
Chapter 10 Printing & the Print Manager

Choose printer from software?.....	134
Generic Printing code (complete Source).....	134

USENET Macintosh Programmer's Guide

From: mm5l+@andrew.cmu.edu (Matthew Mashyna)

Subject: Re: Choose printer from software?

Charles Oldham writes:

- > Is there a way to choose a printer through software? I.e.
- > not using the Chooser ?

I was trying to do this and did get it to work. I got flamed a lot for asking how to do it too!

<flame guard on>

You can set the printer selection by reading in the PAPA resource -8192 from the LaserWriter cdev. The resource contains the pascal strings for the name, type and zone followed by the net address -- net, node & socket (order ?). If my memory's right it's 103 bytes (better check.)

The crudest thing you can do is zero out the resource and stuff your string values for the name type and zone into the resource. The first time you go to print the LaserWriter drivers will have to work a little harder to find the proper address (because you stuffed in 0's). You could go farther and do an NBP lookup for the address and finish the job too.

I know this is really a bad thing to do but I needed to do it for a TCP/IP LPR daemon to direct output from Unix & VMS machines to many Appletalk printers.

Matt Mashyna

●●●

From: Richard M. Siegel

Subject: Generic Printing code (complete Source)

[Rich posted this to the net a while back. MXM]

```
{UGenericPrint.p}
{©1989 Richard M. Siegel, all rights reserved}

{UGenericPrint provides a standard interface for programs that wish to print images}
{using the Print Manager.}
{It will take care of most of the gory details of printing.}

{History:}
{}
{ 4/12/89   RMS Creation}

{Lightspeed Pascal source options: Courier 10, 4 spaces per indent, 4 spaces per tab.}

unit UGenericPrint;
interface

{For MPW Pascal, the USES list will have to be changed. Also, you may wish to use MacPrint
if}
{you're going to run on versions of the System earlier than 4.1}
uses
    MacPrint, editorGlobals;

{Function "GenericPrint" handles all of the printing details. Its arguments
}
{are:
    }
{
    }
```

```
{  hPrint :    A Print Manager handle. If hPrint is NIL, then GenericPrint      }
```

USENET Macintosh Programmer's Guide

```
{
    will allocate its own default print handle and dispose it
    before exiting.
}

{
}

{
    statusDialog :   A pointer to a dialog box which will be displayed while
}
{
    printing. If statusDialog is NIL, no dialog will be displayed. }
{
}
{
}

{
    UserPrepProc :   A function that performs any pre-printing initializations that
    may be required. It should return TRUE if it completed
    successfully, FALSE to abort printing. One thing that the
    UserPrepProc MUST DO: it is imperative that the UserPrepProc set
    hPrint^.prJob.iLstPage to the correct number of pages in the
    document. If all pages are specified in the print dialog, then
    this field of the print handle will be 9999; if this is the case,
    set it to the correct number of pages.
}

{
}

{
    UserPageProc :   A function that does the actual drawing on the page. It is
    supplied with a print handle, a rectangle representing the
    drawing area, and a page number. UserPageProc should return
    TRUE if it completed successfully, or FALSE if you wish to
    abort the printing process. The current grafPort will be the
    printing grafPort, so if your routine changes it, be sure to
    restore it.
}

{
}

{
    GenericPrint will return noErr if printing was completed successfully,
}
{
    and a negative OS result code otherwise.
}

function GenericPrint (hPrint: THPrint; statusDialog: DialogPtr; function UserPrepProc
(hPrint: THPrint): Boolean; function UserPageProc (hPrint: THPrint; pageRect: Rect; pageNum:
Integer): Boolean): OSErr;

implementation
function GenericPrint (hPrint: THPrint; statusDialog: DialogPtr; function UserPrepProc
(hPrint: THPrint): Boolean; function UserPageProc (hPrint: THPrint; pageRect: Rect; pageNum:
Integer): Boolean): OSErr;
    type
        TPrDevice = (MacScreen, ImageWriter, Unknown, LaserWriter);

    var
        scratch: Boolean;

        wasNIL: Boolean;    {TRUE if we needed to allocate hPrint}

        curPrError: OSErr;    {The last printing error}
        PrIsOpen: Boolean;    {TRUE if we're in the middle of a PrOpen/PrClose pair}
        DocIsOpen: Boolean;   {TRUE if we're in a PrOpenDoc/PrCloseDoc pair}
        PageIsOpen: Boolean;  {TRUE if we're in a PrOpenPage/PrClosePage pair}

        nCopies: Integer;    {Number of copies to be printed.}
        prDevice: TPrDevice; {The kind of printer we're printing on}
        draftMode: Boolean;  {Draft or spool?}
```

USENET Macintosh Programmer's Guide

```
prPort: TPrPort;
prStatus: TPrStatus;

savePort: GrafPtr;

i, p: Integer;

procedure CleanUp; {This procedure cleans up after the routine and exits.}
begin
    if wasNil and (hPrint <> nil) then
        DisposHandle(Handle(hPrint));

    if PageIsOpen then
        PrClosePage(prPort);

    if DocIsOpen then
        PrCloseDoc(prPort);

    if PrIsOpen then
        PrClose;

    SetPort(savePort);

    GenericPrint := curPrError;
    Exit(GenericPrint);
end;

procedure CheckPrError;
begin
    if curPrError <> noErr then
        CleanUp;
end;

begin
    GenericPrint := noErr;
    PrIsOpen := False;
    DocIsOpen := False;
    PageIsOpen := False;

    GetPort(savePort);

    wasNil := (hPrint = nil);

    PrOpen;
    curPrError := PrError;
    CheckPrError;
    PrIsOpen := True;

    {If we need to, allocate the print handle.  If the allocate failed, exit out}
    {with a memFullErr result.}
    if wasNil then
        begin
            hPrint := THPrint(NewHandle(SizeOf(TPrint)));
            if hPrint = nil then
                begin
                    curPrError := memFullErr;
                    CleanUp;
                end;
            {Initialize the print handle}
            PrintDefault(hPrint);
        end;

    {Validate the print handle.}
```

```
scratch := PrValidate(hPrint);  
if not PrJobDialog(hPrint) then
```

USENET Macintosh Programmer's Guide

```
        CleanUp;

{Call the user's prep proc, and exit if it returns false.}
    if not UserPrepProc(hPrint) then
        CleanUp;

{We need to determine some parameters to print correctly.}

    {Figure out which printer we're using}
    prDevice := TPrDevice(BSR(hPrint^.prStl.wDev, 8));

    {Determine whether we're in draft mode or not.}
    draftMode := not Odd(hPrint^.prJob.bJDocLoop);

    {If we're in draft mode on an ImageWriter, we have to honor the number of}
    {copies; otherwise, the printer driver takes care of it for us.}
    if draftMode and (prDevice = ImageWriter) then
        nCopies := hPrint^.prJob.iCopies
    else
        nCopies := 1;

{Now we're ready to begin the printing loop.}

    {Open the print document. If it fails, leave.}
    prPort := PrOpenDoc(hPrint, nil, nil);
    docIsOpen := True;
    curPrError := PrError;
    CheckPrError;

    SetPort(@prPort^.gPort);
    for i := 1 to nCopies do
        begin
            for p := hPrint^.prJob.iFstPage to hPrint^.prJob.iLstPage do
                begin
                    thePage := p;
                    PrOpenPage(prPort, nil);
                    PageIsOpen := True;
                    scratch := UserPageProc(hPrint, hPrint^.prInfo.rPage, p);
                    PrClosePage(prPort);
                    if not scratch then
                        begin
                            curPrError := iPrAbort;
                            CleanUp;
                        end;
                end;
            end;

        PrCloseDoc(prPort);
        DocIsOpen := False;
        curPrError := PrError;

        if (not DraftMode) and (curPrError = noErr) then
            begin
                PrPicFile(hPrint, nil, nil, nil, prStatus);
                curPrError := PrError;
            end;
        end;
    end.
```