1` contains “Last Name” and field `2` contains “First Name”. The final field “`pc`” contains PCL code, but it is only loaded when the 56th, 112th, and the 168th record are being processed. The field is defined as non-global so its value disappears between records. pc will contain the complete PCL code to define the new offset and “goto first data line” printcodes.

Page Offsets

The portrait page has printable width of eight inches. That translates to 5760 decipoints (8*720). So if we divide the page into three equal columns, our offsets would be at 0, 1920, and 3840 for each of our three columns respectively. The first data line will always be row five regardless of the column.

Embedding PCL codes into filePro variables always involves the ESC character, `chr("27")`. On any processing table that uses PCL, I always define the global variable `ec=chr("27")`, which is then used to further define other codes. We can define a page offset code for each of our three columns.

Define the following as global, but keep pc as non global.

```
   1 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: pf="y"
       Then: goto start
   2 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If:
       Then: ec=chr("27"); rn="0"; pf="y";
   3 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: 'code for PCL5 printers
       Then: 'oa=ec{"&l0U"; ob=ec{"&l1920U"; oc=ec{"&l3840U"; od=ec{"&a-56R"
   4 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: 'code for PCL3 printers at 10 pitch
       Then: oa=ec{"&a0L"; ob=ec{"&a27L"; oc=ec{"&a54L"; od=ec{"&a-56R"
   5 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
start    If:
       Then: rn=rn+"1"
   6 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: rn="1"   ' disable perf skip
       Then: pc=ec{"&l0L"
   7 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: rn eq "56"
       Then: pc=ob{od;
   8 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: rn eq "112"
       Then: pc=oc{od;
   9 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If: rn="168" ' ... enable perf skip
       Then: pc=oa{ec{"&l1L"; rn="0"
  10 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If:
       Then: end
  11 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If:
       Then: pf(1,*,g); oa(8,*,g); ob(8,*,g); oc(8,*,g); od(7,*,g); pc(25,*);
  12 -------   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
         If:
       Then: ec(1,*,g); rn(3,.0,g);
```

Figure 3

```
        oa=ec{"&l0U"
        ob=ec{"&l1920U"
        oc=ec{"&l3840U"
```

Once a column has been filled, the cursor has to be returned back to the first data line. All we have to do is have the printer back up 56 lines.

```
        od=ec{"&a-56R"
```

A relative positioning command is used to crawl back up the page rather than an absolute row number, just in case the heading had been defined with different line spacing. When executed from the last data line, the cursor will be returned to the first data line waiting to begin a new column.

Pagination

There is still a problem remaining albeit quite obscure at first glance. Consider what happens on the last page of the report. Remember filePro considers each page to have 172 lines. In the case where it works out that just a few records end up on the final page, filePro will pad the end of the page with enough linefeeds to fill the 172 lines.

If enough lines are added to the end of the output, the printer will eject one or more blank pages after the last page. In the worst case, there would only be one record on the last page and filePro would emit 167 linefeeds. HP printers won't eject a partially filled blank page, but they will do so a page that is completely "filled" with blank lines. As this is beyond the control of the processing table, we have to look to the printer to deal with it. There is a PCL command to fix most problems, and this is no exception.

Perf Skip Command `(^[&l#L)`

The perf skip command is the PCL mechanism, that when enabled, causes the printer to eject the page when a linefeed moves the cursor past the bottom margin. When disabled, the printer will only eject the page upon receipt of a formfeed or printer reset. Legal values to the command are "0" to disable and "1" to enable the command. The default is "1" (enabled). What we can do is disable the perf skip on the first record of the page and enable it only on the last record. It must be enabled on the last record of the page so "filled" pages are ejected properly. Because the perf is disabled in all but the last record on the page, any trailing linefeeds on a partially "filled" last page won't push a sheet out of the printer. The final printer reset from the printcode table will take care of getting the last page out of the printer.

Running the Operation

Refer to Figure 3 that shows a complete processing table. Once the records are being processed, we need to keep track of which record on the current page is being processed. Assume the record count for the current page is held in the global variable `rn`. `rn` starts as "`0`" and is incremented as each record is processed.

Note that when the counter reaches the 168th record, `od` is omitted from `pc`. This allows the final linefeed to cause the printer to eject the page. Additionally, `rc` was reset to "`0`" so the process starts over on the next page and the new page offset is also set to zero "`0`" for the next page.

Duplex Printing

Although it is not addressed in the processing table as shown, the methodology falls apart when printing on the back side of the sheet because in a duplex job as the offset is applied in the reverse direction. That's great for binding but is totally wrong for this application. You can tidy this up a bit, but for the sake of discussion lets assume the variable oe contains the odd/even page status, where "0" represents odd and "1" represents even. Start with `oe="0"`, and then when processing the 168th record for the page, toggle the value of oe between "0" and "1". Define `oa`, `ob`, `oc` with negative values if `oe="1"` even or with positive values if `oe="0"`. There are no duplex issues when using the "left margin" command to position the columns.

Left Margin Command `(^[&a###L)`

The PCL3 DeskJet printers do NOT understand the "sheet offset" command period. They do, however, understand the "left margin" command. These two commands are different. Not so much in what they do on the surface, but more in the ramifications of each of them. While the "sheet offset" command physically moves the PCL coordinate system, the "left margin" command only changes the point that a CR returns to, if you get the meaning. ☺ So after a "sheet offset" command, the location 1400X is physically different from where it had been previously as the coordinate system was shifted. The "left margin" command is defined only in terms of a character position, so its actual physical position is dependent on the pitch of the current font. The location 1400X is unaffected by the "left margin" command. The command is in the following form…

```
        ^[&a###L
```

where `###` is the character position along the line where the new left margin is located. If you are using a fixed pitch font and are not relying on any PCL positioning codes for alignment, then the left margin code will probably be quite satisfactory. Because the argument to the "left margin" command is dependent on the current pitch, there is no universal command set for a two, three, or four column format. That will be dependent on the current font pitch. In any case, let's set up the variable for a 10 pitch scenario for our current three column format.

```
        oa=ec{"&a0L"
        ob=ec{"&a27L"
        oc=ec{"&a54L"
```

or if you wanted 12 pitch then

```
        oa=ec{"&a0L"
        ob=ec{"&a32L"
        oc=ec{"&a64L"
```

When defined as a "left Margin" command, the same filePro variables should work equally as well on the format for the LaserJet or the DeskJet.

The processing table contains the setup for both PCL5 and PCL3 printers. It is set for PCL3 at the moment. To use the PCL5 code, comment out line 4 and then uncomment line 3. When printing a duplex job, the PCL3 setup would preclude you from having to worry about the odd/even page considerations.

Try It Out

Included in the files is "`prc.column`", which is the runable "prc" table shown in Figure 3. Use any database you have that has enough records with a suitable field for sorting. Define an output format with four heading lines and one data line. Define lines per page and lines to print, both as 172. You can populate the heading with whatever pleases you. On the data line, just put `*yourfield` on the leftmost position, and then at an appropriate distance to the right to, far enough to accommodate the length of `*yourfield` put the field `*pc`. Something like…

```
*4                          *pc
```

where `*4` here represents `*yourfield`. Rename the output processing table to suit you situation, and then run the output with a sort on `*yourfield` and a selection set that will cause at least 200 records to be selected. Hopefully, that will produce a couple of pages with three nicely spaced columns.

Conclusion

The material presented here is quite portable and can be easily modified to suit other layouts. This one was configured to fit a default LaserJet 60 line page. If you require a different page size or orientation change, you will have to initialize the printer with the appropriate page format code for the task. Fundamental to being successful is synchronizing the printer format to the modified filePro output.

If you are unsure about formatting a LaserJet page, review the article I wrote in the first issue of this publication.

Did you have to borrow a friend's copy? Why not get your own? (It's easy!)

filePro Developer's Journal

The definitive source of information for the filePro developer.

Includes all the things you would expect to find in a quality technical journal...

- Hints, tips, and tricks.
- Articles by experienced filePro developers, including fPTechnologies' development team.
- Un(der)documented features.
- New features of the latest filePro release.
- Pointers to other sources of filePro-related information.

... and so much more!

- Cut-and-paste code samples.
- All code samples included. No need to type them in, or purchase them separately.
- Columns devoted to filePro issues specific to: Windows 95/98/2000/NT, Linux, and SCO Unix.
- Interviews with prominent filePro personalities.
- Reviews of filePro-related products and add-ons.
- Delivered electronically via the Internet, so it can't be lost, delayed, or mauled by the post office. (And no extra charges for international delivery.)

Published quarterly, distributed electronically, 50+ pages per issue.
$75 (US currency) annually. Back issues are $25 (US currency) each.
Contact us for information on discounts for multiple subscriptions to the same company.

(Please print clearly)

Name: ______________________________ Phone: ____________________

Company: ______________________________ Fax: ____________________

Address: ______________________________ Country: ____________________

City/State/Zip: ____

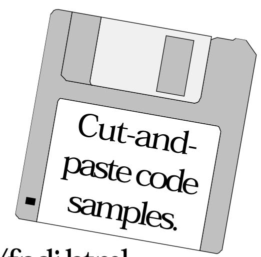
E-mail: ____

[ ] 1 year subscription ($75 US) [ ] Back issues ($25 US each). Which issue(s)? ________________

We accept credit cards through PayPal. Please specify "paypal@hvcomputer.com" as your referral address. Sign up at http://www.paypal.com

If paying via PayPal, you can fax this completed form to (914)206-4184
or
Mail this form with a check (payable in US currency, to "Hudson Valley Computer Associates, Inc."), or credit card information and signature to:

Hudson Valley Computer Associates, Inc.
PO Box 859, 120 Sixth Street
Verplanck, NY 10596-0859



Want more information?
Download our free 14-page sample issue at http://www.hvcomputer.com/fpdj.html