Technical Note TN1092

A Printing Loop That Cares - The Sequel

CONTENTS

This Technote, originally *Technote PR 10 - A Printing Loop That Cares* , discusses how and why your application should add a generic printing loop in order to be compatible with today's printer drivers.

This revised Technote reflects the current Macintosh Printing Manager and discusses proper opening and closing of the Macintosh Printing Manager with calls to `PrOpen` and `PrClose`. It also shows how your application should handle errors at print time and lists the latest error codes.

Updated: [Feb 11 1997]

The Old Way of Handling Printing

In the past (pre-System 7), Apple recommended that developers call `PrOpen` at the beginning of your application and `PrClose` at the end before returning to the Finder. This recommendation was appropriate when your application only had to deal with a single printer driver. However, as more printers became available on the market, it became important that your application took into account the presence of other launched applications and multiple printer drivers.

For instance, the user could open the Chooser at any time and change the current printer driver without the current application's knowledge. If an application followed the old philosophy and a user changed the current printer driver while running the application, the next time the user attempted to print, the wrong driver would be open, the Printing Manager would not be able to find the necessary resources, and the user would get an error.

The original Technote described a method of printing that allowed applications to circumvent all of these problems; this revised Note shows you an even better method.

Back to top

The New Way: a C Print Loop

The following code snippet, `PrintStuff`, represents a simple print loop that your application should use to print. It works as follows:

1. It calls all of the necessary Print Manager calls to print a document.
2. It checks `PrError` after each Print Manager call.
3. If an error is found, all of the Print Manager open calls (i.e., `PrOpen`, `PrOpenDoc`...) have a corresponding close call before posting an error.

You should use the error-checking method in Step #3 to make sure the Print Manager closes properly and that all temporary memory is released.

> **Note:**
> Apple Developer Technical Support currently recommends that applications open and close the printer driver each time your application uses the Printing Manager. We also highly recommend appropriate error checking, as demonstrated in this snippet of code.

The PrintStuff Print Loop

```
void PrintStuff ()
{
  GrafPtr      oldPort;
  short        copies,
               firstPage,
               lastPage,
               numberOfCopies,
               printmgrsResFile,
               realNumberOfPagesinDoc,
               pageNumber,
```

```
               PrintError;
  THPrint      thePrRecHdl;
  TPPrPort     thePrPort;
  TPrStatus    theStatus;

  GetPort(&oldPort);

thePrRecHdl = (THPrint)  NewHandle (sizeof (TPrint));

 /**
     Check to make sure that the memory manager did not produce an error
     when it allocated the print record handle and make sure it did not pass
     back a nil handle.
 **/

  if (thePrRecHdl != NULL && MemError() == noErr)
    {
      PrOpen();

      if (PrError() == noErr)
        {
         /** Save the current resource file (i.e., the printer driver's) so
          the driver will not lose its resources upon return from the pIdleProc.
          **/
          printmgrsResFile = CurResFile();
          PrintDefault(thePrRecHdl);

          if (PrError() == noErr)
            {
              if (PrStlDialog(thePrRecHdl))
                {
                 /**
                    DetermineNumberOfPagesinDoc determines the number of pages
                    contained in the document by comparing the size of the
                    document with rPage from the TPrInfo record (Inside
                    Macintosh: Imaging With QuickDraw p.9-46).
                    It returns the number of pages required to print the
                    document for the currently selected printer.
                  **/

                  realNumberOfPagesinDoc = DetermineNumberOfPagesinDoc
                                              ((**thePrRecHdl).prInfo.rPage);

                  if (PrJobDialog(thePrRecHdl))
                    {
                     /**
                            Get the number of copies of the document that the
                            user wants printed from iCopies of the TPrJob record
                            (Inside Macintosh: Imaging With QuickDraw p.9-47).
                       **/

                       numberOfCopies = (**thePrRecHdl).prJob.iCopies;

                        /**
                            Get the first and last pages of the document that
                            were requested to be printed by the user from FstPage
                            and iLastPage from the TPrJob record (Inside
                            Macintosh: Imaging With QuickDraw p.9-47).
                         **/

                       firstPage = (**thePrRecHdl).prJob.iFstPage;
                       lastPage = (**thePrRecHdl).prJob.iLstPage;

                        /**
                            Print "all" pages in the print loop
                         **/

                       (**thePrRecHdl).prJob.iFstPage = 1;
                       (**thePrRecHdl).prJob.iLstPage = 9999;

                        /**
                            Determine the "real" number of pages contained in the
                            document. Without this test, you would print 9999 pages.
                         **/

                       if (realNumberOfPagesinDoc < lastPage)
                         lastPage = realNumberOfPagesinDoc;
```

```
 PrintingStatusDialog = GetNewDialog(rPrintingDialogID,
    nil, (WindowPtr) -1);

  /**
      Print the number of copies of the document
      requested by the user from the Print Job Dialog.
   **/
 for (copies = 1; copies <= numberOfCopies; copies++)
   {
    /**
   Install a pointer to your pIdle proc in my print record.
     **/
     (**thePrRecHdl).prJob.pIdleProc = checkMyPrintDialogButton();
    /**
       Restore the resource file to the printer driver's.
     **/

     UseResFile(printmgrsResFile);
     thePrPort = PrOpenDoc(thePrRecHdl, nil, nil);

     if (PrError() == noErr)
       {

        /**
            Print the range of pages of the document
            requested by the user from the Print Job Dialog.
         **/
         pageNumber = firstPage;
         while (pageNumber <= lastPage && PrError() == noErr)
           {

/**
  If we've crossed a 128-page boundary,
  close the current print file, send it
  to the printer if necessary, and open a
  new document.
**/

if ((pageNumber - firstPage) % iPFMaxPgs == 0)
  {
    if (pageNumber != firstPage)
      {
        PrCloseDoc(thePrPort);
        if (((**thePrRecHdl).prJob.bJDocLoop ==
                       bSpoolLoop) && (PrError() == noErr))
                       PrPicFile(thePrRecHdl, nil, nil, nil,
                                  &theStatus);
        thePrPort = PrOpenDoc(thePrRecHdl, nil,
                                           nil);
                   }
               }

PrOpenPage(thePrPort, nil);

             if (PrError() == noErr)
               {
                /**
                   rPage (Inside Macintosh: Imaging With
                   QuickDraw p.9-46) is the printable area
                   for the currently selected printer. By
                   passing the current port to the draw
                   routine, enables your app to use the
                   same routine to draw to the screen and
                   the printer's GrafPort.
                 **/
                 DrawStuff ((**thePrRecHdl).prInfo.rPage,
                      (GrafPtr) thePrPort, pageNumber);
               }

             PrClosePage(thePrPort);
             pageNumber++;
           }  /**  End pageNumber loop  **/
        }
     PrCloseDoc(thePrPort);
   } /**  End copies loop  **/
}
```

```
                    /**
                        The printing job is being canceled by the request of the
                        user from the Print Style Dialog or the Print Job Dialog.
                        PrError will be set to PrAbort to tell the Print Manager
                        to abort the current printing job.
                     **/
                   else
                     PrSetError (iPrAbort);   /**  cancel from the job dialog  **/
                 }
               else
                 PrSetError (iPrAbort);   /**  cancel from the style dialog  **/
             }
         }

      if (((**thePrRecHdl).prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
        PrPicFile(thePrRecHdl, nil, nil, nil, &theStatus);

       /**
            Grab the printing error -- once you close the Printing Manager,
            PrError doesn't return a valid result anymore.
       **/

      PrintError = PrError();

      PrClose();

       /**
           You do not want to report any printing errors until you have fallen
           through the printing loop. This will make sure that ALL of the Print
           Manager's open calls have their corresponding close calls, thereby
           enabling the Print Manager to close properly and that all temporary
           memory allocations are released.
        **/
      if (PrintError != noErr)
        PostPrintingErrors (PrintError);
    }

  if (thePrRecHdl != NULL)
    DisposeHandle((Handle) thePrRecHdl);

  if (PrintingStatusDialog != NULL)
    DisposeDialog(PrintingStatusDialog);

  SetPort(oldPort);
}  /** PrintStuff **/
```

Back to top

Checking For Error Conditions While Printing

Your application should always check for error conditions while printing. You can do this by calling `PrError`. `PrError` returns errors from the Printing Manager (and some AppleTalk and OS errors) that may occur during printing.

As the previous example code demonstrates, your application should call `PrError` after each call to a Printing Manager function or procedure. By consistently checking `PrError` after each call, your application will be able to catch any errors created at print time and be able to report them to your user via a dialog box.

Some General Error-Handling Guidelines

The following section provides you with some general error-handling guidelines:

- Don't call `PrError` within your `pIdle` procedure; errors that occur while it is executing are usually temporary and serve only as internal flags for communication within the printer driver -- they are not intended for the application. If you discover that you need to abort printing while in your idle procedure, set a flag to signal yourself, and check your flag after each Printing Manager function. If the flag is set, you can exit in the same manner as if an error occurred.
- On detecting an error after the completion of a printing routine, stop drawing at that point, and proceed to the next procedure to close any previously made open calls. For example, if you detect an error after calling `PrOpenDoc`, skip to the next `PrCloseDoc`. Or, if you get an error after calling `PrOpenPage`, skip to the next `PrClosePage` and `PrCloseDoc`. Remember that if you have called `PrOpen`, then you must call the corresponding `PrClose` to ensure that printing closes properly and that all temporary memory allocations are released and returned to the heap.
- Don't display any alert or dialog boxes to report an error until the end of the printing loop. Once at the end, check for the error again; if there is no error, assume that printing completed normally. If the error is still present, alert the user.

  This procedure -- not displaying any alerts or dialog boxes -- is important for two reasons.

  1. If you display a dialog box in the middle of the printing loop, it could cause errors that can terminate an

otherwise normal job. For example, if the printer is an AppleTalk printer, the connection can be terminated abnormally, since the driver would be unable to respond to AppleTalk requests received from the printer while the dialog box was waiting for input from the user. If the printer does not hear from the Macintosh within a certain period of time (currently 300-600 seconds), it times out, assuming that the Macintosh is no longer there, which results in a prematurely broken connection, causing another error to which the application must respond.

2. The driver may have already displayed its own dialog box in response to an error. In this instance, the driver posts an error to let the application "know" that something went wrong and it should abort printing. For example, in older LaserWriter drivers, when the driver detected that the Laser Prep version which was being downloaded to the LaserWriter was different than the version the user was trying to print with, it displayed the appropriate dialog box informing the user of the situation and gave him or her the option of reinitializing the printer. If the user chose to cancel printing, the driver posted an error to let the application "know" that it needed to abort, but since the driver had already taken care of the error by displaying a dialog box, the error was reset to zero before the printing loop was complete. Your application should check for the error again at the end of the printing loop, and if it still indicates an error, your application can then display the appropriate dialog box.

- If you're using `PrGeneral`, be prepared to receive the following errors: `NoSuchRsl`, `OpNotImpl`, and `resNotFound`. In all three cases, your application should be prepared to continue printing without using the features of that particular opcode.

  In the case of the `resNotFound` error, however, it means the current printer driver does not support `PrGeneral`. This lack of support should not be a problem for your application, but your application needs to be prepared to deal with this error. If you receive a `resNotFound` error from `PrError`, clear the error with a call to `PrSetError(0)`; otherwise, `PrError` might still contain this error the next time you check it, which would prevent your application from printing.

Cancelling or Pausing the Printing Process

If you install a procedure for handling requests to cancel printing, with an option to pause the printing process, beware of timeout problems when printing to network printers. Communication between the Macintosh and a networked printer must be maintained to prevent a job or a wait timeout. If there is no communication for a period of time (roughly 300-600 seconds), the printer times out and the print job terminates due to a wait timeout. Or, if the print job requires more than ten minutes to print, the print job terminates due to a job timeout. Since there is no good method to determine to what type of printer an application is printing, it is probably a good idea to document in your ReadMe the possibility of a network printer timing out for a user who chooses to select "pause" for five minutes or more.

Back to top

Error Messages

The Printing Manager reports the error messages covered in this section. If an error that does not belong to the Printing Manager occurs, the Printing Manager puts it into low memory, where it can be retrieved with a call to `PrError`, and terminates the printing loop, if necessary. As already documented, if you encounter an error in the middle of a printing loop, don't jump out; fall through the loop and let the Printing Manager terminate properly.

The most common error encountered is **-4101**, which is generated if the selected LaserWriter is not available on the network. Since this error is so common, it's a good idea to display a dialog box requesting the user to select a printer from the Chooser when this error is encountered.

Common Printing Manager and System Errors

The following table shows you common printing manager and system error codes.

Error Code	Constant	Description
0	`noErr`	No error
28	[don't know]	Stack/heap collision. Too much stack usage inside QuickDraw [not uncommon if you're calling DrawPicture on QT compressed pictures]
128	`iPrAbort`	Abort the printing process (result of Command-period)
-1	`iPrSavePFil`	Problem saving print file
-17	`controlErr`	Unimplemented Control call
-27	`iIOAbort`	I/O problems
-108	`iMemFullErr`	Not enough heap space

`PrGeneral` Errors

`PrGeneral` is declared like this in C:

```
pascal void PrGeneral (Ptr pData);
```

The `pData` parameter is a pointer to a record called `TGnlData`. The first eight bytes comprise a header shared by all the `PrGeneral` calls:

```
struct TGnlData     {
           short iOpCode;
           short iError;
           long lReserved;
           };
```

After each call to PrGeneral, your application should check the value in the iError field. The possible result codes that can be returned are:

Error Code	Constant	Description
0	noErr	No Error
1	NoSuchRsl	Unsupported Resolution
2	OpNotImpl	Unsupported Opcode
-192	resNotfound	The current printer driver does not support PrGeneral.

For further information on PrGeneral, you should read Pete 'Luke' Alexander's article, "Meet PrGeneral," in *develop 3* .

LaserWriter Driver Family Errors

Error Code	Description
-4101	Printer not found or closed.
-4100	Connection just closed.
-4099	Write request too big.
-4098	Request already active.
-4097	Bad connection refnum.
-4096	No free Connect Control Blocks (CCBs) available.

LaserWriter 8 Internal Errors

Note:
The following error codes are internal LaserWriter 8 errors. They are useful for debugging, but your application should NOT try to interpret or use these error codes during runtime.

-8998	errNotAKey	Couldn't find a key for the desired font number.
-8997	errFaceListBad	(NO LONGER USED)
-8996	errSizeListBad	The size list was not consistent with the face list.
-8995	errFontNotFound	A font query reply didn't match any of the PostScript fonts.
-8994	errUnknownPSLevel	We asked for the printer's PostScript level and got an answer we didn't expect.
-8993	errInLineTimeout	We got tired of waiting for a response from the printer.
-8991	errNoProcSetRes	While generating PostScript prolog, we couldn't find the resource containing the needed procedure sets.
-8990	errBadSpoolFileVersion	While foreground printing (pre-LW8.4) we read the spool file, and the header information was not good.
-8989	errCouldNotMakeNumberedFilename	Couldn't make a unique spool file name by adding numbers to the base name. We ran out of numbers.
-8987	errPSFileName	While saving PS to disk, the filename was bad.
-8986	errBitmapFontMissing	We tried to build a 1-bit bitmap, but failed.
-8985	errDidNotDownloadFont	The PS outline couldn't be found, and there's no 'sfnt'.
-8984	errBadConverterIndex	Couldn't find the entry matching the selection in the "Save to Disk" popup.
-8983	errSpoolFolderIsAFile	(NO LONGER USED)
-8982	errPSFileNameNull	(NO LONGER USED)
-8981	errNullColorInfo	GetColor was called with a NULL handle.
-8980	errNoPagesSpooled	The app made a PrOpenDoc call and PrCloseDoc, but didn't print any pages.
-8979	errBadConverterID	The PDEF we wanted to run as a converter wasn't there.
-8978	errNoPattern	We couldn't find or make a pixpat.
-8977	errPSStateUnderFlow	We tried to pop the topmost graphics state. Oops!
-8976	errChannelNotBinary	Application wants binary data (via PrGeneral), but the actual channel to the printer isn't binary.
-8975	errPrinterNotLevel2	Application wants to use Level 2 PS, but the printer's not hip to Level 2.
-8974	errBadFontKeyType	The type of a font was not PS, TT or bitmap.
-8973	errFunctionNotAvailable	(NO LONGER USED)
-8972	errNULLFormatString	The format string passed to an internal printf-like function was null.
-8971	errNotAFolderAlias	The alias that should point to the "Print Monitor Documents" folder isn't pointing to a folder.

-8970	errMissingPAPA	The PAPA resource we looked for isn't there.
-8969	errMissingPrinterInfo	The current printer does not have an entry in the printer database - usually because it hasn't been set up.
-8968	errUnsupportedDestColorMode	The output colorspace isn't supported.
-8967	errUnknownColorUsage	(NO LONGER USED)
-8966	errUnsupportedCodec	Compressed pixmap wants a codec we can't deal with.
-8965	errInvalidPPD	Tried to open the PPD and couldn't.
-8964	errBadColorSync2Comment	The ColorSync2 PicComment wants a 4-byte selector and we encountered a datasize < 4 bytes.
-8963	errBadFlattenRefCon	ColorSync gave us a NULL refcon in the flatten proc.
-8962	errGlyphsDontFit	A single glyph either didn't end on a 4-byte boundary (a bug in the font) or was greater than 64k.
-8961	errGenericComponentErr	Generic error
-8960	errUnsupportedStream	The PSStream type passed in to a given library call is not supported.
-8959	errProfileNotInList	The internal temporary profile list went bad.
-8958	errUninitializedPort	Uninitialized port
-8957	errHintWrongSize	One of the converter's hints was an unexpected size.
-8956	errSystemProfileNotFound	We tried to use ColorSync, but couldn't find the default System Profile.
-8955	errCFM_EnablerNotPresent	We're trying to use CFM-68K, but the enabler's not there.
-8954	errCouldNotIDArchitecture	
-8953	errPSStreamNullOutProc	Got a bad function pointer for the output routine.
-8952	errTriedToWriteNullBuffer	This should never happen.
-8951	errWhoTookThatOutBuffer	We had a buffer that's gone now. This seems bad.
-8950	errMoreDataToFlush	There's still data to be dealt with.
-8160	zoomRangeErr	
-8152	noPrepErr	
-8151	prepMismatchErr	
-8150	noChosenPrinterErr	
-8133	generalPSErr	PostScript error during transmission of data to printer. Most often caused by a bug in the PostScript code being downloaded.
-8132	manualFeedTOErr	Timeout occurred.
-8131	Printer not responding.	

Back to top

Summary

That's all there is to it. Now your application can print properly with the Macintosh Printing Manager by adhering to the rules specified in this Note and by handling error messages appropriately.

Back to top

References

Inside Macintosh: Imaging With QuickDraw, Chapter 9

StdFileSaver sample code, available on the Developer CD Series: Tool Chest Edition.

Technote PR02: Device-Independent Printing by Ginger Jernigan.

develop 3: "Meet PrGeneral, the Trap That Makes the Most of the Printing Manager" by Pete "Luke" Alexander.

develop 27: "Print Hints: The All-New LaserWriter Driver Version 8.4" by Dave Polaschek.

Back to top

Change History

01-October-1990	Originally written in October 1990, as *Technote PR10 -- A Printing Loop That Cares...*
01-January-1994	Revised as *Technote PR10 -- A Printing Loop That Cares...* with new text and code.
11-February-1997	Technote updated to reflect the current Macintosh Printing Manager and to use C code. The Pascal code was removed. Updated error codes were also added to the Error Messages section.

Back to top

Downloadables

Acrobat version of this Note (K) Download

Back to top
