

[an error occurred while processing this directive]

A Guide to FoxPro - Windows Printing Behavior

All results are the same in both FPW versions and Visual FoxPro, except where indicated.

● Page Contents	
● "PDSETUP being active"	● Other things that they occur when printing graphically in FPW
● SET PRINTER FONT command	● Print in character mode in FPW and Visual FoxPro
● TO FILE or TO PRINTER clause	● GENERIC GENPD
● ?/?? and @SAY command	● How To Change the Default Source of Printer Programmatically
● EJECT command	● Printing Two Reports in One Duplexed Report

● "PDSETUP being active"

This refers to a **Printer Driver Setup** created through the FoxPro/DOS GENPD.APP. There are several ways a PDSETUP can be active:

1. You choose a PDSETUP through the File Menu
2. You type SET PDSETUP TO {something} (where {something} is not "")
3. You type _PDSETUP= (where is not "").
4. Your CONFIG.FP has a line in it that says you start FoxPro (where {something} is not "").
5. Your CONFIG.FP has no PDSETUP line in it, but you have in a prior FoxPro session set a PDSETUP to be the default.
6. You issue a REPORT or LABEL command with a printer driver that had checked the "[] Printer Driver Setup/Label Setup" PDSETUP when you designed the Report or Label.

Please note what happens, though, when you execute the following:

```
SET PDSETUP TO "My LaserJet Setup"
*
* After the above, you are ready to print in
```

```
*
REPORT FORM {report with no PDSETUP defined}
*
* Since the Report has no PDSETUP defined, is
* above will make FoxPro assume you want no l
* printing the report, so it "closes" your La
* PDSETUP is active during (or after) the rep
```

● SET PRINTER FONT command

There is an undocumented FPW command that was added too late to be included in any printed or on-line documentation:

```
SET PRINTER FONT {expC1} [, {expN1}] [STYLE {expC2}]
```

where {expC1} is the font name, {expN1} is the font size in points (default is 10 points), and {expC2} is the font style (default is standard style). Once you issue this command, any subsequent printed output to your printer will use that font definition (unless overridden by a FONT/STYLE clause that some commands offer. Also note that Reports/Labels with Windows objects will print their objects as defined in the Report/Label definition and not use the SET PRINTER FONT). So, for example:

```
SET PRINTER FONT "Arial",14
LIST STATUS TO PRINTER
```

The above will do a print in 14-point Arial. If no SET PRINTER FONT is in effect, then FPW prints in 10-point FoxPrint. A SET PRINTER FONT command stays in effect until either another SET PRINTER FONT command is issued or a SET PRINTER TO command is issued (which sets it back to the default of 10-point FoxPrint).

● TO FILE or TO PRINTER clause

LIST... DISPLAY... TYPE...

The above commands offer a TO FILE or a TO PRINTER clause. Let's use the LIST command in the following examples. All of the other commands above will behave the same way:

```
LIST TO FILE {destination}
```

FPW prints to the {destination} in character mode. If a PDSETUP is active, then it takes effect.

Examples:

```
LIST TO FILE LPT1
    Outputs directly to the parallel port
LIST TO FILE C:\SUBDIR\LPT1.DUM
    Does the same thing.
LIST TO FILE E:\TEST\MYFILE.TXT
    Outputs to that filename.
```

The same happens in FoxPro/DOS. It should be noted, though, that you can get nice fast character-based output to your printer in FPW by using the command LIST TO FILE LPT1.

```
SET PRINTER TO {destination}
LIST TO PRINTER
    or
SET PRINTER TO {destination}
SET PRINTER ON
LIST
```

FPW prints to the {destination} in character mode, using the PDSETUP. However, if a PDSETUP is **not** active, then what happens depends on the {destination}. If starts with the characters "PRN" or "LPT", then FPW ignores the destination and instead prints to the port defined for the currently- active Windows Printer Driver using the current SET PRINTER FONT. Otherwise, it prints to the in character mode. The following pseudo-code will illustrate this more clearly:

```
IF PDSETUP Active
    Print to {destination} in character mode
ELSE
    IF {destination} starts with "PRN" or "LPT"
        Ignore the destination and instead print to the port
        defined for the currently-active Windows Printer Driver
        with the output being printed in the character mode
        (In other words, the output is graphics)
    ELSE
        Print to {destination} in character mode
    ENDIF
ENDIF
```

Examples:

```

SET PRINTER TO LPT2 with PDSETUP Active
  Outputs to the LPT2 port in char mode us:
SET PRINTER TO LPT2 with no PDSETUP Active
  Ignores the LPT2 port and instead prints
  current Windows Printer Driver using the
SET PRINTER TO E:\MYDIR\LPTTEST.DUM with PDSI
  Outputs to that filename using the PDSETU
SET PRINTER TO E:\MYDIR\LPTTEST.DUM with no I
  Ignores the filename stipulated and inste
  defined for the current Windows Printer I
SET PRINTER FONT
SET PRINTER TO C:\TEST\TEST.TXT with PDSETUP
  Outputs to that filename in char mode us:
SET PRINTER TO C:\TEST\TEST.TXT with no PDSEF
  Outputs to that filename in character moc

```

If you have no PDSETUP active, then you can't print to a filename that starts with "PRN" or "LPT". You also cannot print directly to a parallel port. However, this obstacle can be overcome by using the TO FILE clause instead. FoxPro/DOS (of course) does not have the same behavior: If you SET PRINTER TO PRN or LPT1 or LPT2 or LPT3 then output goes to the parallel port no matter what and destinations that start with "PRN" or "LPT" are considered legitimate filenames and output goes to a file.

REPORT FORM... LABEL FORM...

The above commands behave exactly like the commands on the previous page (LIST, DISPLAY, etc). Note that this is only because the Report/Label definition contains DOS objects only. It should be mentioned once again that if the Report/Label has no PDSETUP internally defined for it, and one issues the REPORT/LABEL command with the PDSETUP clause, then Foxpro "turns off" any PDSETUP that happens to be currently active and then FPW will print the output using the currently-active Windows Printer Driver.

We'll use the REPORT command as an example; the LABEL command behaves the same way:

```
REPORT FORM reportdef TO FILE {destination}
```

FPW prints to the using the currently-active Windows Printer Driver, printing the report and all its objects as designated in the Report definition (in other words, graphical output). If the PDSETUP clause is included, it is ignored. If any PDSETUP is active when the command is issued, it is ignored in favor of the Windows Printer Driver.

Examples:

```
REPORT FORM windef TO FILE LPT3
    Outputs directly to the parallel port.
REPORT FORM windef TO FILE E:\TEST\MYFILE.TXT
    Outputs to that filename. The PDSETUP c:
```

```
SET PRINTER TO {destination}
REPORT FORM windef TO PRINTER
    or
SET PRINTER TO {destination}
SET PRINT ON
REPORT FORM windef
```

Acts the same as the TO FILE example stipulated above except for the fact that the is ignored completely. FPW assumes that if you want to print TO PRINTER, then it's going to print to the printer that is defined by the currently-active Windows Printer Driver. It will pay no attention to the you may have defined in the SET PRINTER TO command.

Examples:

```
SET PRINTER TO LPT2
REPORT FORM windef TO PRINTER
    Ignores the LPT2 port and prints to the c:
SET PRINTER TO D:\MYDIR\REPOFILE.TXT
SET PDSETUP TO "My LaserJet Setup"
REPORT FORM windef TO PRINTER
    Ignores the SET PRINTER TO filename and :
    Windows Printer Driver. The active PDSEF
```

If you want to print a Report/Label definition that contains Windows objects, then you cannot designate the destination of your output through the SET PRINTER TO command. This can be overcome by using the TO FILE clause instead or else defining all the appropriate Windows Printer Drivers you may need for output to different ports so you can select the one you want. before printing.

● ??? and @SAY command

The above commands exhibit the same behavior as the LIST TO PRINTER command discussed earlier, except in the case of a PDSETUP being active:

```
IF PDSETUP Active
```

```

IF this is the very first bit of output
  IF printing with @SAY
    IF SET PRINTER is ON
      Print to in character mode
    ELSE
      Ignore the destination and instead pr
      defined for the currently-active Wind
      with the output being printed in the
      (In other words, the output is graphi
    ENDIF
  ELSE
    Print to in character mode using the P
  ENDIF
ELSE
  Continue printing in the mode designated
  If printing with ??? and printing in cha
  then the PDSETUP is used
ENDIF
ELSE
  IF starts with "PRN" or "LPT"
    Ignore the destination and instead print
    defined for the currently-active Windows
    with the output being printed in the curr
    (In other words, the output is graphicall
  ELSE
    Print to in character mode
  ENDIF
ENDIF
ENDIF

```

It is important to know that you will not see any output until you issue a SET PRINTER TO command to close the print job.

It should be noted that if a PDSETUP is active, then the @SAY command does not execute any of the PD functions (which is also true in FoxPro/DOS). The ??? command will execute the _PDRIVER's PDOBJECT() function with whatever STYLE clause you may have stipulated being passed as a parameter to it. The PDADVPRNT() may also be called by a ??? command (and ? will cause PDLINEEND() and PDLINEST() to execute also).

When printing in character mode, there is some strange behavior that goes on when you mix output using *both* ??? and @SAY. It seems that all of the output from the ??? commands get collected into a pool of some sort and the output from the @SAY's get collected in a separate pool. When you issue the SET PRINTER TO to close the print job, the pool of ??? output gets spit out, followed by the pool of @SAY output. It seems that an attempt was made to fix this "pooling" phenomenon in version 2.5a, but it still does not act as it should, as you will see shortly.

It is interesting to note that even though this behavior is exhibited, the ???

command will still update the values of PROW() and PCOL() correctly. Consider the following program:

```

SET PRINTER TO MYFILE.TXT
SET PRINTER ON
SET DEVICE TO PRINTER
?
?? "This is on row "+LTRIM(STR(PROW()))
@ 3,3 say "Row 3 Column 3"
?? " I'm on row "+LTRIM(STR(PROW()))
?
?? "Now I'm on row "+LTRIM(STR(PROW()))
@ 6,10 say "Row 6 Column 10"
?? " This is on row "+LTRIM(STR(PROW()))
?
?? "Now I'm on row "+LTRIM(STR(PROW()))
@ 8,12 say "Row 8 Column 12"
SET DEVICE TO SCREEN
SET PRINTER OFF
SET PRINTER TO
RETURN

```

The program will give the following (expected) output in FoxPro/DOS:

```

Actual Row +-----+
          0 |
          1 | This is on row 1
          2 |
          3 |      Row 3 Column 3 I'm on row 3
          4 | Now I'm on row 4
          5 |
          6 |      Row 6 Column 10 This
          7 | Now I'm on row 7
          8 |      Row 8 Column 12
          | +-----+

```

The same program run in FPW 2.5 will output the following garbage:

```

Actual Row +-----+
          0 |
          1 | This is on row 1 I'm on row 3
          2 | Now I'm on row 4 This is on row
          3 | Now I'm on row 7
          4 |
          5 |      Row 3 Column 3
          6 |
          7 |      Row 6 Column 10
          8 |      Row 8 Column 12
          | +-----+

```

FPW Version 2.5a is different, but still not correct:

```

Actual Row +-----+
          0 |
          1 | This is on row 1 I'm on row 3
          2 |
          3 |     Row 3 Column 3
          4 | Now I'm on row 4 This is on row
          5 |
          6 |         Row 6 Column 10
          7 | Now I'm on row 7
          8 |         Row 8 Column 12
          +-----+

```

Actually, this can be solved by printing solely using @SAY statements. Just substitute all instances of "?" with "@ PROW()+1,0 SAY" and all instances of "???" with "@ PROW(),PCOL() SAY".

Don't mix ??? and @SAY's when printing in character mode!!! If you have no PDSETUP active, then you can't print to a filename that starts with "PRN" or "LPT" or print directly to a parallel port.

???

The ??? command was introduced into the FoxPro language so that you could direct output directly to the printer without incrementing PROW() or PCOL(). In fact, you don't even need SET PRINT to be ON in order for it to work.

I'm afraid I still haven't figured out all the weird things that this command does. In general, what it seems to do is (sometimes) close whatever print job may currently be going on and then starts its own, directing output to the SET PRINTER TO in character mode. However, if you're printing to a file and if any @SAY commands had been previously executed before the ???, then the ??? output and all subsequent ???/??? output will disappear.

For some examples, consider the following program, called TEST. By the way, in case you're wondering about why the WAIT WINDOW command is in here, it's to give FPW time to (possibly) start printing the print job that the first ??? command may have closed.

```

PARAMETERS the_pdsetup,the_dest,atsay_flag
SET PDSETUP TO the_pdsetup
SET PRINTER TO &the_dest
SET PRINTER FONT "Arial",30
SET PRINTER ON
IF atsay_flag
    SET DEVICE TO PRINTER

```

```

    # 1,0 SAY "Here is an @SAY in row 1"
ENDIF
? "Test Line 1"
??? "Now printing a triple"
WAIT WINDOW "Just did the first ??? command."
? "Test Line 2"
??? "Another triple"
? "Test Line 3"
?
IF atsay_flag
    # 6,0 say "Another @SAY in row 6"
    SET DEVICE TO SCREEN
ENDIF
SET PRINTER OFF
SET PRINTER TO
RETURN

```

Now let's look at some examples using the above program (without @SAY's):

```

DO TEST WITH "", "LPT1", .F.
Will output the following:
+-----+
|
| Test Line 1                                     (Printed in 3
| ***PAGE BREAK*****
| Now printing a triple                           (Printed in r
| Test Line 2Another triple
| Test Line 3
|
+-----+

DO TEST WITH "Epson-10cpi", "LPT1", .F.
Will output the following in character mode:
+-----+
|
| Test Line 1Now printing a triple
| Test Line 2Another triple
| Test Line 3
|
+-----+

DO TEST with "", "MYFILE1.TXT", .F.
DO TEST with "Epson-10cpi", "MYFILE1.TXT", .F.
Will both output the following in character m
+-----+
|
| Another triple
| Test Line 3
|
+-----+

DO TEST with "", "MYFILE2.TXT ADDITIVE", .F.
DO TEST with "Epson-10cpi", "MYFILE2.TXT ADDITIV
Will both output the following in character m

```

```

+-----+
|
| Test Line 1Now printing a triple
| Test Line 2Another triple
| Test Line 3
|
+-----+

```

Now let's introduce some @SAY commands:

```
DO TEST WITH "", "LPT1", .T.
Will output the following:
```

```

+-----+
|
| Here is an @SAY in row 1           (Printed in 3
| Test Line 1
| ***PAGE BREAK*****
| Now printing a triple             (Printed in r
| Test Line 2Another triple
| Test Line 3
|
|
| Another @SAY in row 6
|
+-----+

```

```
DO TEST WITH "Epson-10cpi", "LPT1", .T.
Will output the following in character mode:
```

```

+-----+
|
| Test Line 1Now printing a triple
| Test Line 2Another triple
| Test Line 3
|
| Here is an #SAY in row 1
| Another @SAY in row 6
|
+-----+

```

```
DO TEST with "", "MYFILE3.TXT", .T.
DO TEST with "Epson-10cpi", "MYFILE3.TXT", .T.
DO TEST with "", "MYFILE4.TXT ADDITIVE", .T.
DO TEST with "Epson-10cpi", "MYFILE4.TXT ADDITIV
Will all output the following in character mo
```

```

+-----+
|
| Test Line 1
| Here is an @SAY in row 1
|
|
| Another @SAY in row 6
|
+-----+

```

And will all output the following in character mode in FPW:

```
+-----+
|
| Here is an @SAY in row 1
| Test Line 1
|
|
|
| Another @SAY in row 6
|
+-----+
```

The only scenario in which the ??? command works correctly in concert with *both* ??? and @SAY is when printing to a port (rather than a file) using a PDSETUP. However, since printing using a PDSETUP is in character mode, the ??? and @SAY's do not print correctly in reference to each other (see ??? and @SAY pages in this document). The ??? command *does* work correctly with the ??? command alone in the scenario where you print to a File in ADDITIVE mode or when you print to a port with a PDSETUP. Bottom line: It looks like FPW treats the ??? command as a purely DOS feature.

● EJECT Command

When _PADVANCE="FORMFEED", the EJECT command essentially just does a ? CHR(12). Everything works fine with EJECT when running under a Windows Printer Driver. However, when you have a DOS PDSETUP active and an EJECT executed just prior to closing your print job, it gets lost. Actually, it seems to sit around in an internal buffer somewhere and gets output next time you start a new print job. Consider the following:

```
_PADVANCE="FORMFEED"
SET PDSETUP TO "Epson-10cpi"
SET PRINTER TO MYFILE1.TXT
EJECT
SET PRINTER TO
```

After executing the above, MYFILE1.TXT is empty. Now do the following:

```
SET PRINTER TO MYFILE2.TXT
SET PRINTER ON
?
SET PRINTER OFF
SET PRINTER TO
```

Now MYFILE2.TXT contains a formfeed, followed by the carriage return and

line feed created by the ? command.

If you do an EJECT, it may be a good idea to execute the command ?? "" immediately afterwards. This will force the form feed character to be output.

Since EJECT does a ? CHR(12), you shouldn't mix it with @SAY commands if you are printing in character mode, because of the strange behavior exhibited by mixing ?/? and @SAY commands. What happens is that all the formfeed characters get printed first and *then* all the #SAY's. I suggest you use a @ 0,0 in place of an EJECT when doing @SAY output.

When _PADVANCE="LINEFEEDS", then all bets are off if you are printing in character mode. There have been times when I've executed the following statements 10 times in a row and gotten 10 different sets of file contents:

```
_PADVANCE="LINEFEEDS"
SET PDSETUP TO ""
SET PRINTER TO MYFILE.TXT
EJECT
SET PRINTER TO
```

It seems that sometimes the print job closes too fast for all the linefeeds to get into the file.

● Other interesting things that they occur when printing graphically in FPW

Unlike FP/DOS, FPW will always give you the true values of PROW() and PCOL(). If you output REPLICATE(" ",200) to your printer in FP/DOS and then look at the value of PCOL(), it will happily give you a value of 200, even though your printer overflowed the output onto 2 or more lines. In FPW, on the other hand, PCOL() will give you the *actual* print head column position. The same is true with PROW(). If you output 100-or-so ? commands to your printer in FPD, it doesn't know that you've overflowed onto a new page, but FPW *does* and the PROW() will accurately reflect the row position on the new page.

Not only will PCOL() and PROW() return true values, but they can be translated into different font measurements as well. Try the following:

```
SET PRINTER ON
SET PRINTER FONT "FoxPrint",10
? "This is a test"
prow_before=PROW()      &&Returns 1
pcol_before=PCOL()     &&Returns 14
```

```

SET PRINTER FONT "Arial",12
prow_after=PROW( )      &&>Returns 0.714
pcol_after=PCOL( )     &&>Returns 15.556
SET PRINTER OFF
SET PRINTER TO

```

The decimal portion of the PROW() and PCOL() values are dependent on the resolution (dots/inch) of your printer. For example, let's look at 10-point FoxPrint on a 300-dpi Laser Printer. This font has a 12 characters/inch pitch horizontally, which comes out to $300/12=25$ dots/character, so the most horizontal precision you can achieve in column coordinates is $1/25=0.04$. By the same token, a 10-point character is $10/72$ inches tall (1 point= $1/72$ inch). So $300*10/72=42$ dots/character vertically, and vertical precision is $1/42=0.0238$. If we analyzed the same size character on a dot-matrix printer with 144x120 (vertical x horizontal) dots/inch resolution, you could get no more precise than 0.05 units vertically (20 dots) or 0.10 units horizontally (10 dots).

All that this precision stuff above means is that you can only place objects on the printed page with a limited amount of accuracy, and that depends on your printer. For example, if I attempt to position the 300-dpi laser print head by issuing the command @ 12.03,15.03 the print head will *actually* be positioned at the closest dot coordinate, which is actually 12.0238,15.04. On the 144x120 dot-matrix printer, the actual coordinates would end up being 12.05,15.00. You can verify this by looking at the values of PROW() and PCOL() when positioning your output via the @SAY command.

The ? command has an interesting property: It will always move the print head to the next *integral* printer row. Try the following code:

```

SET PRINTER ON
SET PRINTER FONT "Arial",10
? "Line 1"      &&Prints in 10-point Arial
? "Line 2"
? "Line 3"
? "Line 4" FONT "Arial",11
? "Line 5" FONT "Arial",11
? "Line 6" FONT "Arial",11
SET PRINTER OFF
SET PRINTER TO

```

You'll notice that the first 3 lines appear normal, but lines 4 through 6 are spread apart vertically. This is because FPW is advancing the print head to the next integral *10-point Arial* row (since that is the active SET PRINTER FONT). Since 11-point Arial is slightly taller than 10-point Arial, the ? command is forced to do a line-feed to the next available 10-point row, which ends up being 2 rows below the 11-point line.

The same thing happens if you mix fonts on the same line. FPW somehow keeps track of the tallest font and performs a line-feed far enough down for that tallest font to be visible. Output the following to your printer:

```
? "10" FONT "FoxPrint",10
?? "12" FONT "FoxPrint",12
?? "30" FONT "FoxPrint",30
?? "20" FONT "FoxPrint",20
? REPLICATE("X",20) FONT "FoxPrint",10
```

You will notice that the line of X's on that second line prints just below the 30-point characters of the first line.

By the way, this "integral row" with the ? command also occurs when outputting to the screen.

● Print reliably in character mode in FPW and Visual FoxPro

If you want to print reliably in character mode in FoxPro/Windows, follow these rules:

- 1) Use a PDSETUP. If you don't use PDSETUPS else use my Generic GENPD that I outline a
- 2) Create and edit all your Reports and Labels transport them. Of course, if you want to graphically rather than character-based, then edit them in FPW and/or use the transport
- 3) Don't mix output with the ?/?? and @SAY commands your job or all @SAY's.
- 4) As far as EJECT is concerned, issue a ?? command sure that the formfeed character is output using #SAY commands, then do an @ 0,0 rather
- 5) If you want to use the ??? command, then use ?/?? commands; don't use @SAY commands in
- 6) Also with the ??? command, make sure you use SET PRINTER TO destination.
- 7) If you're outputting data to your printer make sure you issue a SET PRINTER TO command so and starts physically printing to your printer.

In general, using a PDSETUP will solve the vast majority of your character-

based printing problems.

● GENERIC GENPD

If you don't usually use PDSETUPs, then outlined below is how to create a whole GENPD application which you can use to force FPW to print everything in character mode like FP/DOS. Although it's essentially useless (redundant) in FP/DOS, it will work in that environment also.

Step #1: Create a file called GENRICPD.PRG:

```
*
* GENRICPD.PRG - Generic Printer Driver Set
*
PARAMETERS calltype,pdname
PRIVATE retval
IF PARAMETERS()=0    &&Called from CONFIG.FI
    calltype=0
    pdname=" "
ENDIF
IF calltype=2        &&Called from FPD Repo
    RETURN IIF(EMPTY(pdname),"Generic","")
ENDIF
retval=" "
IF pdname="?"        &&Called from FPD File,
    retval=IIF(EMPTY(_PDSETUP),"Generic","")
ELSE
    retval="Generic"
ENDIF
_PDRIVER=" "        &&Close current Printer
IF NOT EMPTY(retval)
    PUBLIC _PDPARMS[3]
    _PDPARMS[1]="Generic"
    _PDPARMS[2]=.F.    &&PdPageEnd() just e
    _PDPARMS[3]=.F.    &&PdLineEnd() just e
    _PDRIVER="GENRICDV.PRG" &&Open Printer I
    (calls PDONLOAD if exists)
ENDIF
_PDSETUP="-"+retval    &&The "-" prevents a
RETURN retval
```

Step #2: Create a file called GENRICDV.PRG:

```
*
* GENRICDV.PRG - Generic Printer Driver Pro
*
PROCEDURE PdOnUnload
```

```

RELEASE _PDPARMS
RETURN

FUNCTION PdDocSt
PARAMETERS height,width
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.F.
RETURN ""

FUNCTION PdPageSt
PRIVATE ctlchars
ctlchars=IIF(_PDPARMS[2],CHR(12)+CHR(13))
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.F.
RETURN ctlchars

FUNCTION PdPageEnd
  _PDPARMS[2]=.T.
  _PDPARMS[3]=.F.
RETURN ""

FUNCTION PdLineSt
PRIVATE ctlchars
ctlchars=IIF(_PDPARMS[3],CHR(13)+CHR(10))
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.F.
RETURN ctlchars

FUNCTION PdLineEnd
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.T.
RETURN ""

FUNCTION PdAdvPrt
PARAMETERS here,there
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.F.
RETURN IIF(here>there,CHR(13)+SPACE(there))

FUNCTION PdObject
PARAMETERS theobj,objstyle
  _PDPARMS[2]=.F.
  _PDPARMS[3]=.F.
RETURN theobj

```

Step #3: Create a Project called GENRICPD.PJX that contains the 2 programs GENRICPD.PRG and GENRICDV.PRG, making GENRICPD.PRG the "Main" Program. Then Build the Project into an APP called GENRICPD.APP.

You can put this Generic PDSETUP into place by executing the following two commands:

```
_GENPD="GENRICPD.APP"
_PDSETUP="Generic"
```

You may need to fully qualify the GENRICPD.APP in the above _GENPD line if the APP is in a different directory. You can either execute the above 2 commands from within FPW or put the following 2 lines into your CONFIG.FPW so that the Generic PDSETUP is in effect every time you start FPW:

```
_GENPD="GENRICPD.APP"
PDSETUP="Generic"
```

By the way, it's interesting to note that only the _GENPD="GENRICPD.APP" line is required in your CONFIG.FP if you are starting FoxPro/DOS. FoxPro/DOS automatically executes the _GENPD application on startup; whereas, FPW does not. That is why you must add the PDSETUP="Generic" to your CONFIG.FPW.

Because of the way I've written GENRICPD.PRG, you can put almost any string into the _PDSETUP system variable and it will cause the PDSETUP to be in effect (and will set the system variable _PDSETUP to "Generic" anyway). The exceptions are:

```
_PDSETUP=" "           &&Turns the PDSETUP off
_PDSETUP="?"         &&Toggles the PDSETUP on/off
_PDSETUP="-"+{any string} &&Does not actually execute
```

The reason for the "?" toggle is because of how FoxPro/DOS calls the _GENPD application from its File Menu's Printer Setup Option. And, as outlined in the FoxPro Printer Driver documentation, setting _PDSETUP to any string that starts with a dash ("-") will set the _PDSETUP system variable to the portion of the string after the dash, but will not actually execute the _GENPD application.

Once you have the Generic PDSETUP active, then all printer output in FPW will be just like printing without a PDSETUP in FoxPro/DOS, as long as you follow the rules I stipulated in the "Print in character mode" section of this document.

● How To Change the Default Source of Printer Programmatically

● Microsoft Visual FoxPro for Windows, versions 3.0, 3.0b, 5.0

To set the default source (upper/lower tray) of a printer programmatically under Windows 95 and Windows NT 4.0, use the SetPrinter() Win32 API function.

Note that the SetPrinter() API call is platform-dependent and works only on Windows 95 and Windows NT 4.0. (It does not work on earlier versions of Windows NT or on a 16-bit platform.)

To set or change the default source of printer under Windows 95 and Windows NT 4.0, call the following:

```
SetPrinter(HANDLE hPrinter, DWORD dwLevel, LPBYTE lpbPrinter,
dwCommand)
```

Obtain the hPrinter parameter from OpenPrinter() as a handle that identifies the desired printer. Set the dwLevel parameter to 2, and point lpbPrinter to the PRINTER_INFO_2 structure. Set the dwCommand parameter to 0. Fill out the PRINTER_INFO_2 structure appropriately.

There are few ways of changing the (upper/lower) printer tray dynamically. The most common is to use PCL printer commands. To do this, insert the escape sequence in the file, along with the content you are trying to print. Unfortunately, this method makes it difficult to work with Visual FoxPro since you are limited by the printer's settings (you need to know the designated printer beforehand), and the reports in Visual FoxPro use the printer driver internally once the printing job is established.

The following sample program illustrates the contents of the .dll file written in Microsoft Visual C++ 4.0. It shows a way to retrieve a handle identifying the specified printer or print server.

NOTE: This sample program illustrates many Microsoft Visual C++ commands. The use of these commands is beyond the scope of Microsoft FoxPro Product Support. Users with substantial experience using API routines should be able to write the following sample .dll file. For this sample to work, you need a .def file to export the chgbin function.

Sample Program (DLL to Change the Default Source of Printer)

```
#include "stdio.h"
#include <windows.h>
```

```
BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called,
LPVOID lpReserved) {
```

```
    return 1;
    UNREFERENCED_PARAMETER(hInst);
    UNREFERENCED_PARAMETER(ul_reason_being_called);
    UNREFERENCED_PARAMETER(lpReserved);
}

#define ErrReturn    if (GetLastError()) {ClosePrinter(hPrinter),
printf("error"); return -1;}

short FindID(LPPRINTER_INFO_2 pPrinter, int flg);

BOOL MyFreeMem(LPVOID pMem) {
    return VirtualFree(pMem, 0, MEM_RELEASE);
}

#define UPPER_BIN 1
#define LOWER_BIN 2

int APIENTRY chgbin(char *ptrname, int flg)
{
    HANDLE    hPrinter = NULL;
    DWORD     cbBuf;
    DWORD     pcbNeeded = 0;
    LPTSTR    pPrintername;
    short     nSource;

    pPrintername = ptrname;

    PRINTER_DEFAULTS pd;
    ZeroMemory(&pd, sizeof(pd));
    pd.DesiredAccess = PRINTER_ALL_ACCESS;

    int result1 = OpenPrinter(pPrintername,&hPrinter, &pd);
    ErrReturn;
    int result = GetPrinter(hPrinter, 2, NULL, 0, &pcbNeeded);
    DWORD Error = GetLastError( );

    if( Error == ERROR_INSUFFICIENT_BUFFER )
    {
        BOOL bRet = FALSE;
        HANDLE hMem = NULL;
        LPPRINTER_INFO_2 pPrinter;
    }
}
```

```

        hMem = GlobalAlloc(GHND, pcbNeeded);
        if (hMem) pPrinter = (LPPRINTER_INFO_2)GlobalLock(hMem);
        cbBuf = pcbNeeded;
DWORD cbNeeded;
if (GetPrinter(hPrinter, 2, (LPBYTE)pPrinter, pcbNeeded, &cbN
{
    if ((nSource = FindID(pPrinter,flg)) < 0) return -1;

    pPrinter->pDevMode->dmDefaultSource = nSource;
    pPrinter->pDevMode->dmFields = DM_DEFAULTSOURCE;

    DocumentProperties(NULL,hPrinter,pPrintername,pPrinter-
>pDevMode,pPrinter->pDevMode,

        DM_IN_BUFFER|DM_OUT_BUFFER);
    SetPrinter(hPrinter,2,(unsigned char *)pPrinter,0);

    MyFreeMem(pPrinter);
    ClosePrinter(hPrinter);
}
}

Error = GetLastError( );
return 0;

}

typedef struct _tagDevCaps {

    TCHAR    pPrinterName[80];
    TCHAR    pPort[80];
    WORD    wCurCap;
    WORD    wCurPlatForm;
    HINSTANCE hDriver; //only used if on Win32s;
    DWORD    (CALLBACK* pfnDevCaps) (
        LPTSTR    pDevice, // address of device-name string
        LPTSTR    pPort, // address of port-name string
        UINT      fwCapability, // device capability to query
        LPTSTR    pOutput, // address of the output
        LPDEVMODE pDevMode // address of structure with device c
    );

} DEVCAPS;

LPVOID MyAllocMem(DWORD cb) {

    return VirtualAlloc(NULL, cb, MEM_RESERVE|MEM_COMMIT, PAGE_RI

}

```

```

#define MAX_AMOUNT      256
#define MAX_BINS       16

short FindID(LPPRINTER_INFO_2 pPrinter, int flg) {

    DEVCAPS MyDevCaps;
    MyDevCaps.hDriver = NULL;
    MyDevCaps.pfnDevCaps = NULL;
    BOOL bRet = FALSE;

    lstrcpy(MyDevCaps.pPrinterName, pPrinter->pPrinterName);
    lstrcpy(MyDevCaps.pPort, pPrinter->pPortName);
    MyDevCaps.pfnDevCaps = (LPFNDEVCAPS)&DeviceCapabilities;

    DWORD      dwBufSize = 0;
    BOOL       bResult = 1;
    WORD FAR   *pawBinList;

    if (MyDevCaps.pfnDevCaps)
    {
        // get number of bins
        dwBufSize = MyDevCaps.pfnDevCaps ((LPTSTR )MyDevCaps.pPrin

(LPTSTR )MyDevCaps.pPort, (WORD)DC_BINS,

                                (LPTSTR )NULL, (LPDEVMODE)NUI

        pawBinList = (WORD FAR *)MyAllocMem(dwBufSize* (sizeof(WC

        // fill buffer with bin list
        MyDevCaps.pfnDevCaps ((LPTSTR )MyDevCaps.pPrinterName, (I

)MyDevCaps.pPort, (WORD)DC_BINS,

                                (LPTSTR )pawBinList, (LPDEVMODE)NULL);

        // display bin info
        // protects from bad drivers

        if ((dwBufSize > 0) && (dwBufSize < MAX_AMOUNT))
        {
            for (int i=0; i< (int)dwBufSize;i++)
            {
                if (pawBinList[i] < MAX_AMOUNT)
                {
                    if (pawBinList[i] < MAX_BINS)
                    {
                        if (flg == UPPER_BIN && pawBinList[i] == UPPER_
                        return (pawBinList[i]);
                        else if (flg == LOWER_BIN && pawBinList[i] == LOWI

```

```

        return (pawBinList[i]);
    }
}
}
}
// clean up
MyFreeMem(pawBinList);

return (-1);
}
return (-1);
}
}

```

In the FoxPro Application

```

DECLARE integer chgbin IN c:\bin.DLL STRING, INTEGER

** 1 = Upper    2 = Lower
retval = chgbin("HP LaserJet 4Si MX",2)

USE CUSTOMER
LIST TO PRINT

```

Sample Program (Notes)

- To create a .dll file in Visual C++, please refer to Help in Microsoft Visual C++ 4.0.
- The Chgbin.dll should be in the same directory as your project, or should be in the path of the Windows Win32 directory.

WARNING: Any use by you of the code provided in this article is at your own risk. Microsoft provides this code "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose.

The function chgbin takes two parameters. The first parameter takes the actual printer name. You could get the printer name from FoxPro by using the APRINTER() function, which retrieves all the existing drivers in the current windows operating system. The second parameter is a flag that tells the chgbin function how to set the upper/lower tray. The numeric value 1 indicates that the upper tray should be used, and the numeric value 2 indicates that the lower tray is being used.

This DLL does not work properly if the printer is a network printer. The network

printer cannot be configured programmatically since multiple users need to access the printer at the same time. In order to solve this problem, you need to add a local printer driver and force the local driver to print to a file (where the file is redirected to network printer address). In this way, the printer driver can have its own settings, which can be used by a FoxPro application to refer to the network printer.

It is highly recommended to change the setting back to its default (or previous setting) once the program is terminated since other applications might use the same driver.

Visual FoxPro stores the printer and bin to use inside the .frx file for reports. So changing the paper source for the Printer Driver in Windows does not cause the Report to print to a different paper source.

● **Printing Two Reports in One Duplexed Report**

● **Microsoft Visual FoxPro for Windows, versions 3.0, 3.0b, 5.0, 5.0a**

Because FoxPro reports are sent as separate print jobs, they are also printed on separate pages. So, if you have two one-page reports and you want to have them printed as one duplex report, you would have to combine the reports into one, or combine the output of the two reports--as the program code in this article illustrates.

NOTE: The COPY FILE command used at the end of this program may not work under Windows NT. This is another issue being researched at this time.

There are some cases where extra characters may be printed at the top of the two pages, but adjustments can be made to the code to prevent this. Specifically, you need to change the adjustment on the lines using the FSEEK() function so that the file pointer is moved by fewer characters.

Please note the assumptions that this code makes:

- You have a printer connected to a local printer port (LPT1 for this code to work without modification), or you have captured a network printer to LPT1.
- You have only two reports to be printed together, and they print one page each.

One additional note is that this program creates several files as temporary files in

the default directory. Please take note of the names of the files so as not to overwrite any files you may already have (since SAFETY is turned OFF). These temporary files are not deleted, but could be with only a few additional lines of code.

```

**=====**
** Begin program **
**=====**
***** Environment Settings
** Printer Settings... this allows the user to
** choose which printer settings will be used in
** creating the file. But remember, the file
** will still be sent to the port indicated in
** the COPY FILE command at the end of the program.
SET PRINTER TO GETPRINTER()

** Safety should be off in case we have
** already run this procedure before
tmpsafe = SET("SAFETY")
SET SAFETY OFF
***** End environment settings

***** Create template file
** Create blank report
IF FILE("blank.frx")
    DELETE FILE blank.frx
ENDIF
DEFINE WINDOW test FROM 0,0 TO 10,10
KEYBOARD "{CTRL+W}"
CREATE REPORT blank
RELEASE WINDOW blank
RELEASE WINDOW test

** Change the printer settings to ensure duplex printing is
** turned on and use selected settings. Please note that
** duplex printing will work only on printers that support it
USE blank.frx
REPLACE expr WITH "DUPLEX=2" && turn duplex on
USE

** Create blank table to cover two pages
SELECT 0
IF USED("blank")
    USE IN blank
ENDIF
IF FILE("blank.dbf")
    DELETE FILE blank.dbf
ENDIF
CREATE TABLE BLANK (test C)
FOR nCnt = 1 TO 25
    APPEND BLANK

```

```
ENDFOR

** Create template print file from blank report
REPORT FORM blank TO template.prn
USE IN blank

** Create variables for each line of the template file
htemplate = FOPEN("template.prn")
linecnt = 0
DO WHILE !FEOF(htemplate)
    linecnt = linecnt + 1
    lnvar = "line" + ALLTRIM(STR(linecnt))
    STORE FGETS(htemplate) TO &lnvar
ENDDO
=FCLOSE(htemplate)
***** End create

***** Process the individual reports
** Remove printer settings from the reports
** so they use the selected settings
&& First report
report1 = GETFILE("FRX","Please select first report")
IF EMPTY(report1)
    =MESSAGEBOX("You did not specify a file for the first report"
        CHR(13) + "so this program cannot continue.")
    RETURN
ENDIF
USE (report1) IN 0 ALIAS one
SELECT one
onetmp = one.expr
REPLACE expr WITH ""
USE IN one
&& Second report
report2 = GETFILE("FRX","Please select second report")
IF EMPTY(report2)
    =MESSAGEBOX("You did not specify a file for the second report"
        CHR(13) + "so this program cannot continue.")
    RETURN
ENDIF
USE (report2) IN 0 ALIAS two
SELECT two
rtmp = two.expr
REPLACE expr WITH ""
USE IN two

** Send reports to files
REPORT FORM (report1) TO one.txt NOCONSOLE
REPORT FORM (report2) TO two.txt NOCONSOLE

** Restore printer settings to the reports
&& First report
USE (report1) IN 0 ALIAS one
```

```
SELECT one
REPLACE expr WITH rtmp
USE IN one
&& Second report
USE (report2) IN 0 ALIAS two
SELECT two
REPLACE expr WITH rtmp
USE IN two

** Determine the size of the input files
tmpdb4 = SET("COMPATIBLE")
SET COMPATIBLE ON
onesz = FSIZE("one.txt")
twosz = FSIZE("two.txt")
SET COMPATIBLE &tmpdb4
***** End process

***** Create final print file
** Delete the file, if it exists, and create
** a new file to contain the reports
IF FILE("final.prn")
    DELETE FILE final.prn
ENDIF
hFinal = FCREATE("final.prn")

** Write header codes
totchar = 0 && this will contain the length of the header
FOR nCnt = 1 TO linecnt-2
    lnvar = "line" + ALLTRIM(STR(nCnt))
    totchar = totchar + FPUTS(hFinal,EVAL(lnvar))
ENDFOR
pgvar = "line" + ALLTRIM(STR(linecnt-1)) && page break line
ejvar = "line" + ALLTRIM(STR(linecnt)) && end of job line

** Write body of first report file
hOne = FOPEN("one.txt")
=FSEEK(hOne,totchar-(linecnt-2))
=FWRITE(hFinal,FREAD(hOne,onesz-totchar-LEN(EVAL(ejvar))))

** Write pagebreak codes
=FPUTS(hFinal,EVAL(pgvar))

** Write body of second report file
hTwo = FOPEN("two.txt")
=FSEEK(hTwo,totchar-(linecnt-2))
=FWRITE(hFinal,FREAD(hTwo,twosz-totchar-LEN(EVAL(ejvar))))

** Write end-of-job codes
=FPUTS(hFinal,EVAL(ejvar))

** Close all files
=FCLOSE(hFinal)
```

```
=FCLOSE(hOne)
=FCLOSE(hTwo)
***** End create

***** Send the final file to the printer
** This line copies the file to the printer
** connected to LPT1: in Windows. You may also
** specify LPT1 or LPT2 instead of PRN.
COPY FILE final.prn TO PRN
***** End send

***** Restore Environment Settings
SET SAFETY &tmpsafe
***** End restore
**=====**
** End program **
**=====**
```

Send email to webmaster@programatica.com for further assistance or call +507
260-6462.

Hosted By  Programatica WEBDesign ®

Powered By



SiliconGraphics™

Copyright © 1996-1997. Programatica WEB Design ®.
All Rights Reserved Worldwide.