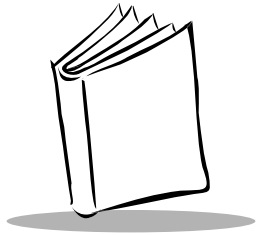
Printer Software Manual for the Palm Computing Platform

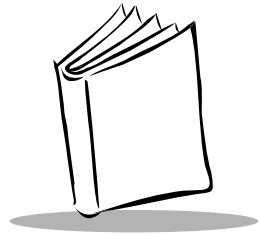
70-35203-01
Revision B
July 1999

symbol®

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com



Contents



Chapter 1 Introduction

This introduction contains an overview of the *Printer Software Manual for the Palm Computing Platform* and provides a list of the appropriate reference documents and conventions. This guide is for developers who want to create print applications for the Palm III or Symbol Palm Terminal. This guide assumes that you are familiar with the CodeWarrior development environment.

The *Printer Software Manual for the Palm Computing Platform* is part of the Symbol Software Development Kit (SDK). Developers can use the SDK to create print applications that use the Palm III and SPT transports (serial port, infrared) to send data from the handheld device to the supported printers listed in the following table.

Commercial Printers	Portable Label Printers
PCL	Comtec
PostScript	Eltron
	Monarch
	O'Neil

See Appendix A for a list of supported printer names and models.

Application Programming Interface (API)

The Symbol printer API has four main features:

- The ability to handle the Palm's multiple communication transports, such as the IrDA port and the RS232 serial port. This feature simplifies applications that use printing, because they do not need to deal with the specifics of each transport.
- High-level text and graphics functions that shield the developer from the details and the escape sequences that must be used to print text and graphics.
- A generic write function that passes the data directly to the printer with no filtering or formatting.
- A set of interfaces for a printer capability database. A printer capability database is similar to a UNIX printcap file. The API provides printcap information for all supported printer types; the Symbol Print API calls retrieve specific printcap information and use it to communicate with a printer. The API can also return printcap information to the application. The printcap file can be expanded for other types of printers.

This API works with other Palm APIs and libraries, such as the Symbol Scanner API, and fits seamlessly into a full-feature printer API that supports JetSend, PCL, and PostScript drivers.

API Architectural Overview

The printer API is a shared library that accesses a printer capability (printcap) database. The printcap database contains:

Printer transport parameters—Transport parameters include parity, baud rate, stop bits, and so on. These parameters will vary from printer to printer, based on which transports are supported. For example, an IR-based printer entry will specify baud rate, while a serial printer will specify baud rate, stop bits, and parity settings.

Printer-specific commands—Each printer has unique commands for such operations as resetting the printer, querying the status of a printer, and initializing the printer. These printer-specific commands have been included in the printcap database, so that developers don't have to know the specific commands for the different printers.

The printcap database is distributed with the printer API. The printers that are currently supported by the printcap database is listed in the Appendix; this list is expected to grow rapidly, and updates to the printcap database can be obtained from Symbol Technology's web site at:

http://www.symbol.com/palm

Figure 1-1 shows an architectural overview of the printer API and its associated printcap database.

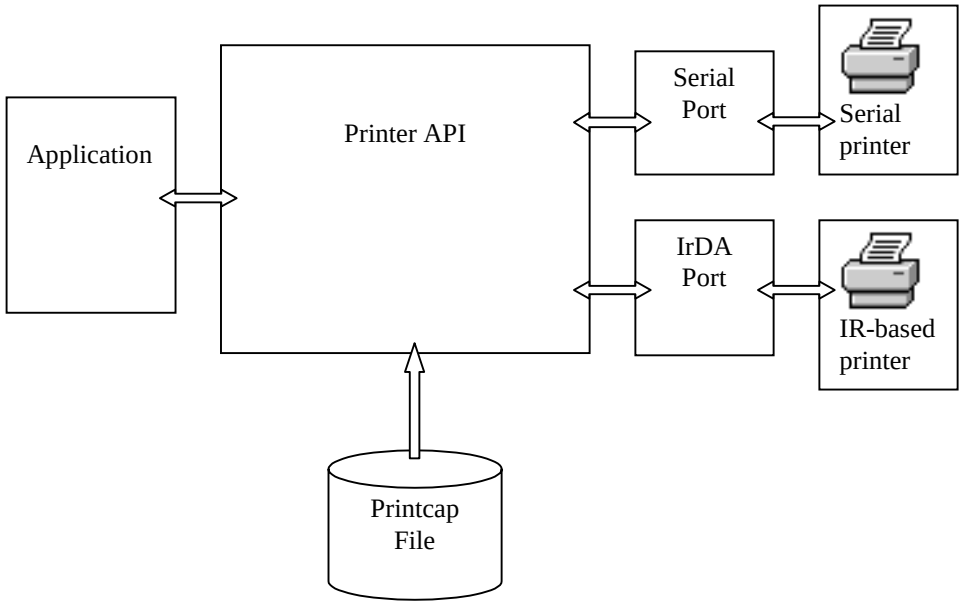


Figure 1-1. Printer API Interfaces

The Symbol printer API architecture provides an interface between your application and the supported transport mechanisms on the Palm device. Currently, two printer transports are supported: IR and serial.

The printer library uses the printcap database to communicate with the desired printer. The printcap database converts generic commands, such as “reset,” into printer-specific strings that each printer can understand.

Your application can also communicate with a printer that is not included in the printcap database. When you use the **`ptOpenPrinter`** function call, you need only set the printer model name to "Unknown," and the printer shared library will not query the printcap database for information. Instead, it is up to you as the developer to provide the parameters. For instance, to reset the printer, the **`ptResetPrinter`** function call is used, but you would be required to provide the reset string to be sent to the printer.

What This Manual Contains

The rest of this manual is organized as follows.

- Print Commands—Description of each function call in the API. These calls are divided into the following sections:
  - General Purpose Interface Functions
  - High-level API Calls
  - Lower-level API Calls

  Also included are definitions of the external data structures and enumerated types used by the Symbol Printer API.
- Sample Application—An example application that uses some of the function calls available in the API.
- Appendix A—A list of supported printers and models. It also provides the information for each printer that is defined in the printcap database.
- Glossary—Definitions of terms used in this manual.

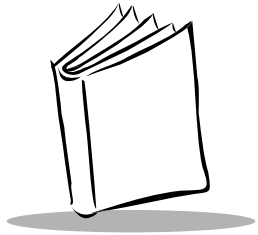
System Requirements

Hardware requirements:	Palm III or SPT 1500 SPT 1700
Software requirements:	Palm OS 3.0 HotSync 2.0 or greater

Conventions Used in this Manual

This guide uses the following typographical conventions.

This style	Is used for . . .
`Fixed width font`	Code elements such as functions, structures, fields, and bitfields
`->`	Input
Blue	Hyperlinks
Italics	Emphasis (for other elements)



Chapter 2 API Function Calls

Introduction

The functions described in this chapter allow you to send text and simple graphic data from a Palm III device or SPT to a supported printer.

Returned Status Definitions

The printer commands in this chapter may return one or more of the status codes described below.

STATUS CODE	DEFINITION
PTStatusAlreadyOpen	The printer is already open.
PTStatusAlreadyConnect	A connection has already been made to the printer.
PTStatusBadParameter	One of the parameters of the function call was incorrect.
PTStatusErrLine	When using the serial transport, a failure occurred during the SerReceive application call.
PTStatusFail	The function call failed.
PTStatusIrBindFailed	When using the IR transport, a failure occurred during the IrBind application call.
PTStatusIrConnectFailed	When using the IR transport, a failure occurred during the IrConnectReq application call.
PTStatusIrConnectLapFailed	When using the IR transport, a failure occurred during the IrConnectLap application call.

STATUS CODE	DEFINITION
PTStatusIrDiscoverFail	When using the IR transport, a failure occurred during the IrDiscoveryReq application call.
PTStatusIrNoDeviceFound	When using the IR transport, no device was found during the IrDiscoveryReq application call.
PTStatusIrQueryFailed	When using the IR transport, a failure occurred during the IrIASQuery application call.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusNotOpen	Tried to use or close a port that was not open.
PTStatusOK	The function call was successfully executed.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.
PTStatusPrinterNotFound	The target printer is not defined in the printcap database.
PTStatusRomIncompatible	The Palm OS version must be 3.0 or greater.
PTStatusTimeOut	A timeout occurred while waiting for data from the printer.
PTStatusTransportNotAvail	The requested transport (serial port, IrDa) is not available.

These status codes are also defined in the **ptPrint.h** files included with the SDK.

Print Commands

The calls in the API provide three types of functions:

FUNCTION TYPE	PURPOSE	PAGE
General Purpose Interface Functions	Control the printer's operation.	2-4
High-level API Calls	Print text and simple graphics.	2-17
Lower-level API Calls	Retrieve printcap entries and send user-formatted data to the printer.	2-27

Also included in this chapter are descriptions of the external data structures and enumerated types used by the Symbol Printer API.

General Purpose Interface Functions

The general purpose interface function calls tell the printer to do the following:

- Open
- Close
- Connect
- Disconnect
- Initialize
- Reset
- Get the printer's status

The calls included in this section are:

ptClosePrinter

Purpose Cleans up and closes the shared library.

Prototype `PTStatus ptClosePrinter ();`

Returned Status

`PTStatusOK`	The function call was successfully executed.

If an error occurred, the returned status will be the following:

`PTStatusNotOpen`	Tried to use or close a port that was not open.

Comments Call **`ptClosePrinter`** after the printing is completed.

See Also **`ptOpenPrinter`**

ptConnectPrinter

Purpose Establishes a connection to the printer using the serial port or an IR connection.

Prototype

```
PTStatus ptConnectPrinter (
                    CharPtr printerName);
```

Parameters

->	**printerName**	Name of printer; used for future transports other than serial or IR.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusNotOpen	Tried to use or close a port that was not open.
PTStatusBadParameter	Serial only. One of the parameters of the function call was incorrect.
PTStatusFail	IrDa only. The function call failed.
PTStatusTransportNotAvail	The requested transport (serial port, IrDa) is not available.
PTStatusAlreadyConnect	A connection has already been made to the printer.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusIrConnectFailed	When using the IR transport, a failure occurred during the IrConnectReq application call.

PTStatusTimeOut	IrDa only. A timeout occurred while waiting for data from the printer.
PTStatusIrNoDeviceFound	IrDa only. When using the IR transport, no device was found during the IrDiscoveryReq application call.
PTStatusIrDiscoverFail	IrDa only. When using the IR transport, a failure occurred during the IrDiscoveryReq application call.
PTStatusIrConnectFailed	IrDa only. When using the IR transport, a failure occurred during the IrConnectReq application call.
PTStatusIrConnectLapFailed	IrDa only. When using the IR transport, a failure occurred during the IrConnectLap application call.
PTStatusIrQueryFailed	IrDa only. When using the IR transport, a failure occurred during the IrIASQuery application call.

Comments For serial and IR transports, set printerName to NULL.

To save battery power on the handheld device, be sure to disconnect the printer (by calling **`ptDisconnectPrinter`**) after your application has successfully printed the data. This is especially important if your application is using the serial port. However, you do not need to close the printer until the application is finished.

See Also **`ptClosePrinter`**

ptDisconnectPrinter

Purpose Disconnects the printer.

Prototype

```
PTStatus ptDisconnectPrinter ();
```

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be the following:

PTStatusNotOpen	Tried to use or close a port that was not open.

Comments This function call closes the port for the serial transport. If the printer is connected through IR, the IR connection is disconnected.

To save battery power on the handheld device, use this function call to disconnect the printer after your application has successfully printed the data. This is especially important if your application is using the serial port. However, you do not need to close the printer until the application is finished.

See Also **ptConnectPrinter**
ptClosePrinter

ptInitPrinter

Purpose Initializes the printer.

Prototype

```
PTStatus ptInitPrinter (
                    VoidPtr initPtr, ULong length);
```

Parameters

-> **initPtr** — Pointer to the string sent as the initialization to the printer. If the initialization string is NULL, the "is" (initialization string) value is taken from the printcap database and sent to the printer.

-> **length** — Length of initialization string.

Returned Status

PTStatusOK — The function call was successfully executed.

If an error occurred, the returned status will be the following:

PTStatusNotOpen — Tried to use or close a port that was not open.

Comments The initialization string is a set of hexadecimal bytes sent to the printer to initialize it. If a printcap entry exists for the attached printer, the default initialization for the printer will already exist in the database, and this argument can be set to NULL. An error is returned if a printcap entry does not exist for the destination printer and the **initPtr** argument is NULL.

This function call overrides any existing initialization settings.

See Also **ptResetPrinter**

ptOpenPrinter

Purpose Opens and initializes the shared library.

Prototype

```
PTStatus ptOpenPrinter(
                CharPtr printerModel,
                PTTransports transport,
                PTConnectSettingsPtr
                connectSettings);
```

Parameters

->	**printerModel**	Model of the printer to be opened. References printcap information.
->	**transport**	The transport type.
->	**connectSettings**	Contains information for baud rate, parity settings, stop bits, and handshaking. This information can overwrite connection settings in the printcap database.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusAlreadyOpen	The printer is already open.
PTStatusRomIncompatible	The Palm OS version must be 3.0 or greater.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrinterNotFound	The target printer is not defined in the printcap database.

PTStatusIrBindFailed	IrDa only. When using the IR transport, a failure occurred during the IrBind application call.
PTStatusTimeOut	IrDa only. A timeout occurred while waiting for data from the printer.

Comments

To use the printcap database default connection settings, set **connectSettings** to NULL. If the printer is connected using IR, only the baud rate applies.

The **ptConnectSettings** structure is documented in the **Data Structures** section at the end of this chapter. It is also included in the **ptPrint.h** file.

See Also

ptClosePrinter

ptPrintApiVersion

Purpose Returns the version number of the Printer Library.

Prototype

```
PTStatus ptPrintApiVersion (
                    CharPtr ptr, Int len);
```

Parameters

-> **ptr**	Pointer to the character buffer in which the version number will be placed.
-> **len**	Size of the character buffer.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusNotOpen	The application must first call **ptOpenPrinter** before using this function.
PTStatusFail	The **ptr** parameter is invalid.

Comments In version 1.0, the buffer must be 12 characters long, as the return string is “ptPrint 1.0.”

You must open the printer library before making this function call. You can close the library after the call.

ptQueryPrinter

Purpose Queries the printer condition.

Prototype

```
PTStatus ptQueryPrinter (
                VoidPtr queryPtr, ULong length,
                VoidPtr queryResPtr,
                ULong queryResLen);
```

Parameters

-> **queryPtr**	Pointer to the memory used to send the query to the printer. If it is NULL, the "qs" (query string) value from the printcap database entry for this printer is used.
-> **length**	Length of query data to be sent to the printer.
-> **queryResPtr**	Pointer to the memory used to hold any response from the query. This buffer is provided and released by the application.
-> **queryResLen**	Length of the query response buffer pointed to by **queryResPtr**.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusErrLine	Serial only. When using the serial transport, a failure occurred during the SerReceive application call.
PTStatusNotOpen	Tried to use or close a port that was not open.

PTStatusTimeOut	IrDa only. A timeout occurred while waiting for data from the printer.

Comments

The data referenced by **queryPtr** is a set of hexadecimal bytes that represents the printer-specific query command. The data is sent to the printer to query its status. The response is then placed into the memory allocated by the application, pointed to by **queryResPtr**.

An error is returned to the calling application if a printcap entry for the destination printer does not exist and the **queryPtr** argument is NULL.

ptResetPrinter

Purpose Resets the printer.

Prototype

```
PTStatus ptResetPrinter (
                    VoidPtr resetPtr, ULong length);
```

Parameters

->	**resetPtr**	Pointer to data that is sent to the printer; this data is used to reset the printer. If it is NULL, the "rs" (reset string) value from the printcap database entry for this printer is used.
->	**length**	Length of reset string.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be the following:

PTStatusNotOpen	Tried to use or close a port that was not open.

Comments The reset string is a set of hexadecimal bytes sent to the printer to reset it. An error is returned if a printcap entry does not exist for the destination printer and the **resetPtr** argument is NULL.

See Also **ptInitPrinter**

High-level API Calls

The high-level API calls send commands to the printer to print the following:

- Text
- Simple graphics

The calls included in this section are:

The high-level API calls rely on the presence of a print buffer that is allocated by calling the **ptStartPrintBuffer** function. Each call to **ptLineToBuffer**, **ptRectToBuffer**, and **ptTextToBuffer** adds commands to the print buffer. Calling **ptPrintPrintBuffer** sends the commands to the printer and frees the print buffer memory.

ptLineToBuffer

Purpose Adds the print line command to the buffer.

Prototype

```
PTStatus ptLineToBuffer (
                UInt xStart, UInt yStart,
                UInt xEnd, UInt yEnd,
                UInt lineThickness);
```

Parameters

-> **xStart**	X-coordinate of line start point.
-> **yStart**	Y-coordinate of line start point.
-> **xEnd**	X-coordinate of line end point.
-> **yEnd**	Y-coordinate of line end point.
-> **lineThickness**	Thickness of the line to be drawn (dots).

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

Comments

`ptLineToBuffer` determines which command sequence from the printcap database is needed to print a simple line on a specific printer. It adds the command to the allocated buffer and includes logic to expand the buffer size if it is not large enough to hold the command and the line.

Refer to the printer documentation for a definition of the start and end points.

See Also

`ptTextToBuffer`
`ptRectToBuffer`

ptPrintPrintBuffer

Purpose Prints the data residing in the print buffer, then frees the memory in the print buffer.

Prototype

```
PTStatus ptPrintPrintBuffer (
                    CharPtr printerName);
```

Parameters

->	**printerName**	Name of printer; used for future transports other than serial or IR.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

Comments For serial and IR transports, set **printerName** to NULL.

To save battery power on the handheld device, be sure to disconnect the printer (by calling **ptDisconnectPrinter**) after your application has successfully printed the data. This is especially important if your application is using the serial port. However, you do not need to close the printer until the application is finished.

After the print buffer has been sent to the printer, the print buffer memory is freed.

ptRectToBuffer

Purpose Adds the print rectangle command to the buffer.

Prototype

```
PTStatus ptRectToBuffer (
                UInt xTopLeft, UInt yTopLeft,
                UInt xBottomRight,
                UInt yBottomRight,
                UInt lineThickness);
```

Parameters

-> **xTopLeft**	X-coordinate of rectangle top left point.
-> **yTopLeft**	Y-coordinate of rectangle top left point.
-> **xBottomRight**	X-coordinate of rectangle bottom right point.
-> **yBottomRight**	Y-coordinate of rectangle bottom right point.
-> **lineThickness**	Thickness of the lines of the rectangle (dots).

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

Comments **`ptRectToBuffer`** determines which command sequence from the printcap database is needed to print a rectangle on a specific printer. It adds the command to the allocated buffer and includes logic to expand the buffer size if it is not large enough to hold the command and the rectangle.

See Also **`ptLineToBuffer`**
`ptTextToBuffer`

ptResetPrintBuffer

Purpose Frees the print buffer memory.

Prototype `PTStatus ptResetPrintBuffer ()`

Returned Status

`PTStatusOK`	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

`PTStatusFail`	The function call failed.
`PTStatusNoMemory`	No dynamic heap space is available to hold library data.
`PTStatusPrintCapFailed`	The function call tried to access a printcap entry that doesn't exist.

Comments This function call performs the same function as **`ptPrintPrintBuffer`**, except it doesn't print the data in the print buffer.

ptSetFont

Purpose Sets the printer's typeface and type size.

Prototype

```
PTStatus ptSetFont (
                    CharPtr fontBuffPtr);
```

Parameters

->	**fontBuffPtr**	Contains information about the font the user would like to set.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

Comments The format of fontBuffPtr is:

<font type>;<font width>;<font height>

The printer API parses the **fontBuffPtr** string to get the font type, width, and length. When text is printed, the API substitutes the variable data for the font.

If a printer doesn't require the font width, you can leave that variable blank by providing only

<font type>;;<font height>

You must include both semicolons between **<font type>** and **<font height>** so the printer API recognizes that the **<font width>** variable is missing.

ptStartPrintBuffer

Purpose Sets the printer library's initial buffer size.

Prototype

```
PTStatus ptStartPrintBuffer (
                ULong size);
```

Parameters

->	**size**	Length of the initial buffer.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

See Also

ptTextToBuffer
ptLineToBuffer
ptRectToBuffer
ptPrintPrintBuffer

ptTextToBuffer

Purpose Adds the print text command to the buffer.

Prototype

```
PTStatus ptTextToBuffer (
                UInt xStart, UInt y Start,
                CharPtr pText);
```

Parameters

-> **xStart**	Beginning X-coordinate of print data.
-> **yStart**	Beginning Y-coordinate of print data.
-> **pText**	Pointer to text that will be printed.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusFail	The function call failed.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.

Comments **ptTextToBuffer** determines which command sequence from the printcap database is needed to print text on a specific printer. It adds the command and the text to the allocated buffer. **ptTextToBuffer** includes logic to expand the size of the buffer if it is not large enough to hold the command and the text.

See Also **ptLineToBuffer**
ptRectToBuffer

Lower-level API Calls

The lower-level API calls do the following:

- Retrieve printcap entries
- Send user-formatted data to the printer

The calls included in this section are:

ptQueryPrintCap

Purpose Queries the printcap database for the given token.

Prototype

```
PTStatus ptQueryPrintCap (
                CharPtr query,
                VoidPtr queryResPtr,
                ULong queryResLen);
```

Parameters

-> **query** — Pointer to the memory used to find the value in the printcap entry for the open printer.

-> **queryResPtr** — Pointer to the memory used to hold any response from the query.

-> **queryResLen** — Maximum length of the query response.

Returned Status

PTStatusOK — The function call was successfully executed.

If an error occurred, the returned status will be the following:

PTStatusNotOpen — Tried to use or close a port that was not open.

Comments After querying the printcap database, the **queryResPtr** buffer is filled in with the query results. Some special characters will be represented by hexadecimal bytes. For example, ESC will be converted to the hexadecimal value 0x1B. Other escape character conversions are also supported, such as ALT, CTRL, Newline, and Carriage Return.

ptWritePrinter

Purpose Writes data to the printer.

Prototype

```
PTStatus ptWritePrinter (
                VoidPtr buffer, ULong length);
```

Parameters

-> **buffer**	Pointer to the data to send to the printer.
-> **length**	Length of the data to be written.

Returned Status

PTStatusOK	The function call was successfully executed.

If an error occurred, the returned status will be one of the following:

PTStatusNotOpen	Tried to use or close a port that was not open.
PTStatusTimeOut	A timeout occurred while waiting for data from the printer.

Comments To successfully use this function call, you must be familiar with the destination printer language.

The content of the buffer is provided by the developer; the assumption is that the contents are well-formatted for the destination printer. With **ptWritePrinter**, the developer can communicate directly with the printer.

The **ptWritePrinter** function call can be used to print barcodes, text, and simple graphics on legacy printers, which rely on forms.

Data Structures

This section defines the data structures and enumerated types used by the printer API as noted in the following table.

Printcap Database

The printcap database is an ASCII-based, token=value file structure that describes the characteristics of each supported printer. This structure is owned by Symbol Technologies and the printer manufacturers. Entries to this database will be updated by Symbol, and will be made available on their website at:

http://www.symbol.com/palm

PTConnectSettings

The **PTConnectSettings** structure represents the serial or IrDA setting information.

```
typedef struct tagPTConnectSettings {

   ULong   baudRate;           // baud rate
   ULong   timeOut;            // time out in System Ticks
   ULong   flags;              // transport flags; see SerialMgr.h
   UInt    recvBufSize;        // receive buffer size

} PTConnectSettings;
typedef PTConnectSettings * PTConnectSettingsPtr;
```

Available serial port baud rates for **connectSettings** are 9600, 14400, 19200, 38400, 57600, and 115200. The available flags in the SerialMgr.h file are:

```
#define serSettingsFlagStopBitsM   0x00000001  // mask for stop bits field
#define serSettingsFlagStopBits1   0x00000000  // 1 stop bits
#define serSettingsFlagStopBits2   0x00000001  // 2 stop bits
#define serSettingsFlagParityOnM   0x00000002  // mask for parity on
#define serSettingsFlagParityEvenM 0x00000004  // mask for parity even
```

Available IR baud rates for **connectSettings** are 9600, 57600, and 115200. Refer to the Irlib.h file for a list of available flags for this structure.

PTStatus

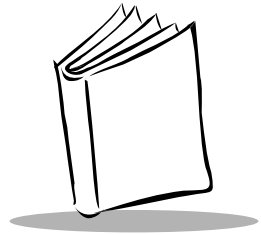
PTStatus represents the possible status values from the printer API to the application.

```
typedef enum
{
   PTStatusAlreadyOpen,
   PTStatusAlreadyConnect,
   PTStatusBadParameter,
   PTStatusErrLine,
   PTStatusFail,
   PTStatusIrBindFailed,
   PTStatusIrConnectFailed,
   PTStatusIrConnectLapFailed,
   PTStatusIrDiscoverFail,
   PTStatusIrNoDeviceFound,
   PTStatusIrQueryFailed,
   PTStatusNoMemory,
   PTStatusNotOpen,
   PTStatusPrinterNotFound,
   PTStatusRomIncompatible,
   PTStatusTimeOut,
   PTStatusTransportNotAvail,
} PTStatus;
```

PTTransports

PTTransports represents the possible transports.

```
typedef enum
{
   PTSerial,
   PTIr,
   PTTransportMax,
} PTTransports;
```



Chapter 3 Sample Application

The Symbol Printer API includes a simple example of a print application. This application is a single screen, and its source code serves as an example of how to use the print library. In the sample application, you choose the transport method, either serial or IR, and then choose one of the printers listed on the right of the screen. Nine buttons are listed in the **COMMAND** popup menu at the bottom left of the screen; their functionality is listed below.

OPEN button—When you tap the OPEN button, the printer library opens, is loaded into memory, and initialized. The printcap database opens, and the entry for the chosen printer is found.

CLOSE button—To close the printer library, tap the CLOSE button. If you have not already opened the print library, an error message displays.

FORM button—When you tap the FORM button, a canned form is loaded onto the specified printer. If you have not opened the printer library, an error message displays, telling you to do so. If the form has already been downloaded to the printer, it should still be memory-resident, and you do not need to download it again. This button uses the low-level API calls to print.

DATA button—To print the form on the attached printer, tap the DATA button. If you have not opened the print library, an error message displays, telling you to do so. This button uses the low-level API to print.

QUERY PRINTER button—To view a list of printer status parameters, tap the QUERY PRINTER button. A popup window opens. Tap the OK button to close the popup.

INIT PRINTER button—To initialize the selected printer, tap the PRINTER INIT button.

RESET PRINTER button—To reset the selected printer, tap the RESET PRINTER button.

API VERSION button—To determine the version number of the print library, tap the API VERSION button. If you have not already opened the print library, an error may be displayed.

HIGH LEVEL API button—To print text or a simple graphic using the high-level API calls, tap the HIGH LEVEL API button. The high-level API functions are supported on all of the printers listed in the sample application, except Monarch. Before using the high-level API calls, you must tap the OPEN button to open the printer. Be sure to CLOSE the printer after the data has been printed.

Six buttons are listed in the **`PrintCap Query`** popup menu at the bottom right of the screen; tap the appropriate button to display the printcap entry for the following:

- Baud Rate
- Stop Bit
- Parity
- Query Value
- Init Value
- Reset Value

To display a printcap entry, you must select both a TRANSPORT and a PRINTER, and the printer library must be opened.

Code Samples

For your application to print data using the Symbol print API, it must first accomplish several important steps. The following sections present some examples of how to do this. Please use the example print application as well.

Opening the Print Library

The first step in using the printer API is to open and initialize the library; this is done with the **`ptOpenPrinter`** function call. In this call, you must specify the printer model name and the transport mechanism. Below is an example of the code needed for the O'Neil Microflash printer.

```
Err error;
Boolean retval = FALSE;
PTConnectSettingsPtr pSettings = NULL;

error = ptOpenPrinter( "Oneil", PTSerial, pSettings );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,
       "Cannot open print library or find ONeil printcap entry");
       return retval;
}
```

If you are using a printer that is not included in the printcap database, specify "Unknown" as your printer model name, as shown below.

```
Err error;
Boolean retval = FALSE;
PTConnectSettings cSettings;

// specify connection settings
cSettings.baudRate = PTDefaultSerBaudRate;
cSettings.timeOut = PTDefaultSerTimeout;
cSettings.flags = PTDefaultSerFlags;
cSettings.recvBufSize = PTDefaultSerRecvBuf;

// connect to serial printer
error = ptOpenPrinter( "Unknown", PTSerial, &cSettings );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,
       "Cannot initialize print library");
       return retval;
}
```

Similarly, if you are printing with a PostScript printer, specify it as your printer model name in the **ptOpenPrinter** function call.

```
Err error;
Boolean retval = FALSE;
PTConnectSettingsPtr pSettings = NULL;
```

```
error = ptOpenPrinter( "Postscript", PTSerial, pSettings );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,
       "Cannot open print library or find Postscript printcap entry");
       return retval;
}
```

Connecting to the Printer

The second step in using the printer API is to connect to the printer; this is done with the **ptConnectPrinter** function call. In this call, the library searches for the printer and makes either a serial or IR connection, depending on the transport that you specified in the **ptOpenPrinter** function call.

The printer name parameter has been reserved for use with future transport types. It is not used or recognized if the transport type is serial or IR.

```
Err error;
Boolean retval = FALSE;
CharPtr printerName = NULL;

error = ptConnectPrinter( printerName );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,  "Could not connect printer");
       ptClosePrinter();
       return retval;
}
```

Querying the Printer

To get the printer's status, you can query the printer with the **ptQueryPrinter** function call. This requires that you have already issued the **ptOpenPrinter** call and the **ptConnectPrinter**, and that a valid connection to the printer exists. The example below assumes that you have connected to a printer that has a valid printcap entry; in this case, you do not need to provide a query string, since it is already in the printcap database.

```
#define StatusBuffLen 256

Err error;
Boolean retval = FALSE;
VoidPtr queryPtr = NULL;
Byte statusBuffer[StatusBuffLen];
```

```
error = ptQueryPrinter( queryPtr, (ULong) 0, &statusBuffer,
             (ULong) StatusBuffLen );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,  "Could not query the printer");
       ptClosePrinter();
       return retval;
}
```

In the second query example, let's assume that we've already connected to an “Unknown” printer that doesn't have a printcap entry. In this case, we would have to provide a query string that the printer can understand.

```
#define StatusBuffLen 256

Err error;
Boolean retval = FALSE;
CharPtr queryString = "QUERY";                  // made-up query string
                                                // for Unknown printer
Byte statusBuffer[StatusBuffLen ];

error = ptQueryPrinter( queryString,
             (ULong) ( StrLen(queryString) ),
             &statusBuffer,
             (ULong) StatusBuffLen );

if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,  "Could not query the printer");
       ptClosePrinter();
       return retval;
}
```

As the code example above illustrates, after you connect to an “Unknown” printer, you must supply commands that your printer understands for the following function calls:

- **ptQueryPrinter**
- **ptInitPrinter**
- **ptResetPrinter**

If you do not provide the custom commands for the query, initialization, or reset function calls, they will fail.

Writing Data to the Printer

There are two ways of writing to the printer. You can either use the high-level API calls or you can use the low-level **ptWritePrinter** function call. If you use ptWritePrinter, the data you send to the printer must be well-formed for the specific printer.

The high-level API calls are designed so that you can use the same set of calls for multiple printers. The functions take the input parameters and ensure that the data destined for the printer is well-formed for the specific printer.

Using the Low-Level API

If you have created a custom label or custom form for the printer, you can simply send the form data to the printer using the **ptWritePrinter** function call. In the example below, the form data has been statically included in the C-code. (This form data is customized for the Comtec RP3 printer.)

```
Err error;
Boolean retval = FALSE;
CharPtr pForm =   "! DF SHELF.FMT\r\n" \
                  "! 0 200 230 230 1\r\n" \
                  "BOX 100 100 480 210 1\r\n" \
                  "CENTER\r\n" \
                  "TEXT 4 3 0 15 " "\\\\\r\n" \
                  "TEXT 4 0 0 95 " "\\\\\r\n" \
                  "BARCODE UPCA 1 1 40 0 145 " "\\\\\r\n" \
                  "TEXT 7 0 0 185 " "\\\\\r\n" \
                  "FORM\r\n" \
                  "PRINT\r\n";
CharPtr pData =  "! UF SHELF.FMT\r\n" \
                  "$99.99\r\n" \
                  "SWEATSHIRT\r\n" \
                  "40123456784\r\n" \
                  "40123456784\r\n";

// download the form to the printer
error = ptWritePrinter( pForm, StrLen(pForm)  );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,  "Form download failed");
       ptClosePrinter();
       return retval;
}
```

```
// write the data to the printer
error = ptWritePrinter( pData, StrLen(pData) );
if (error != PTStatusOK)
{
       ErrNonFatalDisplayIf(error,  "Write failed");
       ptClosePrinter();
       return retval;
}
```

Using the High-Level API

If you would like to write data to the printer using the high-level API, you must use a series of function calls to set up the print buffer, format the output, and send the resulting print buffer to the printer. For example:

```
// initialize the print buffer with a starting length
PTStatus       status;
status = ptStartPrintBuffer( 256 );
if ( PTStatusOK != status )
       return status;

// format some text
status = ptTextToBuffer( 0, 30, "This is some sample text" );
if ( PTStatusOK != status)
{
       ptResetPrintBuffer();
       return status;
}

// format a line, y = 50, xStart=0, xEnd=120
status = ptLineToBuffer( 0, 50, 120, 50 );
if ( PTStatusOK != status )
{
       ptResetPrintBuffer();
       return status;
}
```

```
// format a rectangle
//    xStart, yStart = (20, 215)
//    xEnd, yEnd = (200, 280)
//    thickness = 4 dots
status = ptRectToBuffer( 20, 215, 200, 280, 4 );
if ( PTStatusOK != status )
{
      ptResetPrintBuffer();
      return status;
}

// write the print buffer to the printer
status = ptPrintPrintBuffer( NULL );
if ( PTStatusOK != status )
{
      ptResetPrintBuffer();
      return status;
}
```

Disconnecting the Printer and Closing the Library

In your application, once you're done with the printing operation, disconnect the printer and close the print library.

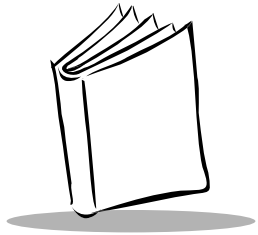
```
Err error;

// Disconnect the printer
error = ptDisconnectPrinter( );

// Close the printer
error = ptClosePrinter();
```

If your application could print data at various times, you should call **`ptDisconnectPrinter`** to disconnect the printer between print jobs; however, you do not need to close the printer each time (by using the **`ptClosePrinter`** call).

If you do not disconnect the printer between print jobs, you leave the serial port or IR connection open, and the battery on the Palm or SPT device could drain quickly.



Appendix A

Supported Printers

This Appendix lists the information that is currently maintained in the printcap database. It is subject to change, and will in fact change rapidly as Symbol supports additional printers. Current versions of the printcap database can be obtained from Symbol Technology's web site at:

http://www.symbol.com/palm

The first piece of information you need is the printer model name. For instance, if you are working with the ONeil printer, you need to provide the string “ONeil” as the printer model in the **`ptOpenPrinter`** function call. If you are working with the Paxar Monarch 9490 printer, you need to provide either the string “Paxar Monarch 9490” or “Monarch 9490.”

Additional information in the printcap database is also listed here. Under each supported printer model name is a list of fields that are included in the printcap database. The format of each piece of information in the printcap database is a token=value pair. For instance, to retrieve the reset string for the RP3 printer, you would provide “rs” as the query parameter in the **`ptQueryPrintCap`** function call.

The abbreviations used in the tables that follow are:

- br=baud rate
- sb=stop bits
- pr=parity
- is=initialization string
- rs=reset string

- qs=query string
- fD=default font
- f1...fn=other available fonts
- bi=high-level API init
- bt=text command
- bl=line command
- bb=rectangle (box)
- be=high-level API end

Variables in Printcap Strings

The following table lists the definitions of the variables found in the printcap strings for the supported printers. The high-level function calls substitute these variables with the appropriate arguments.

Variable	Definition
%s	text string
%t	thickness
%L	text length
%x	xStart coordinate
%y	yStart coordinate
%X	xEnd coordinate
%Y	yEnd coordinate
%l	horizontal length (xEnd - xStart)
%H	vertical height (yEnd - yStart)
%B	Line extension ([xEnd - xStart] + thickness) See the section that follows.
%f	font name
%h	font height
%w	font width

Printing Rectangles

The information in this section applies ONLY to printers that do not print rectangles. These printers create a rectangle by drawing four separate lines.

To construct a rectangle, the printer draws four lines, using the following beginning and ending points:

When a line is printed, the beginning and ending coordinates correspond to the line's top or bottom left corner. However, the line's thickness is not taken into account, and the lines therefore may not line up properly. (Notice how line DC extends only to the left side of line BC.) To compensate for the line's thickness, an additional variable, **%B**, has been defined that adds the line's thickness to the ending coordinates. (For example, in the illustration above, %B would add the thickness of line BC to the ending coordinate of line DC.)

Portable Label Printers

Printer model name: Comtec

Comtec printcap tokens and associated values:

Token	Token description	String Value
br	Baud rate	19200
sb	Stop bits	1
pr	Parity	n
rs	Reset string	<ESC>N
qs	Query string	<ESC>v
fD	Default font; Comtec font #7, size=0 (char height = 24 pixels)	7;0
f1	Other available font; Comtec font #4, size=0 (height = 47 pixels)	4;0
bi	High-level API init	! 0 200 200 500 1\r\nSETFF 25 2.5\r\n
bt	Text command	TEXT %f %w %x %y %s\r\n
bl	Line command	LINE %x %y %X %Y %t\r\n
bb	Rectangle (box)	BOX %x %y %X %Y %t\r\n
be	High-level API end	FORM\r\nPRINT\r\n

Comtec has no initialization string (“is”).

All supported Comtec printers understand the same printer language.

Printer model name: Eltron

Eltron printcap tokens and associated values:

Token	Token description	String Value
rs	Reset string	<CTRL>@
br	Baud rate	9600
sb	Stop bits	1
pr	Parity	n
fD	Default font; Eltron font #2, 16.9 cpi, 7 pt.	2;1;1
f1	Other available font; Eltron font #3, 14.5 cpi, 10 pt.	3;1;1
bi	High-level API init	\r\nN\r\n
bt	Text command	A%x,%y,0,%f,%w,%h,N,"%s"\r\n
bl	Line command	LO%x,%y,%l,%t\r\n
bb	Rectangle (box)	LO%x,%y,%t,%H\r\nLO%X,%y,%t,%H\r\nLO%x,%y,%l,%t\r\nLO%x,%Y,%B,%t\r\n
be	High-level API end	P1\r\n

Eltron has no initialization string (“is”) or query string (“qs”).

All supported Eltron printers understand the same printer language.

Printer model name: Paxar Monarch 9490
Monarch 9490

Monarch 9490 printcap tokens and associated values:

Token	Token description	String Value
br	Baud rate	9600
sb	Stop bits	1
pr	Parity	n
is	Initialization string	<CTRL>ER{I,E,"~123~044~034~124~125~126~094","","|}\<CTRL>ER
rs	Reset string	<CTRL>PR
qs	Query string	005

Monarch printers do not support the high-level function calls and the associated tokens:

- Default font (“fD”)
- Other available font (“f1”)
- High-level API init (“bi”)
- Text command (“bt”)
- Line command (“bl”)
- High-level API end (“be”)

Printer model name: ONeil

ONeil printcap tokens and associated values:

Token	Token description	String Value
br	Baud rate	9600
sb	Stop bits	1
pr	Parity	n
rs	Reset string	<ESC>+{RE!}
qs	Query string	<ESC>+{IR?}
fD	Default font; MicroFlash font 204 (MF204) (20.4 CPI, 224 characters block normal)	MF204;1;1
f1	Other available font; MicroFlash font 102 (MF102) (10.2 cpi, 223 characters medium block bold)	MF102;1;1
bi	High-level API init	<ESC>EZ\r\n{PRINT\:\r\n
bt	Text command	@%y,%x\:%f,HMULT%h,VMULT%w|%s|\r\n
bl	Line command	@%y,%x\:HLINE, length %l, thick %t|\r\n
bb	Rectangle (box)	@%y,%x\:T, L %l, T %t|\r\n@%y,%X\:V, L %H, T %t|\r\n@%y,%x\:V, L %H, T %t|\r\n@%Y,%x\:T, L %B, T %t|\r\n:\
be	High-level API end	} \r\n{AHEAD\:200} \r\n

ONeil has no initialization string (“is”).

All supported ONeil printers understand the same printer language.

Commercial Printers

Printer model name: PCL

PCL printcap tokens and associated values:

Token	Token description	String Value
rs	Reset string	<ESC>E
br	Baud rate	9600
sb	Stop bits	1
pr	Parity	n
fD	Default font; Times Roman, 10 pt.	<ESC>(s1p10v0s5t0B
f1	Other available font; Times Roman, 12 pt.	<ESC>(s1p12v0s5t0B
f2	Other available font; Times Roman, 12 pt., bold	<ESC>(s1p12v0s5t3B
bi	High-level API init	<ESC>\\\%0A\n %f
bt	Text command	%f <ESC>*p%xx%yY\n <ESC>&p%LX%s\n
bl	Line command	<ESC>*p%xx%yY\n <ESC>*r1A <ESC>*c%la%tb0P\n <ESC>*rB\n
bb	Rectangle (box)	<ESC>*r0A\n <ESC>*t75R\n <ESC>*p%xx%yY\n <ESC>*c%la%tb0P\n <ESC>*rB\n\ <ESC>*r0A\n <ESC>*t75R\n <ESC>*p%Xx%yY\n <ESC>*c%ta%Hb0P\n <ESC>*rB\n\ <ESC>*r0A\n <ESC>*t75R\n <ESC>*p%xx%yY\n <ESC>*c%ta%Hb0P\n <ESC>*rB\n\ <ESC>*r0A\n <ESC>*t75R\n <ESC>*p%xx%YY\n <ESC>*c%Ba%tb0P\n <ESC>*rB\n
be	High-level API end	<ESC>&l0H\n

The PCL language does not define an initialization string (“is”) or query string (“qs”); these are specific to each printer. If you call **`ptInitPrinter`** or **`ptQueryPrinter`** and don’t specify an initialization string or query string, you will get a **`ptStatusFail`** error message.

Printer model name: Postscript

Postscript printcap tokens and associated values:

Token	Token description	String Value
br	Baud rate	9600
sb	Stop bits	1
pr	Parity	n
fD	Default font; Times Roman, 12 pt.	Times-Roman findfont\r\n 12 scalefont\r\nsetfont\r\n
f1	Other available font; Courier, 12 pt.	Courier findfont\r\n12 scalefont\r\nsetfont\r\n
bi	High-level API init	\\%!\r\n%f\r\n\/pageheight 792 def/r/n\/fontheight 12 def\/cord { fontheight add } def/r/n
bt	Text command	%f\r\n newpath\r\n %x pageheight %y cord sub moveto\r\n (%s) show\r\n
bl	Line command	newpath\r\n %x pageheight %y sub moveto\r\n %t setlinewidth\r\n %X pageheight %Y sub lineto\r\n closepath\r\n stroke\r\n
bb	Rectangle (box)	newpath\r\n %x pageheight %y sub moveto\r\n %t setlinewidth\r\n %X pageheight %y sub lineto\r\n %X pageheight %Y sub lineto\r\n %x pageheight %Y sub lineto\r\n closepath\r\n stroke\r\n
be	High-level API end	showpage\r\n

The Postscript language does not define an initialization string (“is”), reset string (“rs”), or query string (“qs”); these are specific to each printer. If you call **ptInitPrinter**, **ptResetPrinter**, or **ptQueryPrinter** and don’t specify an initialization string, reset string, or query string, you will get a **ptStatusFail** error message.

Using Forms with Legacy Printers

Many of the legacy printers support custom labels or forms. Typically a form can be designed with a specific size and format and with different objects included, such as bar code fields, text, and simple graphics. To create a form, you must refer to the documentation that is distributed by each printer manufacturer. Once a form is created, it can be downloaded to a printer; then variable data can be sent to the printer to "fill out" the form. A good use of this technology would be in a vertical application, where the same type of form or label is printed repeatedly but with different data each time.

A form is a series of commands to the printer. We've provided an example below that is specific for the Comtec RP3 printer:

```
"! UTILITIES\r\n" \
"IN-MILLIMETERS\r\n" \
"SETFF 25 2.5\r\n" \
"PRINT\r\n" \
"! DF SHELF.FMT\r\n" \
"! 0 200 200 210 1\r\n" \
"BOX 100 100 480 210 1\r\n" \
"CENTER\r\n" \
"TEXT 4 3 0 15 " "\\\\\r\n" \
"TEXT 4 0 0 95 " "\\\\\r\n" \
"BARCODE UPCA 1 1 40 0 145 " "\\\\\r\n" \
"TEXT 7 0 0 185 " "\\\\\r\n" \
"FORM\r\n" \
"PRINT\r\n";
```

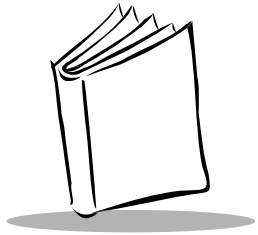
(In this example, "\n" represents a newline character, and the backslash "\" character is a continuation character, so the form can be spread out over several lines.) The example above prints out a barcode, a price, and a description on the Comtec RP3 printer.

Also, some printer manufacturers allow you to design forms on your desktop PC. Some of the design tools will produce text-based files. If they do, you can simply copy the text to your Palm application source code (either as a static string or as a string resource in Constructor).

Writing the form to the printer

Once the form data has been included into your application as a string, it's simple to get a pointer to your string. The string (which is really an entire form layout) can then be written directly to the printer, using the **`ptWritePrinter`** function call.

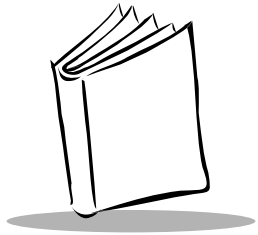
If the **`ptWritePrinter`** function returns with no error, you can assume that the form has been loaded onto your printer, and it's now a matter of writing variable data to the printer. This variable data can also be written to the printer using the **`ptWritePrinter`** function call.



Glossary

Term	Definition
Form	A set of printer-specific commands that format a receipt or label. Refer to the printer manufacturer's documentation for more detailed information.
General purpose interface functions	A set of commands that control the basic operation of the printer, such as opening the printer, initializing the printer, and closing the printer.
High-level API calls	A set of commands to the printer to print text and simple graphics.
Initialization string	A printer-specific command that initializes or sets defaults for a printer. Refer to the printer manufacturer's documentation for more detailed information.
Legacy printer	A printer that uses printer-specific or proprietary control commands. Compare with industry standard commands such as PCL or Postscript.
Lower level API calls	API calls that write directly to the printer.
Printcap database	A database of printer capability and initialization information, such as baud rates, stop bits, and reset commands.

Term	Definition
PCL	Printer Control Language. A language that enables application progams to control various Hewlett-Packard printers.
Postscript	A programming language that describes the appearance of a printed page.
Printer library	A static body of compiled code, containing an API to interface with the serial and IrDa ports of a Palm PDA. The library is linked with the application.
Printer model	The name of the printer in the printcap entry. For example, Comtec RP3.
Printer name	Not currently used. Will be used to identify specific printers.
Shared library	A shared body of compiled code that resides on the Palm PDA. At run time, this library is dynamically linked with the application.



Index