Issue Date: FoxTalk December 2000

Report Writer Tips and Tricks

Cathy Pountney
cathy@frontier2000.com

Output is one of the key things developers have to create for users. It's not always fun, but it's one of those necessary evils we have to do. Over the course of the next few issues, Cathy Pountney is going to show you several tips and tricks that will allow you to push the limits of the standard FoxPro Report Writer and overcome some of its limitations.

FoxPro's Report Writer is a good tool, but it's certainly not perfect. The Report Writer available in Visual FoxPro 6.0 is virtually the same Report Writer that was available in the days of 2.x. Because of this fact, many developers have resorted to third-party tools like Crystal Reports or FoxFire! Report Writer.

Others, however, have learned to work around the limitations of the standard Report Writer included with FoxPro. After all, this is FoxPro, and you can do *anything* with FoxPro! It just takes a little ingenuity, a little creativity, and a lot of time beating your head against the wall until you discover the solution. This article—and my subsequent articles—is dedicated to all of those developers who have done just that. I'm not claiming to have discovered all of these tips and tricks. The FoxPro community has always been a tight-knit group that shares ideas with each other, and I'm just compiling them in one place so you can benefit from the ideas of others without having to go through the head-banging yourself.

So, let's get started. In this month's installment, I'm going to cover six different tips and tricks like suppressing the Printing... dialog box and printing page x of y. The accompanying **Download file** includes working samples that incorporate each of these tips and tricks.

Suppress the Printing... dialog box

The bad news—you can't suppress it. The good news—you can fake it out!

The Printing... dialog box always appears when you print to the printer. This, you can't stop. But you can move the window off the screen so it *appears* as if you've suppressed it. Place the following code in a UDF or class method that's accessible from the report:

```
 IF WEXIST("Printing...")
   MOVE WINDOW "Printing..." TO -1000, -1000
ENDIF
```

In the Title band of the report, call the UDF or class method you just created. If you're working with Visual FoxPro, put the call in the "On Entry" portion of the Title band. If you're working with FoxPro 2.x, put the call in the expression of a field placed in the Title band. Also, if you're working with FoxPro 2.x, remember to use RETURN '' to return from your UDF. Otherwise, ".t." will appear on your report where you've placed the field. If you don't really need a Title band, change the band height to zero.

When the user prints the report, the dialog box will flash for a moment and then *disappear*. However, realize that it hasn't really disappeared. It's still there. You just can't see it. So if you had a glimmer of hope thinking that this would allow you to print reports from COM objects—no such luck! You'll have to take this one up with the Fox Team <s>.

This trick also paves the way for you to create and display your own printing dialog box. You could display your own friendlier dialog box, complete with the full descriptive name of the report (instead of invxyz.frx). And if you knew the total page count before printing (see the section "Print page x of y" later in this article), you could even display a progress bar in your dialog box that updates the progress after each page.

Prompt for printer from preview

Note that this tip doesn't apply to FoxPro 2.x; sorry, guys!

To preview a report, issue the following:

```
 REPORT FORM report1 PREVIEW
```

The report is displayed on the screen along with a button to print a hard copy of this report. Unfortunately, the user isn't prompted to select a printer. Instead, the default printer is used, which might not always be desirable. To allow the user to preview the report *and* to have the option to select a printer, use this command:

```
REPORT FORM report1 TO PRINTER PROMPT PREVIEW
```

By combining the TO PRINTER PROMPT and PREVIEW clauses, the user gets full control over whether and when they print to the printer, as well as which printer the output goes to. FYI, you *must* put PREVIEW after TO PRINTER PROMPT. If you reverse these two clauses, you'll get an error when you try to run it. Trust me—my head's still a little sore from hitting the wall over this one.

Is the user previewing or printing?

Many times you need to know whether the user is previewing the report on the screen or printing the report to the printer. For example, if the report is for invoices that are printed on preprinted forms, you wouldn't include the company name, address, logo, or graphical lines on the report. Yet, this makes it difficult to read when previewed on the screen. The solution is to include the company name, address, logo, and graphical lines on the report, but only print them when the user is printing to the printer. Simply place the following code in the "Print When" expression of each object:

```
NOT WEXIST('Printing…')
```

Since the Printing… dialog box only appears when the report is output to the printer, checking for the existence of this window determines whether the user is previewing or printing. If you've suppressed this window with the trick mentioned earlier, this logic still holds true. Remember, you didn't *suppress* the dialog window, you *hid* the dialog window.

Did the user cancel printing?

Often, you need to update some type of "printed" flag once the user has printed a particular report. You might have decided to put your update code right after your code to print the report. But there's a better way, and here's why. First, you need to make sure the user really did *print* the report. You wouldn't want to update the flag if the user only previewed the report. Second, you need to make sure the user didn't cancel the report before it was finished. To do this, start by placing the following code in a UDF or class method that's accessible from the report:

```
IF WEXIST("Printing...")
  lPrinted = .t.
ENDIF
```

Next, use the following code to call your report:

```
PRIVATE lPrinted
lPrinted = .f.
REPORT FORM xxxxxx NOCONSOLE TO PRINTER PROMPT PREVIEW
IF l_Printed
  DO Update_Code
ENDIF
```

Finally, call the UDF or class method from the summary band of the report. If you're working with Visual FoxPro, put the call in the "On Exit" portion of the Summary band. If you're working with FoxPro 2.x, put the call in the expression of a field placed in the Summary band. As mentioned earlier, if you're working with FoxPro 2.x, make sure you use RETURN '' to exit your UDF to avoid printing ".t." on the final report.

After the report command executes, the preceding code conditionally performs the update routine only if lPrinted is true. This ensures that the user printed the report and didn't cancel before it completed. Of course, you can't be sure it physically printed, but that's something over which you have no control.

Print page x of y

This has to be one of the most, if not *the* most, frequently asked questions by FoxPro developers. Unfortunately, there isn't any special variable created by FoxPro to hold the total number of pages. You have to create a memory variable yourself by printing the report twice—once as a phantom print to determine the total number of pages and once to actually print the report for the user.

First, declare a memory variable for holding the page count (nPages). Next, print the report to a file. When the report is done processing, the _PAGENO memory variable will contain the page number of the last page. Save this to your memory variable (nPages = _PAGENO) and print the report again, this time to the printer. In the Report Designer, you can use the following expression to print the page number:

```
'Page ' + ALLTRIM(STR(_PAGENO)) + ' of ' ;
+ ALLTRIM(STR(_nPages))
```

Although this might sound easy at first, the final solution isn't quite so simple. There are a few issues you need to deal with.

First, you need to prompt the user for the printer *before* you print the phantom report. Otherwise, you could end up with an incorrect page count. When printing to a file, the default printer driver is used. It's possible that the user would choose a different printer that has a printable area different from the default printer's. This means that the phantom report might break at different points than the actual report, thus causing the final page count to be different.

Another issue you need to deal with is getting a unique filename to print the phantom report to and then erasing that file when you're done. You wouldn't want a bunch of temporary files to build up on your user's system. You might be tempted to avoid printing to a file by using REPORT FORM xxxx NOCONSOLE. Although this *might* work, it doesn't *always* work. I'm not sure why, but sometimes this method doesn't always return the correct page count. You're better off sticking with printing to a file even though it means more work for you.

The last issue you need to deal with is determining whether the user canceled the first phantom print. You can do this with a slight modification to the UDF described earlier in the "Did the user cancel printing?" section. Here's the complete code to print this report:

```
 * Initialize variables
PRIVATE lPhantom, lPrinted, nPages, cTemp
lPrinted = .f.
nPages = 0
cTemp = SUBSTR(SYS(2015), 3, 10) + '.txt'
* Prompt for the printer
=SYS(1037)
* Print the Phantom report
WAIT WINDOW ;
  'Calculating Page Count...' NOWAIT
lPhantom = .t.
REPORT FORM report1 NOCONSOLE TO FILE &cTemp
nPages = _PAGENO
ERASE (cTemp)
lPhantom = .f.
WAIT CLEAR
* Print the real report
IF lPrinted
  REPORT FORM report1 TO PRINTER PROMPT PREVIEW
ENDIF
******************
PROCEDURE Suppress
******************
* Suppresses the "Printing..." dialog
IF WEXIST('Printing...')
  MOVE WINDOW 'Printing...' TO -1000, -1000
ENDIF
RETURN
*****************
PROCEDURE Printed
*****************
* Update lPrinted flag
IF lPhantom
  lPrinted = .t.
ENDIF
RETURN
```

Because you're printing the report to a file on hard disk before printing the actual hard copy, it will take much longer to print the final report for the user. If the report is only expected to be a few pages, the user will probably be willing to live with the performance hit. In fact, he or she might not even notice it. But if the report is expected to be several hundred pages long, the user might not be willing to wait for the double processing. Be sure your users understand the penalty for printing the total page count.

Chain several reports together

I have a client who has one report that, in their mind, consists of three different FoxPro reports. This client wants to choose one option from the menu, select the printer once, and have the full report print. What I'd like to code is something like this:

```
 REPORT FORM report1 NOCONSOLE TO PRINTER PROMPT
REPORT FORM report2 NOCONSOLE TO PRINTER ADDITIVE
REPORT FORM report3 NOCONSOLE TO PRINTER ADDITIVE
```

Unfortunately, the ADDITIVE clause isn't part of the REPORT command. But you can pull this scenario off by using the SYS (1037) command. This command prompts the user with a printer dialog box and leaves that printer selected so you can call each of your reports, one after the other, as shown here:

```
 =SYS(1037)
```

```
REPORT FORM report1 NOCONSOLE TO PRINTER
REPORT FORM report2 NOCONSOLE TO PRINTER
REPORT FORM report3 NOCONSOLE TO PRINTER
```

But wait—there's more! This client's scenario gets a little trickier. Besides printing all three reports as one, they also want the page numbers to continue from one report to the next. For example, if each report consists of three pages, "report1" needs to show pages 1-3, "report2" needs to show pages 4-6, and "report3" needs to show pages 7-9 in the footer. To do this, I use a memory variable to offset the standard FoxPro page numbers, as shown here:

```
 PRIVATE nOffset
nOffset = 0
=SYS(1037)
REPORT FORM test1 NOCONSOLE TO PRINTER
nOffset = nOffset + _PAGENO
REPORT FORM test2 NOCONSOLE TO PRINTER
nOffset = nOffset + _PAGENO
REPORT FORM test3 NOCONSOLE TO PRINTER
```

In each of the report definitions, I use _PAGENO + nOffset, instead of _PAGENO, to print the page number in the footer.

Looking ahead

Hopefully, I've shed some new light on the FoxPro Report Writer for you. In future articles, I'm going to shed even more light and cover many more tips and tricks. Look for coverage of multiple detail bands, multi-page detail bands, and dynamic graphic images in upcoming issues of *FoxTalk*. If you have an idea you'd like to see covered, or if you've discovered something yourself, drop me an e-mail and I'll see what I can do about including it in a future issue.