New Technical Notes
Macintosh

®

Developer Support

PrGeneral
Imaging

Revised by: March 1988
Written by: Ginger Jenigan May 1987

The Printing Manager architecture has been expanded to include a new procedure called `PrGeneral`. The features described here are advanced, special-purpose features, intended to solve specific problems for those applications that need them. The calls to determine printer resolution introduce a good deal of complexity into the application's code, and should be used only when necessary.

Version 2.5 (and later) of the ImageWriter driver and version 4.0 (and later) of the LaserWriter driver implement a generic Printing Manager procedure called `PrGeneral`. This procedure allows the Print Manager to expand in functionality, by allowing printer drivers to implement various new functions. The Pascal declaration of `PrGeneral` is:

```
PROCEDURE PrGeneral (pData: Ptr);
```

The `pData` parameter is a pointer to a data block. The structure of the data block is declared as follows:

```
TGnlData = RECORD {1st 8 bytes are common for all PrGeneral calls)
   iOpCode   : INTEGER;  {input}
   iError    : INTEGER;  {output}
   lReserved : LONGINT;  {reserved for future use}
   {more fields here, depending on particular call}
END;
```

The first field is a 2-byte opcode, `iOpCode`, which acts like a routine selector. The currently available opcodes are described below.

The second field is the error result, `iError`, which is returned by the print code. This error only reflects error conditions that occur during the `PrGeneral` call. For example, if you use an opcode that isn't implemented in a particular printer driver then you will get a `OpNotImpl` error.

Here are the errors currently defined:

```
CONST
   noErr = 0;              {everything's hunky}
   NoSuchRsl = 1;          {the resolution you chose isn't available}
   OpNotImpl = 2;          {the driver doesn't support this opcode}
```

After calling `PrGeneral` you should always check `PrError`. If `noErr` is returned, then you can proceed. If `ResNotFound` is returned, then the current printer driver doesn't support `PrGeneral` and you should proceed appropriately. See Technical Note #118 for details on checking errors returned by the Printing Manager.

`IError` is followed by a four byte reserved field (that means don't use it). The contents of the rest of the data block depends on the opcode that the application uses. There are currently five opcodes used by the ImageWriter and LaserWriter drivers.

The Opcodes

Initially, the following calls are implemented via `PrGeneral`:

- `GetRslData` (get resolution data): `iOpCode = 4`
- `SetRsl` (set resolution): `iOpCode = 5`
- `DraftBits` (bitmaps in draft mode): `iOpCode = 6`
- `noDraftBits` (no bitmaps in draft mode): `iOpCode = 7`
- `GetRotn` (get rotation): `iOpCode = 8`

The `GetRslData` and `SetRsl` allow the application to find out what physical resolutions the printer supports, and then specify a supported resolution. `DraftBits` and `noDraftBits` invoke a new feature of the ImageWriter, allowing bitmaps (imaged via `CopyBits`) to be printed in draft mode. `GetRotn` lets an application know whether landscape has been selected. Below is a detailed description of how each routine works.

The GetRslData Call

`GetRslData (iOpCode = 4)` returns a record that lets the application know what resolutions are supported by the current printer. The application can then use `SetRsl` (description follows) to tell the printer driver which one it will use. This is the format of the input data block for the `GetRslData` call:

```
TRslRg = RECORD     {used in TGetRslBlk}
      {0 if printer only supports discrete resolutions}
      iMin, iMax:   Integer;
END;

TRslRec = RECORD    {used in TGetRslBlk}
      iXRsl, iYRsl: Integer; {a discrete, physical resolution}
END;

TGetRslBlk = RECORD {data block for GetRslData call}
      iOpCode:      Integer;      {input; = getRslDataOp}
      iError:       Integer;      {output}
      lReserved:    LongInt;      {reserved for future use}
      iRgType:      Integer;      {output; version number}
      XRslRg:       TRslRg;       {output; range of X resolutions}
      YRslRg:       TRslRg;       {output; range of Y resolutions}
      iRslRecCnt:   Integer;      {output; how many RslRecs follow}
      rgRslRec:     ARRAY[1..27] OF TRslRec;  {output; number filled
depends on
                                                    printer type}
       END;
```

The `iRgType` field is much like a version number; it determines the interpretation of the data that follows. At present, a `iRgType` value of 1 applies both to the LaserWriter and to the ImageWriter.

For variable-resolution printers like the LaserWriter, the resolution range fields `XRslRg` and `YRslRg` express the ranges of values to which the X and Y resolutions can be set. For discrete-resolution printers like the ImageWriter, the values in the resolution range fields are zero.

Note: In general, X and Y in these records are the horizontal and vertical directions of the **printer**, not the document! In landscape orientation, X is horizontal on the printer but vertical on the document.

After the resolution range information there is a word which gives the number of resolution records that contain information. These records indicate the physical resolutions at which the printer can actually print dots. Each resolution record gives an X value and a Y value.

When you call `PrGeneral` you pass in a data block that looks like this:

OpCode = 4	1 word
Error Code	1 word
Reserved	2 words
RangeType = 1	1 word
X Resolution Range: min = 0, max = 0	2 words
Y Resolution Range: min =0, max = 0	2 words
Resolution Record Count =0	1 word
Resolution Record #1: X = 0, Y = 0	2 words
Resolution Record #2..27	

Below is the data block returned for the LaserWriter:

OpCode = 4	1 word
Error Code (0 = okay)	1 word
Reserved	2 words
RangeType = 1	1 word
X Resolution Range: min = 72, max = 1500	2 words
Y Resolution Range: min = 72, max = 1500	2 words
Resolution Record Count = 1	1 word
Resolution Record #1: X = 300, Y = 300	2 words

Note that all the resolution range numbers happen to be the same for this printer. There is only one resolution record, which gives the physical X and Y resolutions of the printer (300x300).

Below is the data block returned for the ImageWriter.

OpCode = 4	1 word
Error Code (0 = okay)	1 word
Reserved	2 words
RangeType = 1	1 word
X Resolution Range: min =0, max = 0	2 words
Y Resolution Range: min = 0, max = 0	2 words
Resolution Record Count = 4	1 word
Resolution Record #1: X = 72, Y = 72	2 words
Resolution Record #2: X =144, Y = 144	2 words
Resolution Record #3: X = 80, Y = 72	2 words
Resolution Record #4: X = 160, Y = 144	2 words

All the resolution range values are zero, because only discrete resolutions can be specified for this printer. There are four resolution records giving these discrete physical resolutions.

Note that `GetRslData` always returns the same information for a particular printer type—it is **not** dependent on what the user does or on printer configuration information.

The SetRsl Call

`SetRsl` (`iOpCode` = 5) is used to specify the desired imaging resolution, after using `GetRslData` to determine a workable pair of values. Below is the format of the data block:

```
TSetRslBlk = RECORD {data block for SetRsl call}
      iOpCode:      Integer;      {input; = setRslOp}
      iError:       Integer;      {output}
      lReserved:    LongInt;      {reserved for future use}
      hPrint:       THPrint;      {input; handle to a valid print record}
      iXRsl: Integer;      {input; desired X resolution}
      iYRsl: Integer;      {input; desired Y resolution}
END;
```

`hPrint` should be the handle of a print record that has previously been passed to `PrValidate`. If the call executes successfully, the print record is updated with the new resolution; the data block comes back with 0 for the error and is otherwise unchanged.

However, if the desired resolution is not supported, the error is set to `noSuchRsl` and the resolution fields are set to the printer's default resolution

Note that you can undo the effect of a previous call to `SetRsl` by making another call that specifies an unsupported resolution (such as 0x0), forcing the default resolution.

The DraftBits Call

`DraftBits` (`iOpCode` = 6) is implemented on both the ImageWriter and the LaserWriter. (On the LaserWriter it does nothing, since the LaserWriter is always in draft mode and can always print bitmaps.) Below is the format of the data block:

```
TDftBitsBlk =RECORD {data block for DraftBits and NoDraftBits calls}
      iOpCode:      Integer;      {input; = draftBitsOp or noDraftBitsOp}
      iError:       Integer;      {output}
      lReserved:    LongInt;      {reserved for future use}
      hPrint:       THPrint;      {input; handle to a valid print record}
END;
```

`hPrint` should be the handle of a print record that has previously been passed to `PrValidate`.

This call forces draft-mode (i.e., immediate) printing, and will allow bitmaps to be printed via `CopyBits` calls. The virtue of this is that you avoid spooling large masses of bitmap data onto the disk, and you also get better performance.

The following restrictions apply:

- This call should be made before bringing up the print dialogs because it affects their appearance. On the ImageWriter, calling `DraftBits` disables the landscape icon in the Style dialog, and the Best, Faster, and Draft buttons in the Job dialog.

- If the printer does not support draft mode, already prints bitmaps in draft mode, or does not print bitmaps at all, this call does nothing.
- Only text and bitmaps can be printed.
- As in the normal draft mode, landscape format is not allowed.
- Everything on the page must be strictly Y-sorted, i.e. no reverse paper motion between one string or bitmap and the next. Note that this means you can't have two or more objects (text or bitmaps) side by side; the top boundary of each object must be no higher than the bottom of the preceding object.

The last restriction is important. If you violate it, you will not like the results. But note that if you want two or more bitmaps side by side, you can combine them into one before calling `CopyBits` to print the result. Similarly, if you are just printing bitmaps you can rotate them yourself to achieve landscape printing.

The NoDraftBits Call

`NoDraftBits` (`iOpCode = 7`) is implemented on both the ImageWriter and the LaserWriter. (On the LaserWriter it does nothing, since the LaserWriter is always in draft mode and can always print bitmaps.) The format of the data block is the same as that for the `DraftBits` call.

This call cancels the effect of any preceding `DraftBits` call. If there was no preceding `DraftBits` call, or the printer does not support draft-mode printing anyway, this call does nothing.

The GetRotn Call

`GetRotn` (`iOpCode` = 8) is implemented on the ImageWriter and LaserWriter. Here is the format of the data block:

```
TGetRotnBlk = RECORD {data block for GetRotn call}
        iOpCode:     Integer;     {input; = getRotnOp}
        iError:      Integer;     {output}
        lReserved:   LongInt;     {reserved for future use}
        hPrint:      THPrint;     {input; handle to a valid print record}
        fLandscape:  Boolean;     {output; Boolean flag}
        bXtra: SignedByte;        {reserved}
END;
```

`hPrint` should be the handle to a print record that has previously been passed to `PrValidate`.

If landscape orientation is selected in the print record, then `fLandscape` is true.

How To Use The PrGeneral Opcodes

The `SetRsl` and `DraftBits` calls may require the print code to suppress certain options in the Style and/or Job dialogs, therefore they should always be called before any call to the Style or Job dialogs. An application might use these calls as follows:

- Get a new print record by calling `PrintDefault`, or take an existing one from a document and call `PrValidate` on it.

- Call `GetRslData` to find out what the printer is capable of, and decide what resolution to use. Check `PrError` to be sure the `PrGeneral` call is supported on this version of the print code; if the error is `ResNotFound`, you have older print code and must print accordingly. But if the PrError return is 0, proceed:

- Call `SetRsl` with the print record and the desired resolution if you wish.

- Call `DraftBits` to invoke the printing of bitmaps in draft mode if you wish.

Note that if you call either `SetRsl` or `DraftBits`, you should do so before the user sees either of the printing dialogs.

Further Reference:

- The Printing Manager