# T E C H N O T E :
# Printer Direct Mode APIs for Macintosh Printer Drivers

By Mike Blackstock, GDT Softworks, Inc. & Ben Manuto, Apple Engineering
**mike_blackstock@gdt.com**
**manuto@apple.com**

This Technote describes the APIs for implementing a printer-direct (or pass-thru) mode for Macintosh printer drivers. Implementing this feature will allow applications to identify and send printer data that is unique to the printer connected to the Macintosh without having to generate QuickDraw codes, understand how to connect to and maintain a connection with a particular printer, or handle communications errors with the printer.

Apple's PC Compatibility systems present an interesting dilemma when trying to print from the DOS/Windows side of the CPU because all of the I/O of this system must occur through the Macintosh. DOS applications and Windows printer drivers generate native printer codes for specific printers, but do not generate the QuickDraw commands that Macintosh printer drivers depend on in order to image data to a specific printer. This printer direct mode will allow a "back-door" for these native printer commands to get directly to the printer through the Macintosh printer driver.

This Technote is directed towards printer driver developers who write drivers for non-PostScript printers. Implementing this feature will allow faster and better quality printing with Apple's PC Compatibility products, where printing must always occur on the Macintosh, even though the printer driver generating the printer commands may be on a DOS/Windows system.

This document assumes you are familiar with the Macintosh Printing Manager and Apple's PC Compatibility products. See *Inside Macintosh: Imaging with QuickDraw* for further details on the Printing Manager.

## About the Printer Direct Mode

The printer direct (or pass-thru) mode is specifically intended to allow printer manufacturers and printer driver developers to make products that will easily integrate into the PC Compatibility CPU marketplace. Since most new printers are not being developed for a single CPU platform, the printer direct mode will give end-users the best printing results possible.

## Using the Printer Direct APIs

The printer direct printer driver functions are implemented via a set of `PrGeneral` routines and a reserved `opCode` for those routines. There is a set of six functions (for which there are selectors) that the Macintosh printer driver must support: `Open`, `Close`, `SendData`, `SendFile`, `Despool`, and `Verify`. The `PrGeneral` `opCode` reserved by Apple for this operation is 20 decimal ($14 hex). Each of the five functions take a selector code as defined by these constants:

```
#define kPrinterDirectOpCode    20

#define kPDOpenSelect           1
#define kPDSendDataSelect       2
#define kPDSendFileSelect       5
#define kPDCloseSelect          3
#define kPDDespoolSelect        4
#define kPDVerifySelect         6
```

If the driver does not support the printer direct `opCode`, the `PrGeneral` function should return a `opNotImpl` error. This is the one case where an error code will only be returned in the printer error (`PrError`). For all the rest of the functions, error codes need to be returned in the error field of the parameter block as well as the printing error variable (`PrSetError`).

## Open Printer Direct

The `Open` command alerts the printer driver that data is about to be sent and it should allocate whatever memory structures are needed to begin receiving the printer data. This printer data can be passed to the driver in one of two ways: 1) As a pointer to data blocks of a specific size, or 2) an `FSSpec` record which points to a spool file where the data is located. The appropriate selector code for `Open` command and an appropriate selector for the type of data to be sent to the driver must entered into the selector and `spoolType` fields of the `OpenPDBlk`. The `spoolType` valid values are:

```
#define kPDData 1
#define kPDFileSpec2
```

If any other value is set in the `spoolType` field, the function should return the `pdBadSpoolTypeErr`. The format of the parameter block is:

```
struct OpenPDBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    UInt32 spoolType;   // Ptr to data or FSSpec.
};
typedef struct OpenPDBlk OpenPDBlk, *OpenPDBlkPtr;
```

**ERRORS**

```
opNotImpl,
pdBadSelectorErr,
pdBadJobIDErr,
pdBadSpoolTypeErr
```

The `opCode` is always 20 (`kPrinterDirectOpCode`). The error and reserved fields should be initialized to zero. The `jobID` is returned from the `Open` function and it is the calling function's unique identifier to the print job being opened.

## Send Data

There are two `Send` commands for sending the print job to the printer: `SendPDData` and `SendFileSpec`. For sending data, the driver must minimally be able to handle 4K data blocks which are repeatedly sent to the printer until all the data has been sent. However, more data can be sent if the driver can handle additional memory requirements. If the driver is not able to handle any data block larger than 4K, an error must be appropriately returned by the printer driver.

```
struct SendPDDataBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    Ptr Data;           // Pointer to the data.
    UInt32 count;       // Number of bytes Data points to.
};
typedef struct SendPDDataBlk SendPDDataBlk, *SendPDDataBlkPtr;
```

**ERRORS**

```
opNotImpl,
pdBadSelectorErr,
pdBadJobIDErr,
Memory Manager Errors
```

## Send File

The `Send File` command is issued once and specifies a `fileSpec` for the location of the spooler file containing all of the print job data. This file must be in a closed state when it is passed to the driver. The printer driver is responsible for deleting the spool file when it is finished spooling the file to the printer.

```
struct SendPDFileSpecBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
```

```
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    FSSpec fileSpec;    // Points to file where data is
};
typedef struct SendPDFileSpecBlk SendPDFileSpecBlk,
                                 *SendPDFileSpecBlkPtr;
```

**ERRORS**

```
opNotImpl,
pdBadSelectorErr,
pdBadJobIDErr,
File System Errors
```

## Close Printer Direct

The `Close` commands tells the printer driver to stop receiving printer data and prepare the data to be spooled to the printer.

```
struct ClosePDBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
};
typedef struct ClosePDBlk ClosePDBlk, *ClosePDBlkPtr;
```

**ERRORS**

```
opNotImpl,
pdBadSelectorErr,
pdBadJobIDErr
```

## Despool Print Job

Once the `Close` command is completed, the printer driver will only dispatch the printer data once it has received the `Despool` command. Despooling can take place either in the foreground or the background, depending on the user's selection in the Chooser.

```
struct DespoolPDBlk {
    SInt16 opCode;                  // Printer Direct opCode = 20
    OSErr error;                    // Result code
    UInt32 reserved;                // reserved
    UInt32 selector;                // Function selector code
    UInt32 jobID;                   // ID referencing this job
    UniversalProcPtr idleProcPtr;   // idleProc
};
typedef struct DespoolPDBlk DespoolPDBlk, *DespoolPDBlkPtr;
```

**ERRORS**

```
opNotImpl,
iPrAbort,
pdBadSelectorErr,
pdBadJobIDErr,
pdDespoolFailedErr,
```

An `idleProcPtr` can be supplied for `Despool` function which the driver should execute periodically. The `idleProc` should be a function which takes no arguments and returns no values and should be of the form:

```
pascal void MyIdleProc(void);
```

The primary purpose of the `idleProc` is to sense events when a user is attempting to abort the print job. Should the `idleProc` sense this, it must set the printing error (via the `PrSetError` function) with the `iPrAbort` error. Otherwise, the `idleProc` should return a `noErr`. The driver should check the error returned and abort the print job upon receiving the abort error.

Should the despool procedure be aborted, it should return the `iPrAbort` error. If the despool function should fail for any other reason, it should return the `pdDespoolFailedErr` so the calling application can prompt the user.

If the `idleProc` is `nil`, it is the printer driver's option to supply a default `idleProc` routine.

## Verify Printer Direct Mode

The `Verify` function allows an application to verify that the driver supports printer direct mode. If the driver does not support the printer direct `opCode`, the `PrGeneral` function should return an `opNotImpl` error.

```
struct VerifyPDBlk {
    SInt16 opCode;          // Printer Direct opCode = 20
    OSErr error;            // Result code
    UInt32 reserved;        // reserved
    UInt32 selector;        // Function selector code
    OSType drvrCreator;     // Printer driver's creator type
    NumVersion drvrVersion; // Printer driver's version number
};
typedef struct VerifyPDBlk VerifyPDBlk, *VerifyPDBlkPtr;
```

The `drvrCreator` and `drvrVersion` fields are returned by the printer driver. The `drvrCreator` is a 4 character `OSType` that is part of the driver's file info. The `drvrVersion` field is of type `NumVersion` which is defined in the `Types.h` file of the Macintosh headers. It is defined there as follows:

```
struct NumVersion {
    UInt8 majorRev;         /*1st part of version number in BCD*/
    UInt8 minorAndBugRev;   /*2nd & 3rd part of version number
                             share a byte*/
    UInt8 stage;            /*stage code: dev, alpha, beta, final*/
    UInt8 nonRelRev;        /*revision level of non-released version*/
};
typedef struct NumVersion NumVersion;
```

Based on the creator and version information, and any pertinent information in the printer record, it is up to the calling application to determine whether the printer data being sent to this driver would be handled properly by this printer.

# Summary of the Printer Direct APIs

## Constants

```
#define kPrinterDirectOpCode    20

#define kPDOpenSelect           1
#define kPDSendDataSelect       2
#define kPDSendFileSelect       3
#define kPDCloseSelect          4
#define kPDDespoolSelect        5
#define kPDVerifySelect         6

#define kPDNoData               0
#define kPDData                 1
#define kPDFileSpec             2

enum {
    pdBadSelectorErr            = -10001,
    pdBadSendModeErr            = -10002,
    pdBadJobIDErr               = -10003,
    pdDespoolFailed             = -10004
};
```

## Data Types

```
struct OpenPDBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    UInt32 spoolType;   // Ptr to data or FSSpec.
};
typedef struct OpenPDBlk OpenPDBlk, *OpenPDBlkPtr;
```

```
struct SendPDDataBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    Ptr Data;           // Pointer to the data.
    UInt32 count;       // Number of bytes Data points to.
};
typedef struct SendPDDataBlk SendPDDataBlk, *SendPDDataBlkPtr;


struct SendPDFileSpecBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;    // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
    FSSpec fileSpec;    // Points to file where data is
};
typedef struct SendPDFileSpecBlk SendPDFileSpecBlk,
                                *SendPDFileSpecBlkPtr;


struct ClosePDBlk {
    SInt16 opCode;      // Printer Direct opCode = 20
    OSErr error;        // Result code
    UInt32 reserved;;   // reserved
    UInt32 selector;    // Function selector code
    UInt32 jobID;       // ID referencing this job
};
typedef struct ClosePDBlk ClosePDBlk, *ClosePDBlkPtr;


struct DespoolPDBlk {
    SInt16 opCode;                  // Printer Direct opCode = 20
    OSErr error;                    // Result code
    UInt32 reserved;                // reserved
    UInt32 selector;                // Function selector code
```

```
    UInt32 jobID;                   // ID referencing this job
    UniversalProcPtr idleProcPtr;   // idleProc
};
typedef struct DespoolPDBlk DespoolPDBlk, *DespoolPDBlkPtr;


struct VerifyPDBlk {
    SInt16 opCode;          // Printer Direct opCode = 20
    OSErr error;            // Result code
    UInt32 reserved;        // reserved
    UInt32 selector;        // Function selector code
    OSType drvrCreator;     // Printer driver's creator type
    NumVersion drvrVersion; // Printer driver's version number
};
typedef struct VerifyPDBlk VerifyPDBlk, *VerifyPDBlkPtr;
```

# Summary

The printer-direct mode for Macintosh printer drivers described in this Technote will be used by the PC Print Spooler application, which is part of Apple's new PC Compatibility System and should be relased in the first part of 1996. The PC Print Spooler application spools print jobs from the PC side of the CPU to the printer connected to the Macintosh and selected in the Chooser.

The printer-direct mode will be implemented in PowerPrint 3.0.1 Classic and LT driver packages from GDT Softworks in February, 1996. This feature is also being considered in a future release of Apple's LaserWriter 8 driver. In addition, other third-party providers have shown interest in implementing this feature in their printer drivers in the future.

A caveat: Application developers should realize that this feature is currently not supported by Apple's drivers.

## Acknowledgments

Thanks to Ingrid Kelly, Guillermo Ortiz, and Dave Polaschek for reviewing this Technote. Special thanks to Tom Dowdy for reviewing the design and helping us think about the future compatibility. We would also like to thank Greg

Manning, Gord Pawluik, Pierre Houston, and Dave Banks from GDT Softworks for their help in reviewing and contributing to this Technote.