Programmer's Manual

MONARCH® DOS-based PATHFINDER® Ultra® System

```
int LoadFormat(void)
{
 int
  iStatus,
  iLength;

 iLength = strlen(pszFormat);
 iStatus = pclWrite(pszFormat, iLength);
 if (iStatus == 0)
 {
  iLength = strlen(pszBatch);
  iStatus = pclWrite(pszBatch, iLength);
 }
 return iStatus;
 }
```

Monarch®
PAXAR

TC6035PM Rev. CA 6/01

TABLE OF CONTENTS

INTRODUCTION

1

Welcome to the MONARCH® DOS-based PATHFINDER® Ultra® printer's software development kit (SDK). The SDK consists of software and both online and printed documentation.

This chapter introduces the kit, describing

- About this Manual
- System Requirements
- SDK Contents
- Related Documentation

About this Manual

This manual is written for experienced Microsoft® C/C++ programmers who write applications for the printer. These programmers should also be familiar with Monarch's MPCLII printer language.

The following table describes the conventions used in this manual.

Convention	Description
[ ]	Brackets indicate optional items.
...	Ellipses indicate the preceding item is repeated one or more times.
Italics	An item appearing in italics is a variable, a function parameter, or a value of a variable.
Bold	An item appearing in bold is being emphasized.
+	A plus sign placed between two keys indicates to press the keys at the same time.

System Requirements

Following are the hardware and software requirements for the SDK. Refer to your Windows™ and C/C++ documentation for additional requirements.

Hardware Requirements

You need an IBM® PC or 100% compatible with

- 80486 or higher processor
- VGA monitor (SVGA recommended)
- at least 8 MB of memory (16 MB recommended)
- hard disk with 8 MB of free space (not including the space needed for C/C++)
- Microsoft Windows 3.1- or Windows 95®-compatible CD-ROM drive
- serial port
- Microsoft mouse or compatible pointing device
- either of the following serial cables:
  - 9-pin female to 8-pin male PC cable (part #11658730)
  - 25-pin female to 8-pin male PC cable (part #11658731).

Software Requirements

Your PC needs the following software:

- Microsoft Windows 3.1 or Windows 95
- Any third-party products for serial communications (purchased separately)
- Microsoft C/C++ 1.52
- The SDK.

SDK Contents

The SDK software and online documentation files are located in the directory you specified during installation and several sub-directories, described in the following table.

Sub-directory	Description
acropdf	Online documentation
dos	ROM-DOS™ files
bin	Development tools
include	Include files
lib	Library files
samples	Source code samples
font	MPCLII packets containing the printer's external base fonts
images	Pre-built ROM disk images

Related Documentation

The following table describes other documentation for the printer.

Item	Describes...
ROM-DOS User Manual	The Datalight® ROM-DOS operating system that runs on the printer.
Equipment Manual	Printer operation.
Application Notes	Technical information needed for application development (beyond writing a C/C++ program). It contains information on radio networks, memory cards, etc.
MPCLII *Packet Reference Manual*	The MPCLII printer language.
MONARCH® quick-set software Label Designer Online Help	The Label Designer. This utility creates designs for labels the application prints and translates the designs into MPCLII packets.
MONARCH® quick-set software Font Utility Online Help	The Font Utility. This utility converts TrueType and HP fonts into MPCLII font packets.

PRINTER FEATURES

2

There are several printer features that you must understand to write applications. For example, knowing the printer has a display does not help. You must know its size, how it treats messages written to it, how the rows and columns are numbered, and which functions manipulate it.

This chapter describes the following printer features:

- Display
- Keypad
- Speaker
- Memory
- Flash ROMs
- Fonts
- Scanners
- Boot Process
- Windows 95/Network Notes

Display

The printer has either a 4- or 8-row display of 20 characters each. The printer can display characters in either reverse or normal video. A display backlight is also available.

Several functions that manipulate the display use a coordinate system to access a certain point on the display. Coordinate (0,0) (row, column) is the display's upper-left corner.

Printers with 4-row displays have four virtual display pages (numbered 0-3), and printers with 8-row displays have two display pages (numbered 0-1). Each display page has its own cursor. Only one page appears on the display at a time. To ensure the

- application begins on the same display page every time, use vidSetPage to set the display page at the application's beginning.
- display pages are empty at the application's beginning, use vidScroll to clear the pages at the application's beginning and end.

An application can manipulate any display page, regardless of the current page. To manipulate the display, use the functions with the **vid** prefix (see Chapter 4, "Function Reference").

If the application writes to

- a display page other than the current one, the displayed information does not appear until the application uses vidSetPage to switch to the new display page.
- the current display page, the displayed information appears immediately.

You may want the application to write messages longer than 20 characters to the display or have a command move the cursor. If you use standard C functions such as `printf`, `puts`, and `putc` in the application, the message wraps to the next line and the cursor moves. The vid functions truncate the message if it extends beyond the twentieth column and only vidPutCursor moves the cursor.

The display is not compatible with any standard IBM PC display adapter. There is no program-accessible video memory.

Keypad

The following table describes the keypad's data entry modes.

Mode	Acceptable Keys
Numeric/Normal	Numbers, symbols, and arrows
Upper-case Alpha	Upper-case letters
Lower-case Alpha	Lower-case letters
Function Key	Function keys

Normally, when entering data, the operator must enter and exit the modes manually. However, using the Kbd functions described in Chapter 4, an application can change between the modes automatically (although the operator could still manually change modes). When you first turn the printer on, it is in Numeric/Normal mode.

Multiple modes can be in effect at the same time.

Using the trigger is a special case of Function Key mode. See "Reading Trigger Presses" in Chapter 6 for more information.

For more information about the data entry modes, refer to the Equipment Manual.

Speaker

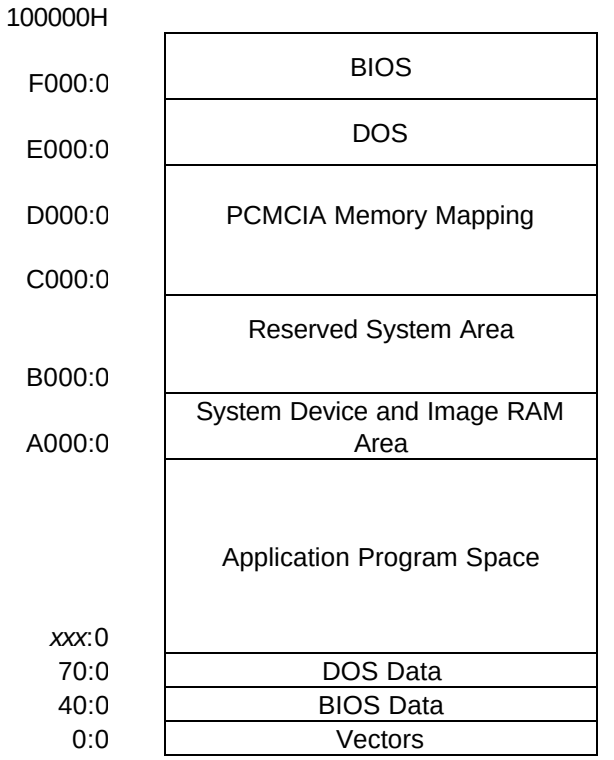
Applications can make the printer speaker beep for different lengths of time and frequencies. You may use the speaker to bring an error to the operator's attention or to indicate a good scan. The spkBeep function (described in Chapter 4) manipulates the speaker.

Memory

The printer has 1 MB of memory, excluding the two flash ROMs described later in this chapter. The application can use only the lower 640K (address 00000H to 0x9FFFF), unless you use the SDISK.SYS driver. See "SDISK.SYS" in Chapter 7 for more information.

You cannot use memory managers, HIMEM.SYS, RAM disks or load DOS high in any memory above the lower 640K.

Following is the address map.



Flash ROMs

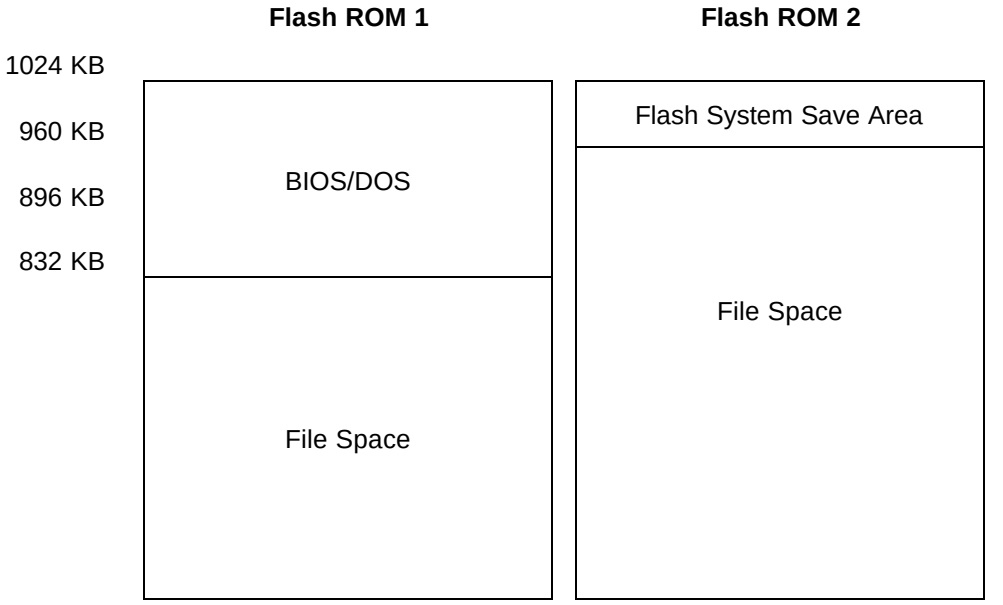
The printer also has two memory areas called flash ROMs. You load disk images (containing the application and related files) into the flash ROMs with the Flash Loader Utility.

Each flash ROM contains 1 MB, of which 832K is available for the first flash ROM and 960K for the second.

Flash ROMs are read-only, except when you load a disk image.

The printer recognizes the first flash ROM automatically. For the printer to recognize the second flash ROM, you must load RDISK.SYS in the CONFIG.SYS file. Without RDISK.SYS, you can still load a disk image into the second flash ROM, but you cannot access it.

When you have loaded disk images into the flash ROMs, you can access them like a disk drive on your PC. A: is the letter designation for the first flash ROM. B: is the letter designation for the second one (assuming you are using RDISK.SYS). Following are the flash ROM memory maps.



The printer uses 1K to 5K of the file space for the file access table. The table's size depends on the number of files in the disk image and the space they use.

For more information, see

- "Creating Disk Images" in Chapter 3
- "Loading Disk Images into the Printer" in Chapter 3
- "FLASH" in Chapter 7.

Fonts

The printer comes with a set of base fonts, one of which is internal (or resident) in the printer. The ID for the internal font (which is reduced) is 2. To use another base font or a font you have created with the Font Utility, the application must

1. Allocate memory to use for the fonts. See "PRINTINIT" in Chapter 5 for more information.
2. Initialize the Print subsystem by calling pclInit. See "pclInit" in Chapter 4 for more information.
3. Open the font file with pclOpen. The base font files are located in the SDK's Font sub-directory.

Following are the base fonts.

Font	File Name	Font ID	Point Size
CG Triumvirate Bold® (Full Character Set)	1000BOAF.PCL	1000	6.5
	1001BOAF.PCL	1001	8
	1002BOAF.PCL	1002	10
	1003BOAF.PCL	1003	12
CG Triumvirate Bold (Partial Character Set)	1004BOAS.PCL	1004	18
	1005BOAS.PCL	1005	22

Font	File Name	Font ID	Point Size
CG Triumvirate Bold Condensed (Full Character Set)	1006BOAF.PCL	1006	6.5
	1007BOAF.PCL	1007	8
	1008BOAF.PCL	1008	10
	1009BOAF.PCL	1009	12
CG Triumvirate Bold Condensed (Partial Character Set)	1010BOAS.PCL	1010	18
	1011BOAS.PCL	1011	22
Letter Gothic Bold (Full Character Set)	1012BOAF.PCL	1012	6
	1013BOAF.PCL	1013	9

The partial character set fonts contain only numeric and special characters. With fonts 1012 and 1013, the space character is only 70% as wide as the other characters.

Scanners

The printer comes with either of two bar code scanners: the 1222 or the 1223.

The 1223 scanner is more versatile than the 1222 scanner. Applications can configure it in more ways and it can scan more bar codes.

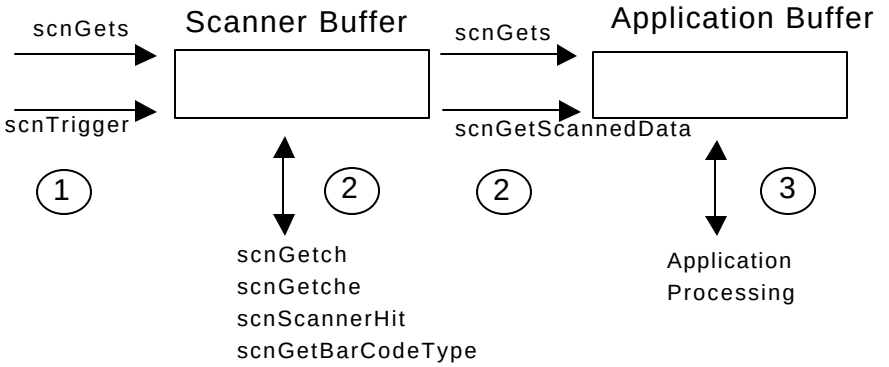
1223 scanners are compatible with scanning applications written for the 1222 scanners. The reverse is not true.

Using the Scanners

To use the either scanner, the application must

1. Enable the scanner with scnOpenScanner or scnOpenScannerShared. scnOpenScanner takes over the printer's serial port (preventing other RS-232 communications), but scnOpenScannerShared shares the port. Therefore, when using scnOpenScanner, the application should disable the scanner immediately before serial communications and enable it immediately afterwards. To avoid processing delays, use scnOpenScannerShared when possible.
2. Configure the scanner (optional). To learn how to configure the scanner, see the appropriate scn functions in Chapter 4, "Function Reference".
3. Call various scanner functions, such as scnGets.
4. Close the scanner with scnCloseScanner.

Scanner Function Overview



The scanner contains a buffer to hold the data from a scan. The application should contain a buffer of its own if it needs to save the data before scanning again. Processing works as follows:

1. Initiate the scan, putting the data in the scanner buffer with either scnGets or scnTrigger.

2. Either:

   - Process the data directly in the scanner buffer with scnGetch, scnGetche, scnGetBarCodeType, or scnScannerHit.

   - Save the data in the application buffer with scnGets (continued from step 1) or scnGetScannedData.

   The application buffer must be one byte longer than the largest string that you may scan.

   ---

3. If needed, process the data in the application buffer.

Boot Process

The printer's boot process is straightforward and flexible enough to enable you to do a variety of things as needed, such as running (or not running) AUTOEXEC.BAT or only certain lines in AUTOEXEC.BAT.

Normal Boot

When the printer boots normally:

1. The Monarch screen appears. By default, the display is disabled (nothing other than the Monarch screen appears). It becomes enabled later in this process (if the command is given). See "Display Control" for more information.

2. A line moves left to right across the display's bottom to indicate the boot's progress.

3. The printer loads CONFIG.SYS (if it exists) and runs AUTOEXEC.BAT. When AUTOEXEC.BAT does not exist, is bypassed (see “Boot Options”), or does not enable the display, the printer prompts the operator for the date and time.

If the printer never enables the display, the Monarch screen never disappears and the printer waits indefinitely for a response to the date prompt.

---

4. If there is an LCD command in AUTOEXEC.BAT that enables the display or you enable the display manually (by holding down Fct when you turn the printer on) the printer enables the display.

5. The Monarch screen disappears and the DOS prompt appears.

Display Control

You can either enable or disable the display.

Enabling the Display

To enable the display, either

- hold down Fct while turning the printer on. When messages appear on the display, release the key.

If you continue to hold down Fct after the messages appear, the display moves to fast mode.

- include the LCD command in AUTOEXEC.BAT. CONFIG.SYS messages are still disabled, because the printer processes it first. See “LCD” in Chapter 7 for more information.

Disabling the Display

To disable the display, include the LCD N command in AUTOEXEC.BAT. See “LCD” in Chapter 7 for more information.

Boot Options

When DOS begins loading, you have the following options for processing CONFIG.SYS and AUTOEXEC.BAT:

- Bypass the files
- Be prompted whether to process each line of both files
- Process every line of both files.

To choose the boot options:

1. Turn on the printer while holding Fct down until messages appear. You must do this step. Otherwise, the date and time prompts (the default) wait for input without appearing on the screen.
2. When "Starting ROM-DOS..." appears, the printer beeps. You have approximately two seconds to do one of the following:
   - Press Fct, then 5 to bypass both files. The boot process continues with default configurations. The printer also prompts you for the date and time.
   - Press Fct, then 8 to request a prompt at each line of both files. Enter Y to process the line or N to bypass it.
   - Do nothing and let the printer process every line of both files.

Booting from the PC

To boot the printer from the PC:

1. Gather the necessary boot files (see "Creating Disk Images" in Chapter 3) on a floppy disk or the root partition of a non-booting hard drive.
2. Run REMSERV from the PC, using the boot files' location as the shared drive.

3. Turn on the printer while holding Alt down until boot messages display. The printer mounts a backup drive (A:) in memory. The printer beeps twice when it starts to boot from the PC. The booting continues using the default configurations. The printer also prompts you for the date and time.

The PC boot drive becomes drive B: on the printer.

Windows 95/Network Notes

Directory Names

The printer does not recognize directory names longer than 8 characters, not including the extension.

Copying Data from the Printer to PC

The procedure to copy data from the printer to a PC varies, depending on your system.

FAT32, NTFS, and Networks

1. Copy the file from the printer to a diskette.
2. Copy the file from the diskette to the PC's hard drive.

FAT16 File Systems

1. Reboot your PC in DOS mode. You must reboot. For this procedure to work, you cannot go to the DOS shell.
2. Connect the cable between the PC and the printer.
3. Lock the PC's hard drive by entering the DOS lock command at the DOS prompt. Enter Y at the "Are You Sure (Y/N)?" prompt.
4. Run Remserv on the PC, specifying the locked drive.
5. Run Remdisk on the printer.

6. Copy the file(s) from their location on the printer to the printer's mapped drive. Include a complete path when copying. The default is that the file is saved in the root directory.

7. Exit Remserv.

8. Unlock the PC's hard drive by entering the DOS unlock command at the DOS prompt.

The file(s) you copied are stored in the PC's root directory.

DEVELOPING APPLICATIONS 3

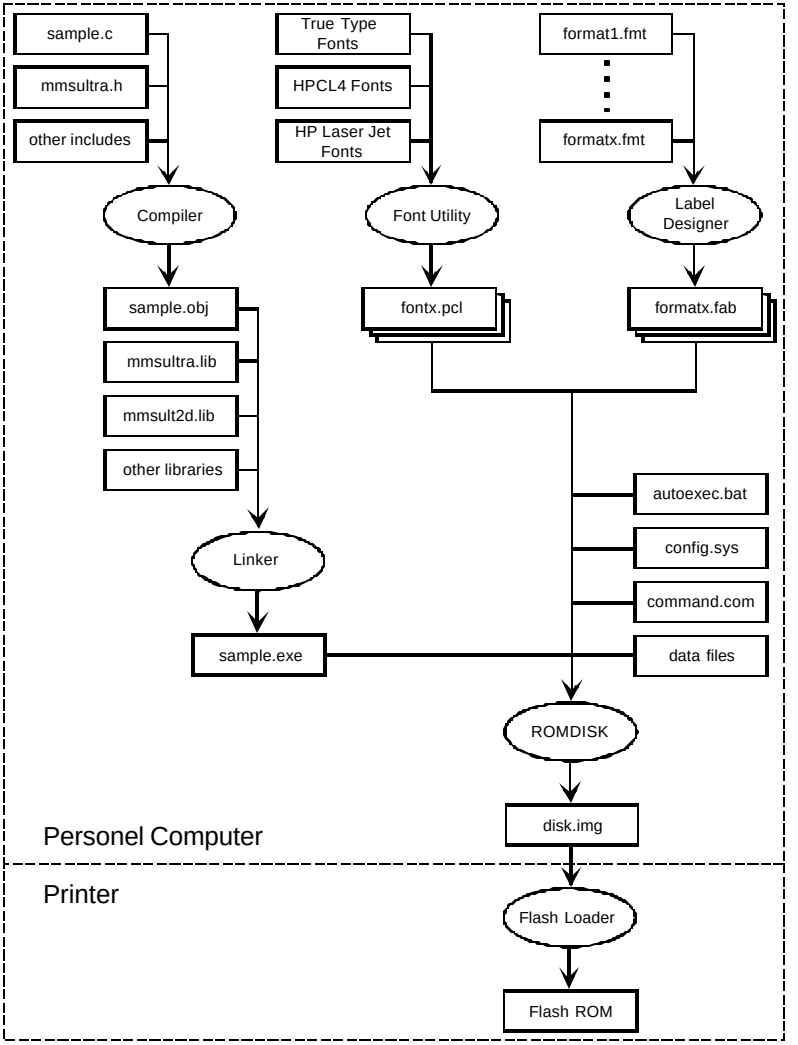
Developing applications is a long, detailed process. You cannot just write the application in Microsoft C/C++. You must also understand the whole picture. For example, you must create MPCLII packets and assess the application's memory needs.

This chapter describes this process. Following is a summary:

1. Create any needed MPCLII packets (if your application prints labels).
2. Assess the application's memory needs.
3. Write the application.
4. Build (compile and link) the application.
5. Run REMSERV on the PC and REMDISK on the printer to enable the printer to access a disk drive on the PC.
6. Test the application with the printer attached to the PC.
7. Create a disk image.
8. Load the disk image into the printer's flash ROM.
9. Quit REMSERV and REMDISK.
10. Test the application with the printer detached from the PC.

Overview

The following diagram is an overview of this process.



Creating MPCLII Packets

An application prints labels by submitting MPCLII packets to the printer. You can create these packets manually with the MPCLII Packet Reference Manual or automatically with Label Designer.

To create the packets in Label Designer, select **Save As MPCL...** from the File Menu after completing the label.

To test your packets without writing an application, use the ONLINE Utility. See “ONLINE” in Chapter 7 for more information.

Assessing Memory Needs

When you write your application, you must determine the application’s memory needs. This information helps you decide, for example, if you need a memory card to supplement the printer’s memory.

The following table lists the approximate amount of memory necessary for some items you might use with your application. The sizes can vary depending on the version of software you are using and the amount of memory they use. For example, memory requirements can vary depending on the FILES, FCBS, BUFFERS, STACKS, and LASTDRIVE settings in CONFIG.SYS.

Item	Component	Size (in bytes)	Total
DOS	System Data	20,000	
	COMMAND.COM	4,096	24,096
Second Flash ROM	RDISK.SYS	2,496	2,496
Static RAM Disk	SDISK.SYS	4,512	4,512
ATA Memory Card	SSELAN.EXE	4,624	
	CS.EXE	44,800	
	CSALLOC.EXE	16	
	ATADRV.EXE	11,040	
	CARDID.EXE	21,056	81,536
Aironet Radio Card with NDIS Driver	SSELAN.EXE	4,624	
	CS.EXE	44,800	
	CSALLOC.EXE	16	
	AWC2N3K.DOS	14,784	64,224
Remote Disk	REMDISK.EXE	11,648	11,648
Libraries	No 2D Symbologies	113,000	113,000
	2D Symbologies	135,000	135,000
	Maxicode	131,000	131,000
	Scanning	8,570	8,570
Third Party Drivers	NetBIOS	48,000	48,000

Example

Following is a sample configuration that uses the printer's printing, scanning, and radio capabilities:

Item	Size (in bytes)
DOS	24,096
Second Flash ROM	2,496
Static RAM Disk	4,512
Radio	64,224
Third-Party NETBIOS	48,000
Printing (standard library)	113,000
Scanning	8,570
Total	264,898
Available for Application (640K – Total)	375,102

Writing Applications

Although you can write your application using any 16-bit compiler, the libraries included in the SDK only work with Microsoft C/C++.

If you are using Visual C++, specify MS-DOS® application (.EXE) for the project type.

Building Applications

A build consists of compiling and linking the application's source code. If your application has multiple source code files, you must compile and link separately. For applications with one source code file, you must compile and link in one step.

To build, use either the menu choices in the C/C++ development environment or the following DOS prompt commands:

- cl (to compile and/or link)
- link (to link only).

Refer to your C/C++ documentation for more information.

Compiling Applications

Although you may also use other options when compiling, you must specify

- /AL (use the large memory model)
- /Zp1 (pack structure members)
- /c (use only with multiple source code files).

Include Files

There are two include files that you can use: MMSULTRA.H and SCAN1223.H. You must always use MMSULTRA.H. Use SCAN1223.H only when you link the application to SCAN1223.LIB.

If you use both the SCAN1223.H and MMSULTRA.H files in your application, place the #include directive for SCAN1223.H before the one for MMSULTRA.H. It does not matter where you place the #include directives for the standard C include files.

Linking Applications

Although you may also use other options when linking, you must specify the

- object files
- executable file
- map file
- libraries you are using.

Libraries

The SDK comes with the following libraries:

Name	Printer Support	Scanner Support
MMSULTRA.LIB	One-dimensional bar codes.	1222 scanner and anything common between the 1222 and 1223 scanners.
MMSULT2D.LIB	All bar codes, except Maxicode.	1222 scanner and anything common between the 1222 and 1223 scanners.
MMSULTMX.LIB	One-dimensional bar codes and Maxicode.	1222 scanner and anything common between the 1222 and 1223 scanners.
SCAN1223.LIB	None. For applications that print and scan with a 1223 scanner, use this library with one of the other libraries listed above, listing this library first when linking.	1223 scanner only. You must use this library if you scan any bar codes unique to the 1223 scanner. Include this library only when you need it.

You can also link to any standard C/C++ libraries.

Using REMSERV and REMDISK

To enable the printer to access a disk drive on the PC:

1. Connect the printer and the PC with a PC cable (see "Hardware Requirements" in Chapter 1). Attach one end of the cable to the serial port on the printer and the other end to the serial port on the PC.

2. Run the REMSERV Utility on the PC and the REMDISK Utility on the printer. You can start either utility first. Both utilities must use the same baud rate and transmission style (packet/non-packet).

   See "REMSERV" and "REMDISK" in Chapter 7 for more information.

3. When you run REMDISK, it specifies the printer drive letter corresponding to the shared PC drive. On the printer, use this letter and any DOS command to access the PC's disk drive.

Testing the Application

Although you can do some limited testing on the PC, you test mostly on the printer. You can

- load and run the application on the printer. If you choose this option, continue with "Creating Disk Images."

- run the application on the printer even though the executable file resides on the PC. If you are going make many changes to the application, this option is quicker and easier. To do so:

  1. Connect one end of a cable (see "Hardware Requirements" in Chapter 1) to the printer and the other end to the PC.

  2. Run the REMSERV on the PC.

  3. Run the REMDISK on the printer.

  4. On the printer, change to the shared PC drive. Then, maneuver to the directory containing the executable file.

  5. At the printer's DOS prompt, start the application.

Running the application on the printer when the executable file resides on the PC may result in several extra seconds to start the application.

Creating Disk Images

Before you can load the application into the printer, you must create a disk image. To do so, gather the files required by the application in a separate directory. This directory may have sub-directories. Then, you must run the ROMDISK Utility. See "ROMDISK" in Chapter 7 for more information.

CAUTION

Be **very** careful when gathering the files to include in your disk image. You must include all files needed to boot the printer and run the application.

Disk image file sizes cannot be larger than 832K (for the first flash ROM) and 932K (for the second flash ROM).

If the second flash ROM does not contain a DOS file system (even an empty one), the printer does not load the disk image.

DOS assigns drive letters in the order you specify the entries in CONFIG.SYS. If you use REMDISK outside of CONFIG.SYS, DOS assigns it the next letter after the ones specified in CONFIG.SYS.

Using the First Flash ROM Only

The first flash ROM requires the following files (others are optional):

- COMMAND.COM
- REMDISK.EXE
- LCD.COM
- AUTOEXEC.BAT (at a minimum, this file must contain one line containing `LCD Y C`).

Using Both Flash ROMs

To use both flash ROMs, include CONFIG.SYS and RDISK.SYS in the first flash ROM's disk image. CONFIG.SYS must contain a reference to RDISK.SYS. See "RDISK.SYS" in Chapter 7 for more information.

Loading Disk Images into the Printer

To load the disk image created with ROMDISK into a flash ROM:

1. Connect one end of a cable (see "Hardware Requirements" in Chapter 1) to the printer and the other end to the PC.
2. Run the REMSERV on the PC.
3. Run the REMDISK on the printer.
4. On the printer, change to the shared PC drive. Enter the Flash command at the DOS prompt on the printer. See "FLASH" in Chapter 7 for more information.
5. At the OK TO PROCEED prompt, enter Y.
6. Set the printer down and wait for the loading procedure to finish. When "PROGRAM SUCCESSFUL" appears on the display, press Enter.

CAUTION

Do Not disturb the printer while loading a disk image.

7. Repeat steps 4 through 6 for the second flash ROM, if applicable.
8. Re-boot the printer.

Recovering from a Disk Image Loading Problem

If the cable connection becomes broken, re-establish the REMSERV/REMDISK link and start over. If this action fails or you lose power, the disk image is corrupt and you must boot the printer from the PC. If the disk images in both flash ROMs become corrupt, you must return the printer to Monarch for service. See "Booting from the PC" in Chapter 2 for information.

Exiting REMSERV and REMDISK

To quit REMSERV:

1. Change the printer's current drive to one other than the shared PC drive.

2. Press any key on the PC. Or, enter the REMQUIT command on the printer (if you included that utility in the disk image). The printer can no longer access the drive until you restart REMSERV.

To quit REMDISK, run it again with the /U option. See "REMDISK" in Chapter 7 for more information.

Testing the Application Away from the PC

It is good practice to re-test your application when you have loaded it into the printer and the printer is detached from the PC.

Training the End Users

The last step in the development of an application is to train the end users (Operators) and/or their supervisor (System Administrator). Depending on the application's complexity, this training may include a class, written instructions, or any other appropriate format.

The Operators and System Administrators must know how to use the application. They also must know how to perform routine procedures (loading supplies, for example) that may vary from descriptions in the Equipment Manual. You must give the Operators and/or System Administrators more specific instructions.

FUNCTION REFERENCE

4

The SDK contains several libraries of functions you can call in your application. This chapter describes these functions. It lists them alphabetically.

The functions are divided into the following categories:

Prefix	Description
Kbd	Keypad Interface
pcl	Printing Interface
scn	Scanning Interface
spk	Speaker Interface
sys	System Interface
vid	Video Interface

All functions in the same category begin with the same prefix.

The function names are case-sensitive.

KbdClrFunct

Description

Changes the keypad's data entry mode to the mode in effect immediately before the application called KbdSetFunct.

Syntax

```
void KbdClrFunct(void);
```

Parameters

None

Return Values

None

Example

See "KbdSetFunct" for an example.

KbdGetMode

Description

Checks if Function Key mode is set or saves the keypad mode (Numeric/Normal, Upper-case Alpha, or Lower-case Alpha) until the application calls KbdRestoreMode.

Syntax

```
int KbdGetMode(void);
```

Parameters

None

Return Values

1 Numeric/Normal mode

2 Upper-case Alpha mode

4 Lower-case Alpha mode (4-line/33-key version only)

9 Function Key mode is set while Numeric/Normal mode is in effect.

10 Function Key mode is set while Upper-case Alpha mode is in effect.

12 Function Key mode is set while Lower-case Alpha mode is in effect (4-line/33-key version only).

Example

See “KbdRestoreMode” for an example.

KbdRestoreMode

Description

Changes the keypad's data entry mode to the one saved previously when the application called KbdGetMode.

Syntax

```
void KbdRestoreMode(int);
```

Parameters

1 Numeric/Normal mode

2 Upper-case Alpha mode

4 Lower-case Alpha mode (4-line/33-key version only)

Return Values

None

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
  int iSavedmode = 0;                  // Saved mode
  int iInput = 0;                      // Alpha key entered

  iSavedmode = KbdGetMode();           // Save the current mode
                                       // To Upper-case Alpha
  KbdSetCaps();
  printf("Press an alphabetic\nkey:  "); // Test the mode
  while (_kbhit())
    ;
  iInput = _getch();
  printf("\nYou pressed %c\n", iInput);
                                       // Return to prev. mode
  KbdRestoreMode(iSavedmode);
  printf("\nPress the same key: ");    // Test the mode
```

```
    while (_kbhit())
        ;
    iInput = _getch();
    printf("\nYou pressed %c\n", iInput);
}
```

KbdSetAlpha

Description

4-line/33-key Printer Only. Changes the keypad's data entry mode to Lower-case Alpha mode.

Syntax

```
void KbdSetAlpha (void);
```

Parameters

None

Return Values

None

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
  int iSavedmode = 0;                  // Saved mode
  int iInput = 0;                      // Key entered

  KbdSetAlpha();                       // Set Lower-case Alpha
  iSavedmode = KbdGetMode();           // Save the current mode
  KbdSetCaps();                        // Set Upper-case Alpha
  printf("Press an alphabetic\nkey: "); // Test the mode
  while (_kbhit())
    ;
  iInput = _getch();
  printf("\nYou pressed %c\n", iInput);
  KbdRestoreMode(iSavedmode);          // Return to prev. mode
  printf("\nPress the same\nkey: ");   // Test the mode
  while (_kbhit())
    ;
  iInput = _getch();
```

```
    printf("\nYou pressed %c\n", iInput);
}
```

KbdSetCaps

Description

Changes the keypad's data entry mode to Upper-case Alpha mode.

Syntax

```
void KbdSetCaps(void);
```

Parameters

None

Return Values

None

Example

See "KbdSetAlpha" for an example.

KbdSetFunct

Description

Changes the keypad's data entry mode to Function Key mode.

Syntax

```
void KbdSetFunct(void);
```

Parameters

None

Return Values

None

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
  int iInput = 0;                          // Prompted input
  int iNum = 0;                            // # of labels to print

  KbdSetNormal();                          // Set Normal mode
  KbdSetFunct();                           // Set Function Key mode
  printf("Press 5 to print\nlabels\n");    // Get input (F5)
  iInput = _getch();
  if (iInput == 0x00)
  {
    KbdClrFunct();
    iInput = _getch();
    if (iInput == 0x3F)
    {                                      // Get # of labels
      printf("How many labels do\nyou need?");
      iNum = _getch();
      printf("\nPrinting %c labels...", iNum);
      /* Branch to printing routine */
    }
```

```
    }
}
```

KbdSetNormal

Description

Changes the keypad's data entry mode to Numeric/Normal mode.

Syntax

```
void KbdSetNormal(void);
```

Parameters

None

Return Values

None

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  char cName[25];              // Entered name
  char cNumber[4];             // Entered number
                               // Set Upper-case Alpha mode
  KbdSetCaps();
  printf("Enter Operator Name:");      // Enter operator name
  gets(cName);
  KbdSetNormal();              // Set Normal mode
  printf("Enter Operator #:");         // Enter operator number
  gets(cNumber);
  /* branch to name and number processing routine */
}
```

pclBatteryOkToPrint

Description

Checks if the printer's NiCd battery (located in the handle) is charged enough to allow printing. It is good programming practice to check the battery level before doing any printing.

Use this function immediately prior to printing, but not during printing. If you use it during printing, the return value is not accurate.

Syntax

```
unsigned short far pclBatteryOkToPrint(void);
```

Parameters

None

Return Values

0	The battery level is too low to allow printing.
Non-zero	The battery level is high enough to allow printing.

Example

See "pclGetOnDemandSensor" for an example.

pclCalibrate

Description

Calibrates the supplies in the printer and gives the supply information to the Print subsystem. You have the option of writing other functions to prompt the operator to override the calibrated values before the supply information is given to the Print subsystem.

Operators can load supplies (as described in the Equipment Manual) before running an application, but they cannot calibrate the supplies until the application calls this function. In general, you should display a prompt ("Load your supplies," for example) and require the operator to press a key (the trigger might be easiest) prior to calling this function.

Do not use this function when using fax paper because it has no black mark to detect.

If an application uses this function, it should not use pclCalibratePaper and pclPaperSetup.

Syntax

```
unsigned short far pascal pclCalibrate(
                    unsigned short usStockLength,
                    unsigned short usStockWidth,
                    unsigned short usStockType,
                    LPFNSUPPLYTYPEPROMPT
                          lpfnSupplyTypePrompt,
                    LPFNSUPPLYPROMPT lpfnSupplyPrompt);
```

Parameters

usStockLength	The supply length in hundredths of an inch. Values are *55-400*.
usStockWidth	The supply width in hundredths of an inch. Values are *120*, *150*, and *200*.
usStockType	The supply type. Values are *MMS_LOW_ENERGY* Paper *MMS_MEDIUM_ENERGY* Fax *MMS_HIGH_ENERGY* Synthetic If you are using linerless supplies, experiment with these values to see which one achieves the best results.
lpfnSupplyTypePrompt	A pointer to an optional programmer-written callback function for supplying stock type information.
lpfnSupplyPrompt	A pointer to an optional programmer-written callback function for supplying stock length and width information.

For lpfnSupplyTypePrompt and lpfnSupplyPrompt, you can enter

- zero
- a function address.

If the value is zero, pclCalibrate prompts the operator for the necessary supply parameters. If the entered values are valid, pclCalibrate uses them and does not prompt the operator. If they are invalid, pclCalibrate re-prompts the operator. For example, if the printer uses the same supply width and supply type, but different lengths, it could call pclCalibrate with valid width and supply type values and an invalid length. Then, pclCalibrate prompts the operator for the length.

If the value is a function address, pclCalibrate calls the corresponding callback function to prompt the operator for the supply parameters. If the operator enters illegal values, the callback function should re-prompt the operator.

The callback functions must return zero for success. Otherwise, pclCalibrate aborts the calibration process.

Declare the callback functions as shown below. Each parameter points to variables containing the calibrated values.

```
unsigned short far pascal SupplyTypePrompt(
    unsigned short far *lpusPaperType);        // The stock type

unsigned short far pascal SupplyPrompt(
    unsigned short far* lpusLenInches,         // Length in 1/100"
    unsigned short far* lpusWidthInches);      // Width in 1/100"
```

You can name the functions anything you want.

Return Values

0	Successful.
Non-zero	An error occurred. For errors between 703-793, the operator corrects the printer condition. Then, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "mmsultra.h"

unsigned short far pascal StockTypePrompt
  (unsigned short far * lpusPaperType);
```

```
unsigned short far pascal StockPrompt
  (unsigned short far * lpusLenInches,
   unsigned short far * lpusWidthInches);

void main()
{
  PRINTINIT rConfig;
  unsigned short usStatus = 0;

  /* Initialize printer with no font storage */
  rConfig.lpuchFntAddr = 0;
  rConfig.ulFntSize = 0;
  usStatus = pclInit(&rConfig);
  if (usStatus != 0)
  {
    printf("Init Failed \nError: %d", usStatus);
    exit(1);
  }

  /* Calibrate with the function doing all the prompts */
  usStatus = pclCalibrate(0xFFFF, 0xFFFF, 0xFFFF, 0, 0);
  if (usStatus != 0)
  {
    printf("Calibrate Failed \nError: %u", usStatus);
    pclClose();
    exit(1);
  }

  /* Calibrate with constant 2" width and paper type */
  /* Let function prompt for the stock length */
  usStatus = pclCalibrate(0xFFFF, 200, MMS_LOW_ENERGY, 0, 0);
  if (usStatus != 0)
  {
    printf("Calibrate Failed \nError: %u", usStatus);
    pclClose();
    exit(1);
  }

  /* Calibrate paper but use our callback function to prompt */
  usStatus = pclCalibrate(0, 0, 0, StockTypePrompt,
StockPrompt);
```

```
  if (usStatus != 0)
  {
    printf("Calibrate Failed\nError: %u", usStatus);
    pclClose();
    exit(1);
  }
  pclClose();
  exit(0);
}

  /* Calibration callback function to prompt for paper type */
unsigned short far pascal StockTypePrompt
  (unsigned short far * lpusPaperType)
{
  short sCols, sPages, sKey;

  for (;;)                                    // loop doing ...
  {                                           // clear screen
    vidSetMode(vidGetState(&sCols, &sPages));
    printf("Enter Stock Type\n(0-2):\n");     // display prompt
    sKey = _getch();                          // get key
    if (sKey == 0)                            // if extended key
      _getch();                               // clear it out
    else if (sKey >= '0' && sKey <= '2')      // if valid type
      break;                                  // stop prompting
  }
  *lpusPaperType = (unsigned short) sKey;     // save the setting
  return(0);                                  // return success
}

/* Calibration callback function to always return constant size */
unsigned short far pascal StockPrompt
  (unsigned short far * lpusLenInches,
   unsigned short far * lpusWidthInches)
{
  *lpusLenInches = 200;
  *lpusWidthInches = 150;
  return(0);                                  // return success
}
```

pclCalibratePaper

Description

Calibrates supplies in the printer. After the application calls this function, you have the option to have it call a function you write to prompt the operator to override the calibrated values. Then it must call pclPaperSetup.

Operators can load supplies (as described in the Equipment Manual) before running an application, but they cannot calibrate the supplies until the application calls this function. In general, you should display a message ("Load your supplies," for example) and require the operator to press a key (the trigger might be easiest) prior to calling this function.

Do not use this function when using fax paper because it has no black mark to detect.

If an application uses this function and pclPaperSetup, it should not use pclCalibrate.

Syntax

```
unsigned short far pclCalibratePaper(
                    unsigned short far* lpusStockLength,
                    unsigned short far* lpusStockWidth);
```

Parameters

lpusStockLength	Pointer to the calibrated supply length in hundredths of an inch. Returned values are *55-400*.
lpusStockWidth	Pointer to the calibrated supply width in hundredths of an inch. Returned values are *0* (not calibrated), *120*, *150*, or *200*.

Return Values

0	Successful.
Non-zero	An error occurred. For errors between 703-793, the operator corrects the printer condition. Then the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

See “pclPaperSetup” for an example.

pclClearError

Description

Resets the Motion Control subsystem after an application receives a motion control error (703-793).

Of the pcl functions, only pclGetErrorMsg cannot generate a motion control error.

The operator must correct the printer condition (a supply jam, for example) before the application calls this function.

Syntax

```
void far pclClearError(void);
```

Parameters

None

Return Values

None

Example

See “pclFeed” for an example.

pclClose

Description

Closes the Print subsystem by freeing all internally allocated resources.

If the application allocates font storage memory with pclOpen, it must free that memory with the appropriate statement immediately after calling pclClose.

A call to this function must occur only once (at the application's end). If the application does not call it, the printer locks up.

Syntax

```
void far pclClose(void);
```

Parameters

None

Return Values

None

Example

See "pclInit" for an example.

pclFeed

Description

Feeds a label through the printer.

Syntax

```
unsigned short far pclFeed(void);
```

Parameters

None

Return Values

0	Successful.
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
  PRINTINIT pConfig;                    // Print data structure
  unsigned short usStatus = 0;          // Battery level
  short sStatus = 0;                    // Command call status

  pConfig.lpuchFntAddr = 0;             // Start Print subsystem
  pConfig.ulFntSize = 0;
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
  {
    usStatus = pclGetBatteryLevel();    // Get the battery level
```

```
    if (usStatus > 711)                    // If level OK,
    {
      usStatus = pclFeed();                // Feed supplies
      if (usStatus != 0)
      {
        printf("Feed Error-- press any key when printer is reset.");
        _getch();
        pclClearError();
      }
    }
    else
      printf("Charge battery");            // Display low level msg
  }
  pclClose();                              // Close Print subsystem
}
```

pclGetBatteryLevel

Description

Retrieves the NiCd battery's level. This battery is located in the printer's handle. It is good programming practice to check the battery level before any processing.

Use this function immediately prior to printing, but not during printing. If you use this function during printing, the return value is not accurate.

Syntax

```
unsigned short far pclGetBatteryLevel(void);
```

Parameters

None

Return Values

<= *711*	You must charge the battery.
712-831	The battery level is high enough to run the printer, but not print.
>= *832*	The battery level is high enough to run the printer and print.

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  PRINTINIT pConfig;                    // Print data structure
  short sStatus = 0;                    // Status of comm. calls
  unsigned short usStatus = 0;          // Battery level

  pConfig.lpuchFntAddr = 0;             // Start Print subsystem
  pConfig.ulFntSize = 0;
```

```
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
  {
    usStatus = pclGetBatteryLevel();     // Get the battery level
    if (usStatus <= 711)                 // Display results
      printf("Charge the battery");
    else
      if (usStatus >= 832)
        printf("Can run and print");
      else
        printf("Can run/cannot print");
  }
  pclClose();                            // Close Print subsystem
}
```

pclGetBlackMarkSensor

Description

Retrieves the black mark sensor's latest state. This state is not necessarily the current state because it is updated only by the Print subsystem.

Syntax

```
unsigned short far pclGetBlackMarkSensor(void);
```

Parameters

None

Return Values

1	The supplies are aligned on the black mark.
0	The supplies are not aligned on the black mark, or the Print subsystem is busy or uninitialized.
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  PRINTINIT pConfig;                    // Print data structure
  unsigned short usStatus = 0;          // Battery level
  short sStatus = 0;                    // Command calls status

  pConfig.lpuchFntAddr = 0;             // Start Print subsystem
  pConfig.ulFntSize = 0;
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
```

```
    printf("Init Failed\nError: %d", usStatus);
  else
  {
    usStatus = pclGetBatteryLevel();          // Check battery
    if (usStatus <= 711)
      printf("Charge your battery");
    else
    {
      usStatus = pclGetBlackMarkSensor();     // Get sensor state
      switch (usStatus)                       // Display result
      {
        case 1:  printf("Supplies are aligned");
                 break;
        case 0:  printf("Supplies misaligned or system error");
                 break;
        default: printf("Error-- press any key when reset.");
                 _getch();
                 pclClearError();
      }
    }
  }
  pclClose();                                 // Close Print subsystem
}
```

pclGetErrorMsg

Description

Testing/Debugging Only. Retrieves a pointer to the error message corresponding to a specified error number. Use this function on error numbers returned by pclOpen, pclWrite, and pclStatus.

Many error messages are longer than 20 characters. An application must format the message before displaying it to avoid truncation. See “Display” in Chapter 2 for more information.

Error numbers that this function can retrieve must be positive. Zero indicates success, and negative numbers are ROM-DOS errors.

Syntax

```
char far* far pclGetErrorMsg(unsigned short usMsgNo);
```

Parameters

usMsgNo	The error number.

Return Values

Pointer to a null-terminated message string	Successful.
0	Unsuccessful.

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  short usStatus = 0;                   // Command calls status
```

```
  char far* cStatus = NULL;              // Error message
  PRINTINIT pConfig;                     // Print data structure

  pConfig.lpuchFntAddr = 0;              // Start Print subsystem
  pConfig.ulFntSize = 0;
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
  {
    if (!pclBatteryOkToPrint())          // Check battery
      printf("Low battery error");
    else
    {                                    // Open file
      usStatus = pclOpen("B:\\MPCLII\\TEST.FAB");
      if (usStatus > 0)
      {
        cStatus = pclGetErrorMsg(usStatus);
        printf("%Fs\n", cStatus);
      }
    }
  }
  pclClose();                            // Close Print subsystem
}
```

pclGetOnDemandSensor

Description

Determines the on-demand sensor's current state.

Syntax

```
unsigned short far pclGetOnDemandSensor(void);
```

Parameters

None

Return Values

1	The sensor is blocked.
0	The sensor is not blocked.
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include "mmsultra.h"

void main(void)
{
  PRINTINIT pConfig;                    // Print data structure
  short sStatus = 0;                    // Command calls status
  char cFormat[100];                    // Format to print

  pConfig.lpuchFntAddr = 0;             // Start Print subsystem
  pConfig.ulFntSize = 0;
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
```

```
  {
    if (!pclBatteryOkToPrint())                    // Check battery
        printf("Battery too low");
    else
    {                                              // Write format
      strcpy(cFormat, "{F,1,A,R,E,400,200,\"1C39\"|");
      strcat(cFormat, "B,1,12,F,320,29,4,12,20,8,L,0|");
      strcat(cFormat, "R,1,\"666666666666");
      strcat(cFormat, "\"|}");
      sStatus = pclWrite(cFormat, strlen(cFormat));
      if (sStatus != 0)
        printf("Format Write error - %d", sStatus);
      else
      {                                            // Start batch
        sStatus = pclWrite("{B,1,N,1|E,0,0,1,1,0,1|}", 24);
        if (sStatus != 0)
          printf("Batch Write error - %d", sStatus);
        else
        {
          while ((sStatus = pclStatus()) == 1)  // Wait until done
            ;
          sStatus = pclGetOnDemandSensor(); // Check sensor
          switch (sStatus)        // Display result
          {
            case 0:  printf("\nNot blocked");
                     break;
            case 1:  printf("\nBlocked");
                     break;
            default: printf("Error-- press any key when reset.");
                     _getch();
                     pclClearError();
          }
        }
      }
    }
  }
  pclClose();                                  // Close Print subsystem
}
```

pclGetSupplyType

Description

Retrieves the current supply type.

Syntax

```
unsigned short far pclGetSupplyType(void);
```

Parameters

None

Return Values

MMS_LOW_ENERGY	Paper
MMS_MEDIUM_ENERGY	Fax
MMS_HIGH_ENERGY	Synthetic
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  unsigned short usType = 0;          // Supply type
  PRINTINIT pConfig;                  // Print data structure
  unsigned short usStatus = 0;        // Battery level
  short sStatus = 0;                  // Command calls status

  pConfig.lpuchFntAddr = 0;           // Start Print subsystem
  pConfig.ulFntSize = 0;
```

```
  sStatus = pclInit(&pConfig);
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
  {
    usStatus = pclGetBatteryLevel();
    if (usStatus <= 711)                    // Check battery
      printf("Charge your battery");
    else
    {
      usType = pclGetSupplyType();          // Get supply type
      switch (usType)                       // Display result
      {
        case MMS_LOW_ENERGY:
                        printf("Using regular paper");
                        break;
        case MMS_MEDIUM_ENERGY:
                        printf("Using fax paper");
                        break;
        case MMS_HIGH_ENERGY:
                        printf("Using synthetic paper");
                        break;
        default:        printf("Error-- press any key when reset.");
                        _getch();
                        pclClearError();
      }
    }
  }
  pclClose();                               // Close Print subsystem
}
```

pclInit

Description

Initializes the Print subsystem with normal memory. An application must call this function before calling any other pcl functions.

Before calling this function:

1. If you are using a font other than the internal font (ID #2), allocate memory for font storage (the amount needed varies by font). See "Font Memory Requirements" in Chapter 5 for more information.

   Use any C/C++ statement to allocate/de-allocate the memory you need.

   ---

2. Set the values for the fields in the PRINTINIT data structure. Enter zero in both fields if you are not using downloadable fonts.

   Initialize and close the Print subsystem only once in the application.

Syntax

```
short far pclInit(LPPRINTINIT lprPrintInit);
```

Parameters

lprPrintInit Pointer to a PRINTINIT data structure.

Return Values

0	Successful.
-1	Memory allocation error.
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <conio.h>
#include <malloc.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  PRINTINIT pPrintparms;                    // Print data structure
  short sStatus = 0;                        // Status of comm. calls

                                            // Start Print subsystem
  pPrintparms.lpuchFntAddr = (unsigned char far
*)malloc(16384);
  pPrintparms.ulFntSize = 16384;
  sStatus = pclInit(&pPrintparms);
  switch(sStatus)
  {
    case 0:  ; /* Branch to printing routine */
             break;
    case -1: printf("Memory Allocation Error");
             break;
    default: printf("Error-- press any key when reset.");
             _getch();
             pclClearError();
  }
  pclClose();                               // Close Print subsystem
  free(pPrintparms.lpuchFntAddr);           // Free allocated memory
}
```

pclOpen

Description

Loads MPCLII packets from the specified file. Generally you use this function for fixed packets, such as formats. An application can call this function as often as needed. It is not necessary to open a particular file more than once in an application.

Syntax

```
short far pclOpen(char far* lpszFileName);
```

Parameters

lpszFileName The fully-qualified path for the file containing the MPCLII packets. If it is in the same directory as the application's .EXE file, specify only the file name.

You must use the drive that the printer recognizes. For example, if the files are on the PC's C: drive, but the printer refers to it as B:, use B: in the path.

Return Values

0 Successful.

Non-zero An error occurred. For errors between 703-793, the operator corrects the printer condition. Then the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

See "pclGetErrorMsg" for an example.

pclPaperInfo

Description

Retrieves information (length, width, and type) about the current supplies.

Syntax

```
void far pclPaperInfo(unsigned short far *lpusStockLength,
                      unsigned short far *lpusStockWidth
                      unsigned short far *lpusStockType);
```

Parameters

lpusStockLength Pointer to the supply length in hundredths of an inch. Returned values are *55-400*.

lpusStockWidth Pointer to the supply width in hundredths of an inch. Returned values are *120*, *150*, and *200*.

lpusStockType Pointer to the supply type. Returned values are

MMS_LOW_ENERGY	Paper
MMS_MEDIUM_ENERGY	Fax
MMS_HIGH_ENERGY	Synthetic

If you are using linerless supplies, experiment with these values to see which one achieves the best results.

Return Values

None

Example

See “pclPaperSetup” for an example.

pclPaperSetup

Description

Gives information about the supplies being used to the Print subsystem.

If you are using paper or synthetic supplies, precede this function with a call to pclCalibratePaper.

If an application uses this function and pclCalibratePaper, it should not use pclCalibrate.

Syntax

```
unsigned short far pclPaperSetup(unsigned short usStockLength,
                                 unsigned short usStockWidth,
                                 unsigned short usStockType);
```

Parameters

usStockLength The supply length in hundredths of an inch. Values are *55-400*.

usStockWidth The supply width in hundredths of an inch. Values are *120*, *150*, and *200*.

usStockType The supply type. Values are

MMS_LOW_ENERGY	Paper
MMS_MEDIUM_ENERGY	Fax
MMS_HIGH_ENERGY	Synthetic

If you are using linerless supplies, experiment with these values to see which one achieves the best results.

Return Values

0	Successful.
703-793	A motion control error occurred. For errors between 703-793. The operator corrects the printer condition. Then the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.
Other non-zero	An error occurred. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "mmsultra.h"

void GetSupplyType(unsigned short *);
void GetStockLength(unsigned short *);
void GetStockWidth(unsigned short *);

unsigned short main(void)
{
  unsigned short usStatus;        // Printer completion status
  unsigned short usStockWidth;    // Stk. width in 1/100th inches
  unsigned short usStockLength;   // Stk. length in 1/100th inches
  unsigned short usSupplyType;    // Supply type, 0 - 2
  unsigned short status = 0;      // Print subsystem init. status
  PRINTINIT rConfig;              // Print subsystem data struct.

  rConfig.lpuchFntAddr = 0;
  rConfig.ulFntSize = 0;
  status = pclInit(&rConfig);
  if (status != 0)
  {
    printf("Init Failed\nError: %d", status);
    exit(1);
  }
  pclPaperInfo(&usStockLength, &usStockWidth,
&usSupplyType);
```

```
  /* GetSupplyType is a programmer-written function */
  GetSupplyType(&usSupplyType);
  /* If fax, save setting, do not calibrate */
  if (usSupplyType == MMS_MEDIUM_ENERGY)
  {
    GetStockWidth(&usStockWidth);
    usStatus = pclPaperSetup(usStockLength, usStockWidth,
usSupplyType);
    return(usStatus);
  }
  usStatus = pclCalibratePaper(&usStockLength,
&usStockWidth);
  if (usStatus != 0)
  {
    pclClearError();
    return(usStatus);
  }
  /* GetStockLength is a programmer-written function */
  GetStockLength(&usStockLength);

  /* GetStockWidth is a programmer-written function */
  GetStockWidth(&usStockWidth);
  usStatus = pclPaperSetup(usStockLength, usStockWidth,
usSupplyType);
  pclClose();
}
```

pclStatus

Description

Retrieves the Print subsystem's status.

After submitting a print job, the application should call pclStatus in a loop, waiting until the printer becomes free. See "Pausing While Printing" in Chapter 6 for more information.

Syntax

```
short far pclStatus(void);
```

Parameters

None

Return Values

0	The Print subsystem is ready.
1	The Print subsystem is busy.
703-793	A motion control error occurred. After the operator corrects the printer condition, the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

See "pclGetOnDemandSensor" for an example.

pclWrite

Description

Writes MPCLII packets to the Print subsystem. The printer stores the packet in memory.

You can send no more than one packet at a time.

A batch packet starts a print job, which makes an asynchronous call to the Print subsystem. After submitting a print job, the application should call pclStatus in a loop, waiting until the printer becomes free. See “Pausing While Printing” in Chapter 6 for more information.

Syntax

```
short far pclWrite(char far* lpchBuffer,
                   unsigned short usCount);
```

Parameters

lpchBuffer A pointer to the data to write.

usCount The number of bytes to write. The maximum size is 64K.

Return Values

0 Successful.

Non-zero An error occurred. For errors between 703-793, The operator corrects the printer condition. Then the application must call pclClearError to reset the Motion Control subsystem. Refer to the MPCLII *Packet Reference Manual* for more information.

Example

See “pclGetOnDemandSensor” for an example.

scnCloseScanner

Description

Disables either scanner. Be sure to disable the scanner only when it is already enabled.

If the application enabled the scanner with scnOpenScanner,

- the application should disable the scanner at the end of processing or immediately before any serial communications.
- this function sets the serial communications port back to external RS-232 connection mode. It also restores the serial port configuration settings (baud rate, parity, data bits, and stop bits) saved when the application enabled the scanner.

Syntax

```
short far scnCloseScanner(void);
```

Parameters

None

Return Values

0 Successful.

-1 The scanner was already disabled.

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {
```

```
        /* Scan Bar Codes */
        sStatus = scnCloseScanner();        // Disable scanner
    }
}
```

scnGetBarCodeType

Description

Retrieves the last scanned bar code's type. Call this function only after receiving successful return codes from a scan that does not also retrieve the results. The bar code scanned stays in the scanner buffer until the application reads it.

Syntax

```
short far scnGetBarCodeType(void);
```

Parameters

None

Return Values

The following values can be returned when you use either library:

DCDE_C39	Code 39 (3 of 9 code)
DCDE_CBAR	Codabar (2 of 7 code)
DCDE_C128	Code 128
DCDE_I2of5	I 2 of 5 (USD-1)
DCDE_UPCA	UPCA
DCDE_UPCE	UPCE
DCDE_EAN8	EAN8
DCDE_EAN13	EAN13
DCDE_UPCA_2	UPCA + 2
DCDE_UPCE_2	UPCE + 2
DCDE_EAN8_2	EAN8 + 2
DCDE_EAN13_2	EAN13 + 2
DCDE_UPCA_5	UPCA + 5
DCDE_UPCE_5	UPCE + 5
DCDE_EAN8_5	EAN8 + 5

DCDE_EAN13_5	EAN13 + 5
DCDE_MSI	MSI

The following values can be returned only when you use SCAN1223.LIB:

DCDE_DISC2OF5	D 2 of 5
DCDE_IATA2OF5	IATA 2 of 5
DCDE_C93	Code 93
DCDE_EAN128	EAN 128
DCDE_UPCE1	UPCE1
DCDE_UPCE1_2	UPCE1 + 2
DCDE_UPCE1_5	UPCE1 + 5
DCDE_TRIOPTC39	Trioptic Code 39
DCDE_BOOKLANDEAN	Bookland EAN
DCDE_COUPONCODE	Coupon Code

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  char cBuffer[50];                     // Application buffer
  int iInput = 0;                       // Trigger input

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner open error-- %d", sStatus);
  else
  {
    printf("Press trigger to scan...\n"); // Perform scan
```

```
    iInput = _getch();
    if (iInput == 0)
    {
      iInput = _getch();
      if (iInput == 0x85)
      {
        sStatus = scnTrigger(1);
        if (sStatus != 0)
          printf("Scanner trigger error-- %d", sStatus);
        else
        {
          spkBeep(1, 1000);              // Beep for success
                                         // Retrieve data
          sStatus = scnGetScannedData(cBuffer);
          if (sStatus <= 0)
            printf("Error retrieving scanner data-- %d", sStatus);
          else
          {                              // Get bar code type
            sStatus = scnGetBarCodeType();
            printf("Bar code is a %d", sStatus);
          }
        }
      }
    }
  }
  sStatus = scnCloseScanner();           // Disable scanner
}
```

scnGetch

Description

Retrieves a character from the scanner buffer without echoing it to the display. If the scanner buffer is empty, this function activates the scanner to let the operator scan another bar code. This function works with both scanners.

This function tracks the characters in the bar code internally. For example, after a scan, a call to scnGetch reads the first character. Subsequent calls to scnGetch read the subsequent characters in the bar code. For example, the second call reads the second character, the third call reads the third character, etc.

Use scnGetche to retrieve characters and echo them to the display.

Syntax

```
short far scnGetch(void);
```

Parameters

None

Return Values

0-255	The retrieved character.
-2	Decoding fault. The scanner scanned the bar code, but the printer could not decode it.
-4	Checksum error.
-9	Scanner is disabled.
-10	Time-out error.

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Status of comm. calls
  char cBuffer[100];                    // Application buffer
  char *pBufptr = NULL;                 // Pointer to cBuffer
  int iInput = 0;                       // Trigger input

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  printf("Press trigger to\nscan...\n"); // Perform scan
  iInput = _getch();
  if (iInput == 0)
  {
    iInput = _getch();
    if (iInput == 0x85)
    {
      pBufptr = scnGets(cBuffer);
      if (pBufptr == NULL)
        printf("\nScanner error");
      else
      {
        spkBeep(1, 1000);               // Beep for success
        sStatus = scnGetch();           // Get first char
        if (sStatus < 0)                // Display result
          printf("Error getting char-- %d", sStatus);
        else
          printf("\nFirst Char is %c", sStatus);
      }
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnGetche

Description

Retrieves a character from the scanner buffer and echoes it to the display. If the scanner buffer is empty, this function activates the scanner to let the operator scan another bar code. This function works with both scanners.

This function tracks the characters in the bar code internally. For example, after a scan, a call to scnGetche reads the first character. Subsequent calls to scnGetche read the subsequent characters in the bar code. For example, the second call reads the second character, the third call reads the third character, etc.

Use scnGetch to retrieve characters without echoing them to the display.

Syntax

```
short far scnGetche(void);
```

Parameters

None

Return Values

0-255	The retrieved character.
-2	Decoding fault. The scanner scanned the bar code, but the printer could not decode it.
-4	Checksum error.
-9	Scanner is disabled.
-10	Time-out error.

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Status of comm. calls
  char cBuffer[100];                    // Application buffer
  char *pBufptr = NULL;                 // Pointer to cBuffer
  int iInput = 0;                       // Trigger input

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  printf("Press trigger to\nscan...\n"); // Perform scan
  iInput = _getch();
  if (iInput == 0)
  {
    iInput = _getch();
    if (iInput == 0x85)
    {
      pBufptr = scnGets(cBuffer);
      if (pBufptr == NULL)
        printf("\nScanner error");
      else
      {
        spkBeep(1, 1000);               // Beep for success
        sStatus = scnGetche();          // Get first char
        if (sStatus < 0)                // Display result
          printf("Error getting char-- %d", sStatus);
        else
          printf(" is the first char", sStatus);
      }
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnGetCodabarInfo

Description

SCAN1223.LIB only. Retrieves a pointer to a CODABARINFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetCodabarInfo, subsequent calls to scnGetCodabarInfo retrieve a pointer to a data structure containing the current values.

See “CODABARINFO” in Chapter 5 to learn more about the CODABARINFO data structure.

Syntax

```
short far scnGetCodabarInfo(LPCODABARINFO lprCodabarInfo);
```

Parameters

lprCodabarInfo	A pointer to a CODABARINFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetCodabarInfo” for an example.

scnGetCode128Info

Description

SCAN1223.LIB only. Retrieves a pointer to a CODE128INFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetCode128Info, subsequent calls to scnGetCode128Info retrieve a pointer to a data structure containing the current values.

See “CODE128INFO” in Chapter 5 to learn more about the CODE128INFO data structure.

Syntax

```
short far scnGetCode128Info(LPCODE128INFO lprCode128Info);
```

Parameters

lprCode128Info A pointer to a CODE129INFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetCode128Info” for an example.

scnGetCode39Info

Description

SCAN1223.LIB only. Retrieves a pointer to a CODE39INFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetCode39Info, subsequent calls to scnGetCode39Info retrieve a pointer to a data structure containing the current values.

See “CODE39INFO” in Chapter 5 to learn more about the CODE39INFO data structure.

Syntax

```
short far scnGetCode39Info(LPCODE39INFO lprCode39Info);
```

Parameters

lprCode39Info A pointer to a CODE39INFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetCode39Info” for an example.

scnGetCode93Info

Description

SCAN1223.LIB only. Retrieves a pointer to a CODE93INFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetCode93Info, subsequent calls to scnGetCode93Info retrieve a pointer to a data structure containing the current values.

See “CODE93INFO” in Chapter 5 to learn more about the CODE93INFO data structure.

Syntax

```
short far scnGetCode93Info(LPCODE93INFO lprCode93Info);
```

Parameters

lprCode93Info A pointer to a CODE93INFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetCode93Info” for an example.

scnGetD2of5Info

Description

SCAN1223.LIB only. Retrieves a pointer to a D2OF5INFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetD2of5Info, subsequent calls to scnGetD2of5Info retrieve a pointer to a data structure containing the current values.

See “D2OF5INFO” in Chapter 5 to learn more about the D2OF5INFO data structure.

Syntax

```
short far scnGetD2of5Info(LPD2OF5INFO lprD2of5Info);
```

Parameters

lprD2of5Info A pointer to a D2OF5INFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetD2of5Info” for an example.

scnGetGeneralInfo

Description

SCAN1223.LIB only. Retrieves a pointer to a GENERALINFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetGeneralInfo, subsequent calls to scnGetGeneralInfo retrieve a pointer to a data structure containing the current values.

See “GENERALINFO” in Chapter 5 to learn more about the GENERALINFO data structure.

Syntax

```
short far scnGetGeneralInfo(LPGENERALINFO lprGeneralInfo);
```

Parameters

lprGeneralInfo	A pointer to a GENERALINFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetGeneralInfo” for an example.

scnGetI2of5Info

Description

SCAN1223.LIB only. Retrieves a pointer to an I2OF5INFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetI2of5Info, subsequent calls to scnGetI2of5Info retrieve a pointer to a data structure containing the current values.

See “I2OF5INFO” in Chapter 5 to learn more about the I2OF5INFO data structure.

Syntax

```
short far scnGetI2of5Info(LPI2OF5INFO lprI2of5Info);
```

Parameters

lprI2of5Info A pointer to an I2OF5INFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetI2of5Info” for an example.

scnGetMSIInfo

Description

SCAN1223.LIB only. Retrieves a pointer to a MSIINFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetMSIInfo, subsequent calls to scnGetMSIInfo retrieve a pointer to a data structure containing the current values.

See “MSIINFO” in Chapter 5 to learn more about the MSIINFO data structure.

Syntax

```
short far scnGetMSIInfo(LPMSIINFO lprMSIInfo);
```

Parameters

lprMSIInfo A pointer to a MSIINFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See “scnSetMSIInfo” for an example.

scnGets

Description

Initiates a scan and moves the scanner buffer's contents to a programmer-defined application buffer as a null-terminated string. If the scanner buffer is empty, this function activates the scanner to let the operator scan a bar code. This function works with both scanners.

If there is any chance the scanned bar code contains a binary zero, do not use this function. Instead, use scnGetch and scnScannerHit in a loop until the scan is complete.

Use scnGetScannedData to retrieve the scanner buffer's contents without activating the scanner when the buffer is empty.

Syntax

```
char far * far scnGets(char far *lpszData);
```

Parameters

lpszData	A pointer to a programmer-defined application buffer where the function copies the scanner buffer's contents. This buffer must be one byte longer than the largest string that you may scan.

Return Values

A pointer to the buffer parameter	Successful.
NULL pointer	An error occurred.

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  char *pStatus2 = NULL;                  // Pointer to cBuffer
  char cBuffer[100];                      // Application buffer
  short sStatus1 = 0;                     // Command calls status
  int iInput = 0;                         // Trigger input

  sStatus1 = scnOpenScanner();            // Enable scanner
  if (sStatus1 == -2)
    printf("Scanner open error-- %d", sStatus1);
  else
  {                                       // Perform scan
    printf("Press trigger to\nscan...\n");
    iInput = _getch();
    if (iInput == 0)
    {
      iInput = _getch();
      if (iInput == 0x85)
      {
        pStatus2 = scnGets(cBuffer);
        if (pStatus2 == NULL)
          printf("Scanner buffer read error\n");
        else
        {
          spkBeep(1, 1000);               // Beep for success
                                          // Display result
          printf("Scan data:\n  %s\n", cBuffer);
        }
      }
    }
  }
  sStatus1 = scnCloseScanner();           // Disable scanner
}
```

scnGetScanInfo

Description

Retrieves a pointer to a SCANINFO data structure containing the default values for each parameter. If you change and set the parameters with scnSetScanInfo, subsequent calls to scnGetScanInfo retrieve a pointer to a data structure containing the current values.

This function is not valid with SCAN1223.LIB.

See “SCANINFO” in Chapter 5 to learn more about the SCANINFO data structure.

Syntax

```
short far scnGetScanInfo(LPSCANINFO lprScanInfo);
```

Parameters

lprScanInfo A pointer to a SCANINFO data structure.

Return Values

0	Successful.
-9	The scanner is disabled.

Example

See “scnSetScanInfo” for an example.

scnGetScannedData

Description

Retrieves the scanner buffer's contents as a null-terminated string, placing them in a programmer-defined application buffer. If the scanner buffer is empty, this function does not activate the trigger to start a scan. This function works with both scanners.

If there is any chance the scanned bar code contains a binary zero, do not use this function. Instead, use scnGetch and scnScannerHit in a loop until the scan is complete.

Use scnGets to retrieve the scanner buffer's contents and activate the scanner when the buffer is empty.

Syntax

```
short far scnGetScannedData(char far *lpszData);
```

Parameters

lpszData A pointer to a programmer-defined application buffer where the function places the scanner buffer's contents. This buffer must be one byte longer than the largest string that you may scan.

Return Values

0	No data available.
-1	Checksum, time-out, or communications error.
-2	Decoding fault. The scanner scanned the bar code, but the printer could not decode it.
-9	Scanner is disabled.

> 0 The type of bar code retrieved. The following values can be returned when you use either library:

DCDE_C39	Code 39 (3 of 9 code)
DCDE_CBAR	Codabar (2 of 7 code)
DCDE_C128	Code 128
DCDE_I2of5	I 2 of 5 (USD-1)
DCDE_UPCA	UPCA
DCDE_UPCE	UPCE
DCDE_EAN8	EAN8
DCDE_EAN13	EAN13
DCDE_UPCA_2	UPCA + 2
DCDE_UPCE_2	UPCE + 2
DCDE_EAN8_2	EAN8 + 2
DCDE_EAN13_2	EAN13 + 2
DCDE_UPCA_5	UPCA + 5
DCDE_UPCE_5	UPCE + 5
DCDE_EAN8_5	EAN8 + 5
DCDE_EAN13_5	EAN13 + 5
DCDE_MSI	MSI

The following values can be returned only when you use SCAN1223.LIB:

DCDE_DISC2OF5	D 2 of 5
DCDE_IATA2OF5	IATA 2 of 5
DCDE_C93	Code 93
DCDE_EAN128	EAN 128
DCDE_UPCE1	UPCE1
DCDE_UPCE1_2	UPCE1 + 2
DCDE_UPCE1_5	UPCE1 + 5
DCDE_TRIOPTC39	Trioptic Code 39
DCDE_BOOKLANDEAN	Bookland EAN
DCDE_COUPONCODE	Coupon Code

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"
```

```
void main (void)
{
  short sStatus = 0;                    // Command calls status
  char cBuffer[100];                    // Internal scanner buffer
  int iInput = 0;                       // Trigger input

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {
                                        // Perform scan
    printf("Press trigger to\nscan...\n");
    iInput = _getch();
    if (iInput == 0)
    {
      iInput = _getch();
      if (iInput == 0x85)
      {
        sStatus = scnTrigger(1);
        if (sStatus != 0)
          printf("\nScanner error-- %d", sStatus);
        else
        {                               // Get data from scan
          sStatus = scnGetScannedData(cBuffer);
          if (sStatus <= 0)
            printf("Data retrieval error-- %d", sStatus);
          else
          {
            spkBeep(1, 1000);           // Beep for success
                                        // Display result
            printf("Scan data:\n  %s\n", cBuffer);
            printf("Bar code type: %d", sStatus);
          }
        }
      }
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnGetUPCEANInfo

Description

SCAN1223.LIB only. Retrieves a pointer to the 1223 scanner's default configuration for UPC and EAN bar codes. See "UPCEANINFO" in Chapter 5 to learn more about the UPCEANINFO data structure.

Syntax

```
short far scnGetUPCEANInfo(LPUPCEANINFO lprUPCEANInfo);
```

Parameters

lprUPCEANInfo A pointer to a UPCEANINFO data structure.

Return Values

0	Successful.
-9	Scanner is disabled.

Example

See "scnSetUPCEANInfo" for an example.

scnOpenScanner

Description

Enables either scanner and sets the configuration values to the defaults. Be sure to enable the scanner only when it is already disabled.

Because this function resets the configuration values to the defaults, you must configure the scanner every time you enable it.

This function directs the serial port to the scanner, making external RS-232 communications unavailable (unlike scnOpenScannerShared) until the application disables the scanner with scnCloseScanner. There may be a slight delay when enabling the scanner for the software to determine the scanner type being used.

Because the scanner takes over the serial communications port, the application should disable the scanner immediately before using the port for serial communications and enable it immediately afterwards. When possible, the application should use scnOpenScannerShared to avoid processing delays (enabling and disabling the scanner repeatedly).

scnOpenScanner also saves the serial port's configuration (baud, parity, data bits, stop bits) to restore when the application disables the scanner. The application must enable the scanner before configuring it.

The application must call this function before it calls any other scanner function (unless it uses scnOpenScannerShared).

Syntax

```
short far scnOpenScanner(void);
```

Parameters

None

Return Values

0 Successful.

-1 The scanner is already enabled.

-2 No scanner is installed or the application cannot communicate with the scanner.

Example

See “scnCloseScanner” for an example.

scnOpenScannerShared

Description

Enables either scanner and sets the configuration values to the defaults. Be sure to enable the scanner only when it is already disabled.

Because this function resets the configuration values to the defaults, you must configure the scanner every time you enable it.

This function allows the serial port to be shared for scanning and serial communications (unlike scnOpenScanner). There may be a slight delay when enabling the scanner for the software to determine the scanner type being used.

Because this function allows the serial port to be shared, application does not need to enable and disable the scanner repeatedly to free the serial port for RS-232 communications. It can enable the scanner at the beginning of processing and disable the scanner at the end of processing.

Using this function has the benefit of eliminating processing delays (enabling and disabling the scanner repeatedly), making the application run faster.

The application must call this function before it calls any other scanner function (unless it uses scnOpenScanner).

Syntax

```
short far scnOpenScannerShared(void);
```

Parameters

None

Return Values

0 Successful.

-1 The scanner is already enabled.

-2 No scanner is installed or the application cannot communicate with the scanner.

Example

```
#include <bios.h>
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  unsigned uStatus = 0;                 // bios_serialcom status
  int iInput = 0;                       // Trigger input
  static char cBuffer[200];             // Application buffer
  char far *pScan = NULL;               // Pointer from scan

sStatus = scnOpenScannerShared();       // Enable shared scanner
  if (sStatus != 0)
    printf("Error opening scanner:\n%d\n", sStatus);
  else
  {                                     // Open port
    uStatus = _bios_serialcom(_COM_INIT, 0,
                              _COM_9600 | _COM_CHR8 |
                              _COM_EVENPARITY |
_COM_STOP1);
    printf("Comm port open\n");
    printf("return status: %d\nPress Enter...", uStatus);
    _getch();
    printf("\nScanning...\n");          // Perform scan
    iInput = getch();
    if (iInput == 0x00)
    {
      iInput = getch();
      if (iInput == 0x85)
      {
```

```
        pScan = scnGets(cBuffer);
        if (pScan == NULL)
          printf("Scanning error\n");
        else
        {
          spkBeep(1, 1000);
          printf("Scanned data:\n%s\nPress Enter...\n",
             &cBuffer);
          _getch();
                                        // Send data out port
          uStatus = _bios_serialcom(_COM_SEND, 0, 0);
          printf("Comm port write\n");
          printf("return status: %d\nPress Enter...", uStatus);
          _getch();
        }
      }
    }
  }
  scnCloseScanner();                    // Disable scanner
}
```

scnScannerHit

Description

Checks for data in the scanner buffer. If there is data, it returns the type of bar code the data is from. This function works with both scanners.

Syntax

```
short far scnScannerHit(void);
```

Parameters

None

Return Values

0 The scanner buffer is empty.

Non-zero The type of bar code in the scanner buffer. When you use either library, the following values can be returned.

DCDE_C39	Code 39 (3 of 9 code)
DCDE_CBAR	Codabar (2 of 7 code)
DCDE_C128	Code 128
DCDE_I2of5	I 2 of 5 (USD-1)
DCDE_UPCA	UPCA
DCDE_UPCE	UPCE
DCDE_EAN8	EAN8
DCDE_EAN13	EAN13
DCDE_UPCA_2	UPCA + 2
DCDE_UPCE_2	UPCE + 2
DCDE_EAN8_2	EAN8 + 2
DCDE_EAN13_2	EAN13 + 2
DCDE_UPCA_5	UPCA + 5
DCDE_UPCE_5	UPCE + 5
DCDE_EAN8_5	EAN8 + 5
DCDE_EAN13_5	EAN13 + 5
DCDE_MSI	MSI

The following values can be returned only when you use SCAN1223.LIB.

DCDE_DISC2OF5	D 2 of 5
DCDE_IATA2OF5	IATA 2 of 5
DCDE_C93	Code 93
DCDE_EAN128	EAN 128
DCDE_UPCE1	UPCE1
DCDE_UPCE1_2	UPCE1 + 2
DCDE_UPCE1_5	UPCE1 + 5
DCDE_TRIOPTC39	Trioptic Code 39
DCDE_BOOKLANDEAN	Bookland EAN
DCDE_COUPONCODE	Coupon Code

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main (void)
{
  short sStatus = 0;                    // Command calls status
  LPSCANINFO SScnconfig;                // Scanner data structure
  sStatus = scnOpenScanner();           // Enable scanner

  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Configure scanner
    sStatus = scnGetScanInfo(SScnconfig);
    if (sStatus != 0)
      printf("Scanner is disabled\n");
    else
    {
      SScnconfig->uchCode39 = SCN_ENABLE;
      sStatus = scnSetScanInfo(SScnconfig);
      if (sStatus != 0)
        printf("Scanner is disabled\n");
      else
      {
        printf("Scanning...\n");        // Perform scan
        sStatus = scnTrigger(1);
```

```
            if (sStatus != 0)
               printf("Scan error-- %d\n", sStatus);
            else
            {
               spkBeep(1, 1000);
               sStatus = scnScannerHit();      // Check scanner buffer
               if (sStatus == 0)
                  printf("Scanner buffer is\nempty\n");
               else                            // Display result
                  printf("Bar code scanned was a %d", sStatus);
            }
         }
      }
   }
   sStatus = scnCloseScanner();               // Disable scanner
}
```

scnSetCodabarInfo

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the CODABARINFO data structure. See “CODABARINFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetCodabarInfo.
3. Sets the values in the CODABARINFO data structure.
4. Calls scnSetCodabarInfo.

When the application disables the scanner, the configuration values return to the defaults.

Syntax

```
short far scnSetCodabarInfo(LPCODABARINFO lprCodabarInfo);
```

Parameters

lprCodabarInfo A pointer to a CODABARINFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPCODABARINFO CDBconfig;              // Codabar data struct.

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get configuration
    sStatus = scnGetCodabarInfo(CDBconfig);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
      CDBconfig->uchEnable = SCN_ENABLE;
      CDBconfig->uchLength1 = 0;
      CDBconfig->uchLength2 = 0;
      CDBconfig->uchEnableCLSIEdit = SCN_DISABLE;
      CDBconfig->uchEnableNOTISEdit = SCN_DISABLE;
                                        // Save values
      sStatus = scnSetCodabarInfo(CDBconfig);
      if (sStatus != 0)
        printf("Error setting values-- ", sStatus);
      else
      {
        ; /* Scan Bar Codes */
      }
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetCode128Info

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the CODE128INFO data structure. See “CODE128INFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetCode128Info.
3. Sets the values in the CODE128INFO data structure.
4. Calls scnSetCode128Info.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetCode128Info(LPCODE128INFO lprCode128Info);
```

Parameters

lprCode128Info A pointer to a CODE128INFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPCODE128INFO config128;              // Code 128 data struct.

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get config.
    sStatus = scnGetCode128Info(config128);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
      config128->uchEnableUSS128 = SCN_ENABLE;
      config128->uchEnableUCCEAN128 = SCN_DISABLE;
      config128->uchEnableISBT128 = SCN_DISABLE;
                                        // Save values
      sStatus = scnSetCode128Info(config128);
      if (sStatus != 0)
        printf("Error setting values- ", sStatus);
      else
        ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetCode39Info

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the CODE39INFO data structure. See “CODE39INFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetCode39Info.
3. Sets the values in the CODE39INFO data structure.
4. Calls scnSetCode39Info.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetCode39Info(LPCODE39INFO lprCode39Info);
```

Parameters

lprCode39Info	A pointer to a CODE39INFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPCODE39INFO config39;                // Code 39 data struct.

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else                                  //Get config.
  {     sStatus = scnGetCode39Info(config39);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
        config39->uchEnable = SCN_ENABLE;
        config39->uchEnableTrioptic = SCN_DISABLE;
        config39->uchCvtC39toC32 = SCN_DISABLE;
        config39->uchEnableC32Prefix = SCN_DISABLE;
        config39->uchLength1 = 0;
        config39->uchLength2 = 0;
        config39->uchVerifyCheckDigit = SCN_DISABLE;
        config39->uchXmitCheckDigit = SCN_DISABLE;
        config39->uchEnableFullASCII = SCN_DISABLE;
                                        // Save values
        sStatus = scnSetCode39Info(config39);
        if (sStatus != 0)
          printf("Error setting values-- ", sStatus);
        else
          ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetCode93Info

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the CODE93INFO data structure. See “CODE93INFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetCode93Info.
3. Sets the values in the CODE93INFO data structure.
4. Calls scnSetCode93Info.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetCode93Info(LPCODE93INFO lprCode39Info);
```

Parameters

lprCode93Info A pointer to a CODE93INFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls config.
  LPCODE93INFO config93;                // Code 93 data structure
  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get config.
    sStatus = scnGetCode93Info(config93);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
        config93->uchEnable = SCN_ENABLE;
        config93->uchLength1 = 0;
        config93->uchLength2 = 0;
                                        // Save values
        status = scnSetCode93Info(config93);
        if (sStatus != 0)
          printf("Error setting values-- ", sStatus);
        else
          ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetD2of5Info

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the D2OF5INFO data structure. See “D2OF5INFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetD2of5Info.
3. Sets the values in the D2OF5INFO data structure.
4. Calls scnSetD2of5Info.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetD2of5Info(LPD2OF5INFO lprD2of5Info);
```

Parameters

lprD2of5Info A pointer to a D2OF5INFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPD2OF5INFO D25config;                // D 2 of 5 data structure

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {
    sStatus = scnGetD2of5Info(D25config);     // Get config.
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                         // Set values
        D25config->uchEnable = SCN_ENABLE;
        D25config->uchLength1 = 0;
        D25config->uchLength2 = 0;
                                              // Save values
        sStatus = scnSetD2of5Info(D25config);
        if (sStatus != 0)
          printf("Error setting values-- ", sStatus);
        else
          ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();              // Disable scanner
}
```

scnSetGeneralInfo

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the GENERALINFO data structure. See “GENERALINFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetGeneralInfo.
3. Sets the values in the GENERALINFO data structure.
4. Calls scnSetGeneralInfo.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetGeneralInfo(LPGENERALINFO lprGeneralInfo);
```

Parameters

lprGeneralInfo A pointer to a GENERALINFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPGENERALINFO GENconfig;              // General data structure

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get config.
    sStatus = scnGetGeneralInfo(GENconfig);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
      GENconfig->uchLaserOnTime = 40;
      GENconfig->uchPowerMode = 1;
      GENconfig->uchTriggerMode = 1;
      GENconfig->uchSameSymbolTMO = 10;
      GENconfig->uchLinearCodeSecur = 1;
      GENconfig->uchBiDirRedun = SCN_DISABLE;
                                        // Save values
      sStatus = scnSetGeneralInfo(GENconfig);
      if (sStatus != 0)
        printf("Error setting values-- ", sStatus);
      else
        ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetI2of5Info

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the I2OF5INFO data structure. See “I2OF5INFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetI2of5Info.
3. Sets the values in the I2OF5INFO data structure.
4. Calls scnSetI2OF5Info.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetI2of5Info(LPI2OF5INFO lprI2of5Info);
```

Parameters

lprI2of5Info A pointer to an I2OF5INFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPI2OF5INFO I25config;                // I 2 of 5 data structure

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get config.
    sStatus = scnGetI2of5Info(I25config);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
        I25config->uchEnable = SCN_ENABLE;
        I25config->uchLength1 = 0;
        I25config->uchLength2 = 0;
        I25config->uchChkDgtAlgorithm = 1;
        I25config->uchXmitCheckDigit = SCN_ENABLE;
        I25config->uchCvtI2of5toEAN13 = SCN_DISABLE;
                                        // Save values
        sStatus = scnSetI2of5Info(I25config);
        if (sStatus != 0)
          printf("Error setting values-- ", sStatus);
        else
          ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetMSIInfo

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the MSIINFO data structure. See "MSIINFO" in Chapter 5 for a description of this data structure.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetMSIInfo.
3. Sets the values in the MSIINFO data structure.
4. Calls scnSetMSIInfo.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetMSIInfo(LPMSIINFO lprMSIInfo);
```

Parameters

lprMSIInfo A pointer to an MSIINFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPMSIINFO MSIconfig;                  // MSI data struct.

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {
    sStatus = scnGetMSIInfo(MSIconfig);       // Get config.
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                         // Set values
        MSIconfig->uchEnable = SCN_ENABLE;
        MSIconfig->uchLength1 = 0;
        MSIconfig->uchLength2 = 0;
        MSIconfig->uchCheckDigits = 0;
        MSIconfig->uchXmitCheckDigit = SCN_ENABLE;
        MSIconfig->uchChkDgtAlgorithm = 0;
                                          // Save values
        sStatus = scnSetMSIInfo(MSIconfig);
        if (sStatus != 0)
          printf("Error setting values-- ", sStatus);
        else
          ; /* Scan Bar Codes */
    }
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetScanInfo

Description

Saves either scanner's configuration values the application set in the SCANINFO data structure. See "SCANINFO" in Chapter 5 for a description of this data structure.

This function is not valid with SCAN1223.LIB.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetScanInfo.
3. Sets the values in the SCANINFO data structure.
4. Calls scnSetScanInfo.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetScanInfo(LPSCANINFO lprScanInfo);
```

Parameters

lprScanInfo A pointer to a SCANINFO data structure.

Return Values

0	Successful.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPSCANINFO SCNconfig;                 // Scanner data structure

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  sStatus = scnGetScanInfo(SCNconfig);      // Get config.
  if (sStatus != 0)
    printf("Scanner is disabled");
  else
  {
    SCNconfig->uchCode39 = SCN_ENABLE;  // Set value
    sStatus = scnSetScanInfo(SCNconfig);    // Save value
    if (sStatus != 0)
      printf("Scanner is disabled");
    else
      ; /* Scan Bar Codes */
  }
  sStatus = scnCloseScanner();          // Disable scanner
}
```

scnSetUPCEANInfo

Description

SCAN1223.LIB only. Saves the 1223 scanner configuration values the application set in the UPCEANINFO data structure. See “UPCEANINFO” in Chapter 5 for a description of this data structure.

To configure the scanner, the application:

1. Enables the scanner with scnOpenScanner or scnOpenScannerShared.
2. Calls scnGetUPCEANInfo.
3. Sets the values in the UPCEANINFO data structure.
4. Calls scnSetUPCEANInfo.

When the application disables the scanner, the configuration values go back to the defaults.

Syntax

```
short far scnSetUPCEANInfo(LPUPCEANINFO lprUPCEANInfo);
```

Parameters

lprUPCEANInfo A pointer to a UPCEANINFO data structure.

Return Values

0	Successful.
-5	Invalid configuration value.
-9	The scanner is disabled.

Example

```
#include <stdio.h>
#include "scan1223.h"
#include "mmsultra.h"

void main(void)
{
  short sStatus = 0;                    // Command calls status
  LPUPCEANINFO UEconfig;                // UPCEAN data structure

  sStatus = scnOpenScanner();           // Enable scanner
  if (sStatus == -2)
    printf("Scanner error-- %d", sStatus);
  else
  {                                     // Get configuration
    sStatus = scnGetUPCEANInfo(UEconfig);
    if (sStatus != 0)
      printf("Scanner is disabled.");
    else
    {                                   // Set values
        UEconfig->uchEnableUPCA = SCN_ENABLE;
        UEconfig->uchEnableUPCE = SCN_DISABLE;
        UEconfig->uchEnableUPCE1 = SCN_DISABLE;
        UEconfig->uchEnableEAN8 = SCN_DISABLE;
        UEconfig->uchEnableEAN13 = SCN_DISABLE;
        UEconfig->uchEnableBookEAN = SCN_DISABLE;
        UEconfig->uchEnableSupps = 2;
        UEconfig->uchEnableSuppRedun = 10;
        UEconfig->uchXmitUPCAChkDgt = SCN_ENABLE;
        UEconfig->uchXmitUPCEChkDgt = SCN_DISABLE;
        UEconfig->uchXmitUPCE1ChkDgt = SCN_DISABLE;
        UEconfig->uchUPCAPreamble = 2;
        UEconfig->uchUPCEPreamble = 0;
        UEconfig->uchUPCE1Preamble = 0;
        UEconfig->uchCvtUPCEtoUPCA = SCN_DISABLE;
        UEconfig->uchCvtUPCE1toUPCA = SCN_DISABLE;
        UEconfig->uchEAN8ZeroExtend = SCN_DISABLE;
        UEconfig->uchCvtEAN8toEAN13 = SCN_DISABLE;
        UEconfig->uchSecurityLevel = 1;
        UEconfig->uchEnableCouponCode = SCN_ENABLE;
```

```
                                        // Save values
        sStatus = scnSetUPCEANInfo(UEconfig);
        if (sStatus != 0)
          printf("Scanner is disabled");
        else
          ; /* Scan Bar Codes */
      }
    }
    sStatus = scnCloseScanner();          // Disable scanner
}
```

scnTrigger

Description

Initiates a scan, placing the scanned data in the scanner buffer. If the LED at the keypad's bottom turns green, the scan was successful. This function works with either scanner.

Call scnGetScannedData immediately after calling scnTrigger. scnGetScannedData allows the application to read the scanned data.

Syntax

```
short far scnTrigger(short sWait);
```

Parameters

sWait Flag indicating whether to wait until the scan is complete. Values are:

- *0* Return immediately. If the application uses this option, the scan may still be in progress when the function returns to the application. If so, it must use scnScannerHit to determine when the scan ends.
- *1* Wait until the scan is complete, timed out, or un-decodable.

Return Values

0 Successful.

-9 The scanner is disabled.

-10 The scanner timed out.

Example

See "scnGetScannedData" for an example.

spkBeep

Description

Sounds the printer's beeper for the specified duration and frequency. If you pass invalid values in either parameter, the beeper does not sound.

Syntax

```
void far spkBeep(unsigned char uchDuration,
                 unsigned short usFrequency);
```

Parameters

uchDuration	The duration in tenths of a second. Values are *1-10*.
usFrequency	The frequency in hertz. Values are *110-10,000*.

Return Values

None

Example

See "scnGets" for an example.

sysGetBIOSVersion

Description

Retrieves the BIOS version and build date.

The date string's format is mm/dd/yy. The version string's format is M.mm ss, where M is the major version, mm is the minor version, and ss is the sub-version. For example, 1.01 S.

Syntax

```
int far sysGetBIOSVersion(unsigned char far *lpszVersion,
                          unsigned char far *lpszDate);
```

Parameters

lpszVersion	Pointer to the BIOS version.
lpszDate	Pointer to the BIOS build date. If you do not need the date, set this parameter to NULL.

Return Values

0	Successful
-1	Unsuccessful.

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  int iStatus = 0;                          // Status of retrieval
  unsigned char ucVersion[9] = "       ";// BIOS version
  unsigned char ucDate[10] = "        "; // BIOS date

  iStatus = sysGetBIOSVersion(ucVersion, ucDate);    // Get info
  if (iStatus != 0)
    printf("BIOS Version retrieval failed.");
  else
  {
```

```
    vidScroll(0, 0, 3, 11, 0, 0x07);          // Display results
    vidPutCursor(0, 0, 0);
    printf("       BIOS       ");
    printf("Version is %s\n", ucVersion);
    printf("Date is %s", ucDate);
  }
}
```

vidBackLightOn

Description

Turns the LCD backlight on or off.

Syntax

```
void far vidBackLightOn(short sOn);
```

Parameters

sOn The state to change the backlight to. Values are *1* (for On) and *0* (for Off).

Return Values

None

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  int cInput = 32;                          // User input

  printf("Turn backlight on?\n(Y/N): ");    // Prompt user
  cInput = _getch();                        // Get input
  switch (cInput)                           // Take action
  {
    case 'N': vidBackLightOn(0);
              break;
    case 'Y': vidBackLightOn(1);
              break;
    default:  printf("Invalid input");
  }
}
```

vidGetState

Description

Retrieves the current video mode as defined in vidSetMode.

Syntax

```
unsigned short far vidGetState(short far* lpsColCnt,
                               short far* lpsPage);
```

Parameters

lpsColCnt	A variable pointer to the number of character columns.
lpsPage	A variable pointer to the current active display page.

Return Values

0	The current video mode (20 column display).

Example

See “pclCalibrate” for an example.

vidPutCursor

Description

Moves the cursor for the specified display page to the specified row and column.

Syntax

```
void far vidPutCursor(unsigned short usRow,
                      unsigned short usCol,
                      short sPage);
```

Parameters

usRow	Row. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-7*.
usCol	Column. Values are *0-19*.
usPage	Display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  vidSetPage(0);                              // Set the page
  vidScroll(0, 0, 3, 19, 0, 0x07);            // Clear the screen
  vidPutCursor(0, 0, 0);                      // Move the cursor
  printf("X\n");                              // Print an X
  printf("The 'X' is at\ncolumn 0, row 0");   // Display message
}
```

vidPutStr

Description

Writes a string of ASCII characters with an attribute at the specified display page's current cursor location. The string overwrites the characters in the affected positions. This function does not move the cursor. Use vidPutCursor to move it.

If the application writes to a display page other than the current one, the written string does not appear until the application sets that page as the current one with vidSetPage. Strings written to the current page appears immediately.

Syntax

```
void far vidPutStr(char far* lpchString,
                   unsigned char uchAttr,
                   short sPage);
```

Parameters

lpchString The string to write. Bell, backspace, carriage return, and line feed characters are invalid. The string's length must be less than or equal to the number of remaining columns in the current row.

uchAttr The string's attribute. Values are:

0x07	Normal video
0x70	Reverse video

sPage The display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include "mmsultra.h"

void main(void)
{
  vidSetPage(0);                    // Set the page
  vidScroll(0, 0, 7, 19, 0, 0x07);  // Clear the screen
  vidPutCursor(0, 0, 0);            // Move the cursor
  vidPutStr("REVERSE", 0x70, 0);    // Print in reverse video
  vidPutCursor(1, 0, 0);            // Move the cursor
  vidPutStr("NORMAL", 0x07, 0);     // Print in Normal video
}
```

vidReadCA

Description

Reads a character and attribute from the current cursor location for the specified display page.

Syntax

```
void far vidReadCA(unsigned char far* lpuchChr,
                   unsigned char far* lpuchAttr,
                   short sPage);
```

Parameters

lpuchChr A variable pointer to the character.

lpuchAttr A variable pointer to the character's attribute. Returned values are:

0x07 Normal video
0x70 Reverse video

sPage The display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  unsigned char ucCharacter = 32;      // Character at position
  unsigned char ucAttribute = 32;      // Attribute at position

  vidSetPage(0);                       // Set page
  vidPutCursor(0, 0, 0);               // Move cursor
                                       // Read char and attribute
  vidReadCA(&ucCharacter, &ucAttribute, 0);
```

```
    vidScroll(0, 0, 7, 19, 0, 0x07);    // Clear screen
                                        // Display results
    printf("Character read is %c\n", ucCharacter);
    if (ucAttribute == 0x07)
      printf("Attribute read is\nnormal");
    else
      printf("Attribute read is\nreverse");
}
```

vidReadCursor

Description

Retrieves the specified display page's current cursor location.

Syntax

```
void far vidReadCursor(unsigned short far* lpusRow,
                       unsigned short far* lpusCol,
                       short sPage);
```

Parameters

lpusRow	A variable pointer to the row. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-7*.
lpusCol	A variable pointer to the column. Values are *0-19*.
sPage	The display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  unsigned short usRow = 0;                    // Row position
  unsigned short usColumn = 0;                 // Column position

  vidScroll(0, 0, 3, 11, 0, 0x07);             // Clear screen
  vidPutCursor(0, 0, 0);                       // Move cursor
  vidReadCursor(&usRow, &usColumn, 0);         // Read position
  printf("Row position is %d", usRow);         // Display results
  printf("\nColumn position is %d", usColumn);
}
```

vidScroll

Description

Does either of the following to the current display page:

- Sets the display to ASCII space characters in either normal or reverse video.
- Scrolls the specified window up or down by a specified number of lines. The application loses any text that scrolls beyond the window's top or bottom. New lines contain ASCII blank characters with the specified attribute.

The specified window is the entire display or part of it.

Syntax

```
void far vidScroll(short sTop,
                   short sLeft,
                   short sBottom,
                   short sRight,
                   short sNumber,
                   unsigned char uchAttr);
```

Parameters

sTop	The window's top row. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-7*.
sLeft	The window's left-most column. Values are *0-19*.
sBottom	The window's bottom row. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-7*.
sRight	The window's right-most column. Values are *0-19*.
sNumber	The action. If you enter 0, the function clears the entire display. If you enter a positive number, the window scrolls up that many lines. If you enter a negative number, the window scrolls down that many lines.

uchAttr Attribute for the cleared area or new lines when scrolling. Values are:

0x07 Normal video
0x70 Reverse video

Return Values

None

Example

```
#include <stdio.h>
#include <conio.h>
#include "mmsultra.h"

void main(void)
{
                                        // Prompt user
  printf("Press Enter to\nclear the display\nin reverse video.");
  _getch();                             // Read Enter
  vidScroll(0, 0, 3, 11, 0, 0x70);      // Clear scr. in rev. video
}
```

vidSetCursorType

Description

Defines the cursor style to use. The display consists of 4 or 8 rows, depending on the printer you have. Each row consists of 8 horizontal lines, for a total of 32 or 64 lines. The cursor, at most, is as tall as a display row. So, it has up to 8 horizontal lines, numbered 0 to 7 (from top to bottom). To define a cursor style, specify a range of consecutive lines.

On printers with the 4-row/33-key printer, there is a gap between lines 31 and 32, where nothing displays. This gap causes a solid block cursor to look fine on the first three rows, but not on the fourth row. Due to this gap, the application should use cursor line 7 only as a single-line cursor.

Syntax

```
void far vidSetCursorType(short sStart,
                          short sEnd);
```

Parameters

sStart	The cursor's top line. Values are *0-7*. The default is 7.
sEnd	The cursor's bottom line. Values are *0-7*. The default is 7.

Return Values

None

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"

void main(void)
{
  int cStyle = 32;                                  // Cursor style

                                                    // Prompt user
  printf("Choose cursor type:\nA (block)\nB (underscore)");
  cStyle = _getch();                                // Get input
  switch(cStyle)                                    // Take action
  {
    case 'A': vidSetCursorType(0, 6);
              break;
    case 'B': vidSetCursorType(7, 7);
              break;
    default:  printf("\nInvalid input");
  }
}
```

vidSetMode

Description

Sets the video mode and clears the screen for the current display page.

Syntax

```
void far vidSetMode(unsigned short usMode);
```

Parameters

usMode The video mode. Enter *0*.

Return Values

None

Example

See “pclCalibrate” for an example.

vidSetPage

Description

Switches to the specified display page and displays it. Switching between pages does not affect their contents.

To ensure the

- application begins on the same page every time, use this function to set the display page at the application's beginning.
- display pages are clear, use vidScroll to clear the pages at the application's beginning.

If the application writes to a display page other than the current one, the written string does not appear until the application sets that page as the current one with vidSetPage. Strings written to the current page appears immediately.

Syntax

```
void far vidSetPage(short sPage);
```

Parameters

sPage	Display page. For 4-line/33-key printers, values are *0-3*. For 8-line/48-key printers, values are *0-1*. The default is *0*.

Return Values

None

Example

```
#include <conio.h>
#include <stdio.h>
#include "mmsultra.h"
```

```
void main(void)
{
  vidSetPage(0);                    // Clear page 0
  vidScroll(0, 0, 3, 11, 0, 0x07);
  vidSetPage(1);                    // Clear page 1
  vidScroll(0, 0, 3, 11, 0, 0x07);
  vidPutCursor(0, 0, 0);            // Write to page 0
  vidPutStr("This is page 0", 0x70, 0);
  vidPutCursor(0, 0, 1);            // Write to page 1
  vidPutStr("This is page 1", 0x70, 1);
                                    // Prompt user
  printf("\nPress any key to\nswitch to page 0");
  _getch();
  vidSetPage(0);                    // Switch pages
  printf("\nPress any key to end");  // Prompt user
  _getch();
  vidScroll(0, 0, 3, 11, 0, 0x07);  // Clear page 0
  vidSetPage(1);                    // Clear page 1
  vidScroll(0, 0, 3, 11, 0, 0x07);
  vidSetPage(0);                    // Set to page 0
}
```

vidWriteC

Description

Writes an ASCII character at the specified display page's current cursor location, overwriting any character that may already be there, but keeping the attribute. After calling this function, call vidPutCursor to move the cursor to a new position.

If the application writes to a display page other than the current one, the written string does not appear until the application sets that page as the current one with vidSetPage. Strings written to the current page appears immediately.

To write a character and attribute, use vidWriteCA.

Syntax

```
void far vidWriteC(unsigned char uchChr,
                   short sCount,
                   short sPage);
```

Parameters

uchChr	The ASCII character to write. Bell, backspace, carriage return, and line feed characters are invalid.
sCount	The number of times to write the character. This number must be less than or equal to the number of columns remaining in the current row.
sPage	The display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include "mmsultra.h"

void main(void)
{
  int iCharacter = 42;                  // An asterisk

  vidSetPage(0);                        // Set page
  vidScroll(0, 0, 7, 19, 0, 0x07);      // Clear screen
  vidPutCursor(0, 0, 0);                // Position cursor
  vidWriteC(iCharacter, 5, 0);          // Write character
}
```

vidWriteCA

Description

Writes a character with an attribute at the specified display page's current cursor location, overwriting any character (and attribute) that may already be there.

After calling this function, call vidPutCursor to move the cursor to a new position.

To write a character without an attribute, use vidWriteC.

If the application writes to a display page other than the current one, the written string does not appear until the application sets that page as the current one with vidSetPage. Strings written to the current page appears immediately.

Syntax

```
void far vidWriteCA(unsigned char uchChr,
                    unsigned char uchAttr,
                    short sCount,
                    short sPage);
```

Parameters

uchChr — The ASCII character to write. Bell, backspace, carriage return, and line feed characters are invalid.

uchAttr — The character's attribute. Values are:

0x07	Normal video
0x70	Reverse video

sCount — The number of times to write the character. This number must be less than or equal to the number of columns remaining in the current row.

sPage The display page. For 4-row/33-key printers, values are *0-3*. For 8-row/48-key printers, values are *0-1*.

Return Values

None

Example

```
#include "mmsultra.h"

void main(void)
{
  int iCharacter = 42;                    // An asterisk

  vidSetPage(0);                          // Set page
  vidScroll(0, 0, 7, 19, 0, 0x07);        // Clear screen
  vidPutCursor(0, 0, 0);                  // Position cursor
  vidWriteCA(iCharacter, 0x70, 5, 0);     // Write character
}
```

DATA STRUCTURE REFERENCE

5

Certain functions described in the last chapter require the application to use certain data structures. This chapter describes these structures. It lists them alphabetically.

Following is an overview:

Name	Description
CODABARINFO	1223 scanner configuration for Codabar bar codes.
CODE128INFO	1223 scanner configuration for Code 128 bar codes.
CODE39INFO	1223 scanner configuration for Code 39 bar codes.
CODE93INFO	1223 scanner configuration for Code 93 bar codes.
D2OF5INFO	1223 scanner configuration for D 2 of 5 bar codes.
GENERALINFO	1223 general scanner configuration.
I2OF5INFO	1223 scanner configuration for I 2 of 5 bar codes.
MSIINFO	1223 scanner configuration for MSI bar codes.
PRINTINIT	Font storage memory
SCANINFO	1222 scanner configuration (not to be used with SCAN1223.LIB).
UPCEANINFO	1223 scanner configuration for UPC/EAN bar codes.

The data structure names are case-sensitive.

CODABARINFO

SCAN1223.LIB only. The CODABARINFO data structure contains 1223 scanner configuration information about Codabar bar codes. To read these values, applications call scnGetCodabarInfo; to set these values, they call scnSetCodabarInfo.

```
typedef struct _CodabarInfo
{
  unsigned char uchEnable;
  unsigned char uchLength1;
  unsigned char uchLength2;
  unsigned char uchEnableCLSIEdit;
  unsigned char uchEnableNOTISEdit;
} CODABARINFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan Codabar bar codes. **Default:** *SCN_DISABLE*
uchLength1 *uchLength2*	Specifies lengths (including start and stop characters) for Codabar bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. Enter values as follows: **Option** — ***uchLength1*** — ***uchLength2*** 1 Length — Value — 0 2 Lengths — Higher Value — Lower Value Range — Min. Value — Max. Value Any length — 0 — 0 **Default:** *5* (*uchLength1*) and *55* (*uchLength2*)

Field	Description
uchEnableCLSIEdit	Enable/disable the ability to strip the start and stop characters from 14-character Codabar bar codes and insert a space after the first, fifth, and tenth characters. **Default:** *SCN_DISABLE*
uchEnableNOTISEdit	Enable/disable the ability to strip the start and stop characters from Codabar bar codes. **Default:** *SCN_DISABLE*

CODE128INFO

SCAN1223.LIB only. The CODE128INFO data structure contains 1223 scanner configuration information about Code 128 bar codes. To read these values, applications call scnGetCode128Info; to set these values, they call scnSetCode128Info.

```
typedef struct _Code128Info
{
   unsigned char uchEnableUSS128;
   unsigned char uchEnableUCCEAN128;
   unsigned char uchEnableISBT128;
} CODE128INFO;
```

Field	Description
uchEnableUSS128	Enable/disable the ability to scan Code 128 bar codes. **Default:** *SCN_ENABLE*
uchEnableUCCEAN128	Enable/disable the ability to scan UCC/EAN-128 bar codes. **Default:** *SCN_ENABLE*
uchEnableISBT128	Enable/disable the ability to scan ISBT 128 bar codes. **Default:** *SCN_ENABLE*

CODE39INFO

SCAN1223.LIB only. The CODE39INFO data structure contains 1223 scanner configuration information about Code 39 bar codes. To read these values, applications call scnGetCode39Info; to set these values, they call scnSetCode39Info.

```
typedef struct _Code39Info
{
  unsigned char uchEnable;
  unsigned char uchEnableTrioptic;
  unsigned char uchCvtC39toC32;
  unsigned char uchEnableC32Prefix;
  unsigned char uchLength1;
  unsigned char uchLength2;
  unsigned char uchVerifyCheckDigit;
  unsigned char uchXmitCheckDigit;
  unsigned char uchEnableFullASCII;
} CODE39INFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan Code 39 bar codes. **Default:** *SCN_ENABLE*
uchEnableTrioptic	Enable/disable the ability to scan Trioptic Code 39 bar codes. Do not enable *uchEnableTrioptic* and *uchEnableFullASCII* at the same time. **Default:** *SCN_DISABLE*
uchCvtC39toC32	Enable/disable the ability to convert Code 39 bar codes to Code 32 bar codes. You must enable *uchEnable* when enabling this parameter. **Default:** *SCN_DISABLE*

Field	Description
uchEnableC32Prefix	Enable/disable the ability to add "A" as a prefix to all Code 32 bar codes. You must enable *uchCvtC39toC32* when enabling this parameter. **Default:** *SCN_DISABLE*
uchLength1 *uchLength2*	Specifies lengths (including check digits) for Code 39 bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. If *uchEnableFullASCII* is enabled, a range or any length is preferred. Enter values as follows: **Option** ***uchLength1*** ***uchLength2*** 1 Length Value 0 2 Lengths High Value Low Value Range Min. Value Max. Value Any length 0 0 **Default:** *2* (*uchLength1*) and *55* (*uchLength2*)
uchVerifyCheckDigit	Enable/disable the ability to check the integrity of Code 39 bar codes. When this parameter is enabled, only Code 39 symbols with a modulo 43 check digit are decoded. **Default:** *SCN_DISABLE*
uchXmitCheckDigit	Enable/disable the ability to transmit check digits with the data. **Default:** *SCN_DISABLE*
uchEnableFullASCII	Enable/disable the ability to scan Full ASCII Code 39 bar codes. The scanner cannot distinguish Code 39 bar codes from Full ASCII Code 39 bar codes. Do not enable *uchEnableTrioptic* and *uchEnableFullASCII* at the same time. **Default:** *SCN_DISABLE*

CODE93INFO

SCAN1223.LIB only. The CODE93INFO data structure contains 1223 scanner configuration information about Code 93 bar codes. To read these values, applications call scnGetCode93Info; to set these values, they call scnSetCode93Info.

```
typedef struct _Code93Info
{
  unsigned char uchEnable;
  unsigned char uchLength1;
  unsigned char uchLength2;
} CODE93INFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan Code 93 bar codes. **Default:** *SCN_DISABLE*
uchLength1 *uchLength2*	Specifies lengths (including check digits) for Code 93 bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. Enter values as follows: **Option** — ***uchLength1*** — ***uchLength2*** 1 Length — Value — 0 2 Lengths — Higher Value — Lower Value Range — Min. Value — Max. Value Any length — 0 — 0 **Default:** *4* (*uchLength1*) and *55* (*uchLength2*).

D2OF5INFO

SCAN1223.LIB only. The D2OF5INFO data structure contains 1223 scanner configuration information about D 2 of 5 bar codes. To read these values, applications call scnGetD2of5Info; to set these values, they call scnSetD2of5Info.

```
typedef struct _D2of5Info
{
  unsigned char uchEnable;
  unsigned char uchLength1;
  unsigned char uchLength2;
} D2OF5INFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan D 2 of 5 bar codes. **Default:** *SCN_DISABLE*
uchLength1 *uchLength2*	Specifies lengths (including check digits) for D 2 of 5 bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. Enter values as follows:

Option	***uchLength1***	***uchLength2***
1 Length	Value	0
2 Lengths	Higher Value	Lower Value
Range	Min. Value	Max. Value
Any length	0	0

Selecting the any length option may lead to mis-scans.
Default: *12* (*uchLength1*) and *0* (*uchLength2*)

GENERALINFO

SCAN1223.LIB only. The GENERALINFO data structure contains general 1223 scanner configuration information. To read these values, applications call scnGetGeneralInfo; to set these values, they call scnSetGeneralInfo.

```
typedef struct _GeneralInfo
{
  unsigned char uchLaserOnTime;
  unsigned char uchPowerMode;
  unsigned char uchTriggerMode;
  unsigned char uchSameSymbolTMO;
  unsigned char uchLinearCodeSecur;
  unsigned char uchBiDirRedun;
} GENERALINFO;
```

Field	Description
uchLaserOnTime	The maximum time (in increments of .1 seconds) of a scan. Values are *5-99*. **Default:** *30*
uchPowerMode	Specifies whether power remains on or the scanner goes into low power mode after a scan. If *uchTriggerMode* is *1* (Continuous) and *uchPowerMode* is *1* (Low Power), the scanner remains continually on. Values are: *0* Continually On *1* Low Power **Default:** *1*
uchTriggerMode	The method that starts the scanner. If *uchTriggerMode* is *1* (Continuous) and *uchPowerMode* is *1* (Low Power), the scanner remains continually on. Values are: *0* Trigger *1* Continuous (always on) **Default:** *0*

Field	Description
uchSameSymbolTMO	The minimum time that must elapse between scans of the same bar code (in increments of .1 seconds). You must set *uchTriggerMode* to 1 when setting this parameter. Values are *1-99*. **Default:** *10*
uchLinearCodeSecur	The scan security level for linear bar codes. Values are *1-4*. Select a higher level for lower quality bar codes. See "Scan Security Levels" for more information. **Default:** *1*
uchBiDirRedun	Enable/disable the requirement to scan bar codes in both directions (forward and reverse). **Default:** *SCN_DISABLE*

Scan Security Levels

The following table describes the security levels used with the uchLinearCodeSecur parameter.

Level Number	Description
1	The following bar code types (provided they meet the specified length requirements) must be scanned successfully twice: Codabar — Any length MSI — 4 characters or less D 2 of 5 — 8 characters or less I 2 of 5 — 8 characters or less
2	All bar code types of all lengths must be scanned successfully twice.

Level Number	Description
3	Bar code types other than the following (or these bar codes, as long as they do not meet the length specification) must be scanned successfully twice: MSI 4 characters or less D 2 of 5 8 characters or less I 2 of 5 8 characters or less
4	All bar code types of all lengths must be scanned successfully three times.

I2OF5INFO

SCAN1223.LIB only. The I2OF5INFO data structure contains 1223 scanner configuration information about I 2 of 5 bar codes. To read these values, applications call scnGetI2of5Info; to set these values, they call scnSetI2of5Info.

```
typedef struct _I2of5Info
{
   unsigned char uchEnable;
   unsigned char uchLength1;
   unsigned char uchLength2;
   unsigned char uchChkDgtAlgorithm;
   unsigned char uchXmitCheckDigit;
   unsigned char uchCvtI2of5toEAN13;
} I2OF5INFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan I 2 of 5 bar codes. **Default:** *SCN_ENABLE*
uchLength1 *uchLength2*	Specifies lengths (including check digits) for I 2 of 5 bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. Enter values as follows: **Option** — ***uchLength1*** — ***uchLength2*** 1 Length — Value — 0 2 Lengths — High Value — Low Value Range — Min. Value — Max. Value Any length — 0 — 0 Selecting the any length option may lead to mis-decodes. **Default:** *14* (*uchLength1*) and *0* (*uchLength2*).

Field	Description
uchChkDgtAlgorithm	Specifies whether the scanner should check the integrity of I 2 of 5 bar codes to ensure they comply with either the Uniform Symbology Specification (USS) or Optical Product Code Council (OPCC) algorithms. Values are: *0* Do not check the integrity *1* Check the integrity against the USS algorithm *2* Check the integrity against the OPCC algorithm **Default:** *0*
uchXmitCheckDigit	Enable/disable the requirement to transmit check digits with the data. **Default:** *SCN_DISABLE*
uchCvtI2of5toEAN13	Enable/disable the requirement to convert 14-character I 2 of 5 bar codes into EAN13 bar codes, and transmit them as EAN13 bar codes. To use this parameter, enable *uchEnable*, set the length to 14, and ensure the data has a leading zero and a valid EAN 13 check digit. **Default:** *SCN_DISABLE*

MSIINFO

SCAN1223.LIB only. The MSIINFO data structure contains 1223 scanner configuration information about MSI bar codes. To read these values, applications call scnGetMSIInfo; to set these values, they call scnSetMSIInfo.

```
typedef struct _MSIInfo
{
   unsigned char uchEnable;
   unsigned char uchLength1;
   unsigned char uchLength2;
   unsigned char uchCheckDigits;
   unsigned char uchXmitCheckDigit;
   unsigned char uchChkDgtAlgorithm;
} MSIINFO;
```

Field	Description
uchEnable	Enable/disable the ability to scan MSI bar codes. **Default:** *SCN_DISABLE*
uchLength1 *uchLength2*	Specifies lengths for MSI bar codes. You can specify one or two lengths, a range of lengths, or that any length is valid. Enter values as follows: **Option** — ***uchLength1*** — ***uchLength2*** 1 Length — Value — 0 2 Lengths — High Value — Low Value Range — Min. Value — Max. Value Any length — 0 — 0 **Default:** *6* (*uchLength1*) and *55* (*uchLength2*).

Field	Description
uchCheckDigits	The number of check digits to use with the bar codes. Values are: *0* One check digit *1* Two check digits. For this value, you must also set *uchChkDgtAlgorithm*. **Default:** *0*
uchXmitCheckDigit	Enable/disable the requirement to transmit data with the check digit. **Default:** *SCN_DISABLE*
uchChkDgtAlgorithm	Specifies the check digit algorithm to use. Values are: *0* Mod 10/Mod 11 *1* Mod 10/Mod 10 **Default:** *1*

PRINTINIT

The PRINTINIT data structure contains information about font storage memory. Use this structure for any font other than the internal font (ID #2). An application uses these values when it calls pclInit to initialize the Print subsystem.

```
typedef struct tagPRINTINIT
{
  unsigned char far* lpuchFntAddr;
  unsigned long ulFntSize;
} PRINTINIT;
```

Field	Description
lpuchFntAddr	Pointer to an area of memory for font storage.
ulFntSize	The amount of memory for font storage (specified in bytes).

Set both fields to zero if the application uses only the internal font.

Font Memory Requirements

Following are the memory requirements (listed in bytes) for the base fonts.

Font ID	Memory Needed
1000	7176
1001	8312
1002	10,744
1003	14,440
1004	6824
1005	9064
1006	5912
1007	8200
1008	10,056

Font ID	Memory Needed
1009	12,440
1010	6280
1011	8392
1012	5272
1013	9112

To determine the memory needed for fonts created with the Font Utility:

1. Open the font file in any text editor.
2. Note the last number on the font packet's first line. For example, in the following line, the number in question is 386.

   {W,1000,A,R,386|

3. Use the number from the previous step in the following calculation:

   (*number* * 16) + 1000

4. Use the result of this calculation as the number of bytes needed for font memory.

SCANINFO

The SCANINFO data structure contains configuration information for the 1222 scanner. To read these values, applications call scnGetScanInfo; to set these values, they call scnSetScanInfo.

You cannot use this data structure with SCAN1223.LIB.

```
typedef struct _ScanInfo
{
  unsigned char uchCode39
  unsigned char uchCodabar;
  unsigned char uchCode128;
  unsigned char uchI2of5;
  unsigned char uchUPCEAN;
  unsigned char uchUPCEAN_2;
  unsigned char uchUPCEAN_5;
  unsigned char uchMSI;
  unsigned char uchContinuousMode;
  unsigned char uchMsiCheckDigits;
  unsigned char uchCode128FunctChars;
  unsigned char uchUPCEANAuto;
  unsigned char uchLowPower;
  unsigned char uchTimeOut;
  unsigned char uchMSICDMod11;
  unsigned char uchUPCECD;
  unsigned char uchUPCACD;
  unsigned char uchUPCPSysOnly;
  unsigned char uchUPCPNone;
  unsigned char uchI2of5Len1;
  unsigned char uchI2of5Len2;
} SCANINFO;
```

Field	Description
uchCode39	Enable/disable the ability to scan Code 39 bar codes. **Default:** *SCN_ENABLE*
uchCodabar7	Enable/disable the ability to scan Codabar bar codes. **Default:** *SCN_ENABLE*
uchCode128	Enable/disable the ability to scan Code 128 bar codes. **Default:** *SCN_ENABLE*
uchI2of5	Enable/disable the ability to scan I 2 of 5 bar codes. The application must also set uchI2of5Len1 or uchI2of5Len2 to the length of the bar code to scan. **Default:** *SCN_ENABLE*
uchUPCEAN	Enable/disable the ability to scan UPCA, UPCE, EAN8, and EAN13 bar codes. This parameter and *uchCode128FunctChars* cannot both be enabled. **Default:** *SCN_ENABLE*
uchUPCEAN_2	Enable/disable the ability to scan all UPC/EAN +2 bar codes (UPCA+2, UPCE+2, EAN8+2, and EAN13+2). **Default:** *SCN_ENABLE*
uchUPCEAN_5	Enable/disable the ability to scan all UPC/EAN +5 bar codes (UPCA+5, UPCE+5, EAN8+5, and EAN13+5). **Default:** *SCN_ENABLE*
uchMSI	Enable/disable the ability to scan MSI bar codes. **Default:** *SCN_ENABLE*

Field	Description
uchContinuousMode	Enable/disable continuous mode scanning. If *uchContinuousMode* is enabled and *uchLowPower* is enabled, the scanner goes into low power mode between scans. **Default:** *SCN_DISABLE*
uchMsiCheckDigits	The number of MSI check digits (either *1* or *2*). **Default:** *1*
uchCode128FunctChars	Enable/disable transmission of Code 128 function characters. This parameter and *uchUPCEAN* cannot both be enabled. **Default:** *SCN_DISABLE*
uchUPCEANAuto	Enable/disable the ability to distinguish UPC/EAN (+2, +5) bar codes from each other. **Default:** *SCN_ENABLE*
uchLowPower	Enables low power mode. If *uchContinuousMode* is enabled and *uchLowPower* is enabled, the scanner goes into low power mode between scans. You should leave this option enabled. **Default:** *SCN_ENABLE*
uchTimeOut	Scanner time-out value in seconds. Values are *1-4*. **Default:** *3*
uchMSICDMod11	The MSI check digit scheme. Values are *0* (Modulo 10/Modulo 10) and *1* (Modulo 11/Modulo 10). **Default:** *0*
uchUPCECD	Enable/disable the return of UPCE check digits with scanned data. **Default:** *SCN_ENABLE*

Field	Description
uchUPCACD	Enable/disable the return of UPCA check digits with scanned data. **Default:** *SCN_ENABLE*
uchUPCPSysOnly	Enable/disable the return of the system character (and not the country code) with UPCE and UPCA scanned data. You must disable *uchUPCPNone* when enabling this parameter. **Default:** *SCN_DISABLE*
uchUPCPNone	Enable/disable the return of the system character and country code with scanned data. **Default:** *SCN_DISABLE*
uchI2of5Len1 and *uchI2of5Len2*	Sets the lengths of the I 2 of 5 bar codes to scan. Values are *0-31*. The application must also set *uchI2of5* to *SCN_ENABLE.* **Default:** *14* (for *uchI2of5Len1*) *0* (for *uchI2of5Len2*)

For the fields where you enable or disable an option, the values are SCN_ENABLE and SCN_DISABLE.

You cannot scan two-dimensional bar codes.

See “scnGetScanInfo” or “scnSetScanInfo” in Chapter 4 for more information.

UPCEANINFO

SCAN1223.LIB only. The UPCEANINFO data structure contains 1223 scanner configuration information about UPC and EAN bar codes. To read these values, applications call scnGetUPCEANInfo; to set these values, they call scnSetUPCEANInfo.

```
typedef struct _UPCEANInfo
{
  unsigned char uchEnableUPCA;
  unsigned char uchEnableUPCE;
  unsigned char uchEnableUPCE1;
  unsigned char uchEnableEAN8;
  unsigned char uchEnableEAN13;
  unsigned char uchEnableBookEAN;
  unsigned char uchEnableSupps;
  unsigned char uchEnableSuppRedun;
  unsigned char uchXmitUPCAChkDgt;
  unsigned char uchXmitUPCEChkDgt;
  unsigned char uchXmitUPCE1ChkDgt;
  unsigned char uchUPCAPreamble;
  unsigned char uchUPCEPreamble;
  unsigned char uchUPCE1Preamble;
  unsigned char uchCvtUPCEtoUPCA;
  unsigned char uchCvtUPCE1toUPCA;
  unsigned char uchEAN8ZeroExtend;
  unsigned char uchCvtEAN8toEAN13;
  unsigned char uchSecurityLevel;
  unsigned char uchEnableCouponCode;
} UPCEANINFO;
```

Field	Description
uchEnableUPCA	Enable/disable the ability to scan UPCA bar codes. **Default:** *SCN_ENABLE*
uchEnableUPCE	Enable/disable the ability to scan UPCE0 bar codes. **Default:** *SCN_ENABLE*

Field	Description
uchEnableUPCE1	Enable/disable the ability to scan UPCE1 bar codes. **Default:** *SCN_DISABLE*
uchEnableEAN8	Enable/disable the ability to scan EAN8 bar codes. **Default:** *SCN_ENABLE*
uchEnableEAN13	Enable/disable the ability to scan EAN13 bar codes. **Default:** *SCN_ENABLE*
uchEnableBookEAN	Enable/disable the ability to scan Bookland EAN bar codes. **Default:** *SCN_DISABLE*
uchEnableSupps	Enable/disable the ability to scan supplemental characters (+2, +5) according to code format conventions (UPCA+2, UPCE0+2, etc.). You can also specify to scan bar codes with or without supplemental characters. Values are: *0* Disable (can scan with or without supplementals, but supplementals are ignored) *1* Enable (cannot scan supplementals) *2* Scan with or without supplementals, but do not ignore the supplementals. **Default:** *SCN_DISABLE*

Field	Description
uchEnableSuppRedun	The number of times to scan bar codes without supplementals before transmission. Five or more is recommended when scanning a mix of UPC/EAN bar codes with and without supplementals. You must set *uchEnableSupps* is set to *2* when setting this parameter. Single digit numbers must have a leading zero. Values are *2-20*. **Default:** *7*
uchXmitUPCAChkDgt	Enable/disable the requirement to transmit UPCA bar codes with the check digit. **Default:** *SCN_ENABLE*
uchXmitUPCEChkDgt	Enable/disable the requirement to transmit UPCE0 bar codes with the check digit. **Default:** *SCN_ENABLE*
uchXmitUPCE1ChkDgt	Enable/disable the requirement to transmit UPCE1 bar codes with the check digit. **Default:** *SCN_ENABLE*
uchUPCAPreamble	Specifies how to transmit lead-in characters for UPCA bar codes. Values are: *0* Do not transmit *1* Transmit system character *2* Transmit system character and country code **Default:** *1*

Field	Description
uchUPCEPreamble	Specifies how to transmit lead-in characters for UPCE0 bar codes. Values are: *0* Do not transmit *1* Transmit system character *2* Transmit system character and country code **Default:** *1*
uchUPCE1Preamble	Specifies how to transmit lead-in characters for UPCE1 bar codes. Values are: *0* Do not transmit *1* Transmit system character *2* Transmit system character and country code **Default:** *1*
uchCvtUPCEtoUPCA	Enable/disable the requirement to convert UPCE0 bar codes to UPCA bar codes before transmission. After conversion, the bar code is affected by UPCA programming selections. **Default:** *SCN_DISABLE*
uchCvtUPCE1toUPCA	Enable/disable the requirement to convert UPCE1 bar codes to UPCA bar codes before transmission. After conversion, the bar code is affected by UPCA programming selections. **Default:** *SCN_DISABLE*
uchEAN8ZeroExtend	Enable/disable the addition of five leading zeros to scanned EAN8 bar codes to make them compatible with EAN13 bar codes. Values are *1* (enable) or *2* (disable). **Default:** *SCN_DISABLE*

Field	Description
uchCvtEAN8toEAN13	Enable/disable the ability to convert EAN8 zero-extended bar codes to EAN13 format. You must enable *uchEAN8ZeroExtend* when enabling this parameter. **Default:** *SCN_ENABLE*
uchSecurityLevel	The scan security level for UPC/EAN bar codes. See "Scan Security Levels" for descriptions of each level. Values are *0-3*. **Default:** *0*
uchEnableCouponCode	Enable/disable the ability to scan UPCA, UPCA+2, UPCA+5, and UPC/EAN128 bar codes. You must set *uchEnableSupps* to *2* when enabling this parameter. **Default:** *SCN_DISABLE*

Scan Security Levels

The security level specifies how aggressive the scanner works during a scan. With a low bar code quality, the scanner must work more aggressively, and vice versa. Choose the minimum security level you need, according to the following guidelines:

Level	Description
0	Provides security sufficient for bar codes that meet specifications.
1	Provides security sufficient for bad scans that are limited to characters 1, 2, 7, and 8.
2	Provides security sufficient for bad scans that are not limited to characters 1, 2, 7, and 8.
3	Provides security for continued bad scans after you increase security to level 2.

You may need to adjust the security as needed during scanning.

PROGRAMMING TECHNIQUES 6

Many applications have common features, such as requiring the operator to press the trigger to initiate processing.

This chapter describes the code you write to add such common features to your application. It includes code for the following features:

- Printing Labels
- Pausing While Printing
- Loading Multiple Packets Together
- Building Packets Dynamically
- Using the Scanner
- Reading Trigger Presses

Printing Labels

An application prints labels by submitting MPCLII packets to the Print subsystem. At a minimum, the application must submit format and batch packets. To submit these packets, use either pclWrite or pclOpen.

For more information, see “pclWrite” or “pclOpen” in Chapter 4. For information about MPCLII packets, refer to the MPCLII Packet Reference Manual.

A batch packet starts a print job, which makes an asynchronous call to the Print subsystem. After submitting a print job, the application should call pclStatus in a loop, waiting until the printer becomes free. See “Pausing While Printing” for more information.

An application can

- print single labels
- print multiple labels
- reprint labels.

No matter what printing method it uses, the application must

1. Initialize the Print subsystem using pclInit.
2. Calibrate the supplies using pclCalibrate or pclCalibratePaper (if using supplies other than fax paper).
3. Call pclPaperSetup if pclCalibratePaper was called in the previous step.
4. Print using any method listed above.
5. Close the Print subsystem with pclClose.

Printing Single Labels

To print single labels, send a format and a batch (with a quantity of 1) to the printer. SAMPLE1.C in the Samples sub-directory illustrates printing single labels.

Printing Multiple Labels

An application can multiple labels in a strip or print loop. When setting up a method to print multiple labels, you must specify

- quantity
- peel mode/on-peel mode
- feed mode.

Quantity

Use either or both of the following methods to control the quantity of labels printed:

- Set up a print loop by placing any C/C++ loop construct around the code that prints the labels. The number of times the loop executes is the number of labels printed.
- Adjust the batch packet's quantity parameter before submitting the batch. Refer to the MPCLII Packet Reference Manual for more information.

Peel Mode/Non-Peel Mode

In peel mode, the printer separates labels from the backing paper, which allows you to apply labels immediately. In non-peel mode, the printer feeds the supply through the printer in a continuous strip. In a print loop, the printer may or may not be in peel mode. The mode you want depends on how you load the supplies in the printer. Refer to the Equipment Manual for more information.

You cannot use linerless supplies in peel mode.

Feed Mode

Feed mode determines how the printer prints the labels. There are two feed modes:

Mode	Description
Continuous	The printer prints all labels together (in one strip) immediately, with no operator intervention.
On-Demand	The printer prints the labels one at a time. It does not print the next label until the operator removes the previous one.

You specify the feed mode in the batch packet with the batch control line's second parameter. Refer to the MPCLII Packet Reference Manual for more information.

Reprinting Labels

To reprint labels, submit the following batch with pclWrite:

```
{B, format, U, quantity|}
```

where format is the format number and quantity is the number of labels to print.

Pausing While Printing

After submitting a packet that prints labels, an application should pause until the printer becomes free. To check if the printer is busy, the application calls pclStatus. By calling pclStatus in a loop, the application pauses until the printer finishes.

The following code illustrates how to pause the application using pclStatus.

```
while ((iStatus = pclStatus()) == 1);
```

See "pclStatus" in Chapter 4 for more information.

Loading Multiple Packets Together

The application can load multiple packets together after it initializes the Print subsystem. You can create one or more text files containing any number of packets, and then pass those file names to pclOpen. The application must call pclOpen only once per file.

If the packet file contains a format and a batch, the call to pclOpen also prints labels.

SAMPLE4.C in the Samples sub-directory illustrates loading multiple packets together.

Building Packets Dynamically

An application can use fixed packets or packets that change every time they are used, such as in an application where the operator enters the quantity for the batch. When packets change every time they are used, the application must build the packet dynamically.

SAMPLE3.C in the Samples sub-directory illustrates building packets dynamically.

Using the Scanner

An application uses the printer's scanner (either the 1222 or the 1223) to read bar codes as follows:

1. Enable the scanner with scnScannerOpen or scnOpenScannerShared.

2. If necessary, configure the scanner using the scnGetxxx and scnSetxxx functions.

3. Use any scanner function to operate the scanner. These functions are described in Chapter 4 and all begin with an scn prefix.

4. Disable the scanner with scnCloseScanner.

SAMPLE5.C in the Samples sub-directory illustrates how to use the scanner.

Reading Trigger Presses

Pressing any key (including the trigger) sends a code to the application for it to read. The trigger emulates an F11 as an extended character code. To read a trigger press, the application reads two characters. If the first is character is a zero and the second is 0x85, the operator pressed the trigger.

SAMPLE1.C in the Samples sub-directory illustrates reading trigger presses.

UTILITY AND DRIVER REFERENCE

7

The SDK contains several utilities and drivers that you may need when creating an application. For example, FLASH is the utility you use to load the application and related files into the printer.

This chapter describes these utilities and drivers. It lists them alphabetically. Following are the utilities and drivers discussed:

- DUMP
- FLASH
- LCD
- ONLINE
- RDISK.SYS
- REMDISK
- REMQUIT
- REMSERV
- ROMDISK
- SDISK.SYS

DUMP

Description

The DUMP Utility displays the contents of binary files or text files in hexadecimal notation or real-mode address memory. It also includes the corresponding ASCII characters. To run this utility, enter the DUMP command at the PC's DOS prompt.

If your file is large or you only want to look its beginning, use the DOS redirection operator (>) to store the output in a text file.

Do Not redirect the output to a flash ROM. Or, if you have Windows 95, do not redirect the output by running this utility on the printer and directing the file to the PC.

Use the DOS type command or a text editor on the PC to look at the file. See the second example below.

Syntax

DUMP *file* [*offset* [*length*]]

DUMP */address* [*length*]

Parameters

file	The file to display.
offset	The hexadecimal offset into the file to start the display from. The default is 0.

/address The hexadecimal address to start memory display. You must specify the address in *segment:offset* form. You can also use the following options to indicate you want to dump a flash ROM:

/A or */1* The first flash ROM.

/B or */2* The second flash ROM.

length The hexadecimal number of bytes to display. The default is the entire file or 80 hexadecimal (128 decimal) bytes if displaying memory.

Examples

```
DUMP DISK.IMG
```

Displays the entire DISK.IMG file.

```
DUMP DISK.IMG 200 140>TEMP.TXT
```

Displays 320 bytes (140 hexadecimal) of file DISK.IMG, starting at offset 512 (200 hexadecimal), redirecting output to a file called TEMP.TXT.

```
DUMP /E000:0000
```

Displays 128 (80 hexadecimal) bytes of memory starting at address E0000.

```
DUMP/CC00:200 400
```

Displays 1024 (400 hexadecimal) bytes of memory at address CC200.

```
DUMP /FFF0:0 100>TEMP.TXT
```

Displays 256 (100 hexadecimal) bytes of memory at address FFF00, redirecting output to file TEMP.TXT.

FLASH

Description

The FLASH Utility loads a disk image into the printer, using the specified flash ROM.

To use this utility, you must include it in the disk image.

For more information, see "Loading Disk Images into the Printer" in Chapter 3.

Syntax

FLASH *imagefile* [*options*...]

Parameters

imagefile A fully-qualified path to the disk image file. If the image file is in the same directory as FLASH.EXE, give only the file name.

You must use the drive that the printer recognizes. For example, if the files are on the PC's C: drive, but the printer refers to it as B:, use B: in the path.

options... One or more of the following options. Note these options are case-sensitive.

/A or */1*	Loads the first flash ROM (default).
/B or */2*	Loads the second flash ROM.
/R:address	Reads the block at *address*. It fills the display with the data. Press Enter to display the next group of data. To quit, press Esc. This option is a debugging tool.

Examples

```
FLASH DISK1.IMG
```

Loads DISK1.IMG into the first flash ROM at address 0.

```
FLASH DISK1.IMG /2
```

Loads DISK1.IMG into the second flash ROM.

LCD

Description

Use the LCD Utility to

- set the display speed
- turn the display on or off
- turn the backlight on or off.

You must reference this utility in AUTOEXEC.BAT to enable the display before you run your application.

To use this utility, you must include it in the disk image.

Syntax

LCD [*options*...]

Options

You can enter one or more of the following options on the command line.

Y	Enable the display. This option is the default.
N	Disable the display.
F	Set the display speed to fast.
S	Set the display speed to slow.
C	Clear display. Using this command is equivalent to using the cls command.
D	Backlight off.
L	Backlight on.

Examples

```
LCD
```

Enable the display without adjusting the speed.

```
LCD Y
```

Enable the display without adjusting the speed.

```
LCD Y F
```

Enable the display and make the speed fast.

ONLINE

Description

To test any MPCLII packets you create, use the ONLINE Utility with Label Designer to perform test prints. The communication parameters specified in the command line must match the parameters specified in Label Designer. This utility supports the same MPCLII packets as the printer.

To run this utility, enter the command at the printer's DOS prompt. While it is running, you can press the following keys:

Key	Description
Fct + 1	Calibrates the supplies in the printer.
C	Clears a printer error.
Esc	Exits the utility.

Refer to the MPCLII Packet Reference Manual or Label Designer Online Help for more information.

To use this utility, you must include it in the disk image.

Syntax

ONLINE [*options*...]

Options

You can enter one or more of the following options on the command line.

/B*baud* Baud rate. Values are *300*, *1200*, *2300*, *4800*, *9600*, *19200*, *38400*, *57600*, and *115k*. The default is 19200.

/P*parity* Parity. Values are *N* (none), *0* (odd), *E* (even), *M* (mark), and *S* (space). The default is N.

/W*len* Data length in bits. Values are *7* and *8*. The default is 8.

/S*stop* Stop bits. Values are *1* and *2*. The default is 1.

/F*control* Flow control. Values are *N* (none), *X* (XON/XOFF), and *P* (DTR). The default is X.

/L:*file* The files containing MPCLII packets (for fonts, etc.) to pre-load when the utility starts. You can specify this option multiple times on a command line. Standard DOS wild cards (*, ?) are allowed.

/M:*size* The amount of memory in kilobytes to reserve for fonts. Values are *0-128*. The default is 64.

Examples

```
ONLINE
```

Runs ONLINE at 19200 baud, with no parity, 8 data bits, 1 stop bit, XON/XOFF flow control, and 64K for font storage memory.

```
ONLINE /B9600 /PO /W7 /S2 /FP
```

Runs ONLINE at 9600 baud with odd parity, 7 data bits, 2 stop bits, DTR flow control, and 64K for font storage memory.

```
ONLINE /M:100 /L:*.PCL
```

Runs ONLINE at 19200 baud with no parity, 8 data bits, 1 stop bit, and XON/XOFF flow control. It pre-loads all .PCL files and reserves 100K for font storage memory.

RDISK.SYS

Description

To use the second flash ROM, use the RDISK.SYS driver, as follows:

1. Enter the line listed below (under “Syntax”) in CONFIG.SYS.
2. Load the first flash ROM’s disk image (including RDISK.SYS and CONFIG.SYS in it).
3. Load the disk image for the second flash ROM.

See “Loading Disk Images into the Printer” in Chapter 3 for more information.

Syntax

```
DEVICE=RDISK.SYS
```

REMDISK

Description

The REMDISK Utility (in conjunction with the REMSERV Utility) allows the printer to access a drive on your PC. This utility uses absolute sector addressing.

To run REMDISK, enter the REMDISK command

- in CONFIG.SYS with a DEVICE command.
- in a batch file.
- at the printer's DOS prompt.

To use this utility, you must include it in the disk image.

REMDISK and REMSERV must use the same baud rate and transmission style (packet/non-packet).

DOS assigns drive letters in the order you specify the entries in CONFIG.SYS. If you use REMDISK outside of CONFIG.SYS, DOS assigns it the next letter after the ones specified in CONFIG.SYS.

See "Using REMSERV and REMDISK" in Chapter 3 for more information.

Syntax

REMDISK [*options*...]

Options

You can enter one or more of the following options on the command line. In most environments, you can use the defaults for each parameter.

/B*nnnn*	Specifies to use baud rate *nnnn*. Values for *nnnn* are: *300*, *1200*, *2300*, *4800*, *9600*, *19200*, *38400*, *57600*, and *115k*. The default is 115k.
+	Specifies to use packet transmissions. If you do not specify this option, the default depends on the baud rate. For baud rates over 19200, REMDISK uses packet transmissions. For baud rates at or below 19200, REMDISK uses non-packet transmissions.
/U	Specifies to unload REMDISK, disabling the drive letter and freeing memory. You cannot use this option if you are running REMDISK from CONFIG.SYS or if you have any other memory-resident programs. You must change drives before you run REMDISK with this option. Otherwise, the drive becomes invalid.
/T*nnnn*	Specifies *nnnn* as the time-out in seconds. Values are *2-3640*.
/?	Displays a short help screen.

Help Screen Notes

Note the following help screen discrepancies that appear if you use the /? option:

- The /n before the /U option should not appear.
- The /H option appears on the screen, but is non-functional. Do not use it.

Examples

```
REMDISK
```

Runs REMDISK at 115k baud, using packet transmission.

```
REMDISK /B9600
```

Runs REMDISK at 9600 baud, with non-packet transmission.

```
DEVICE=C:\REMDISK.EXE /B19200 +
```

Runs REMDISK from CONFIG.SYS at 19200 baud, using packet transmission.

```
REMDISK /U
```

Unloads REMDISK.

REMQUIT

The REMQUIT Utility ends the REMSERV Utility. You enter the REMQUIT command at the DOS prompt on the printer.

To use this utility, you must include it in the disk image.

You can also end REMSERV by pressing any key on the PC.

Before you end REMSERV, change the printer's current drive to one other than the shared PC drive.

Syntax

REMQUIT

REMSERV

Description

The REMSERV Utility (in conjunction with the REMDISK Utility) allows the printer to access a drive on your PC. This utility uses absolute sector addressing.

To run this utility, enter the REMSERV command at the DOS prompt on the PC. To quit REMSERV from the PC, press any key. To quit REMSERV from the printer, enter the REMQUIT command.

REMDISK and REMSERV must use the same baud rate and packet/non-packet transmission style.

See “Using REMSERV and REMDISK” in Chapter 3 for more information.

Using REMSERV with FAT32 Systems

If you use REMSERV to share a FAT32 PC disk drive with the printer, an Invalid Drive error occurs because REMSERV understands only the FAT16 architecture. To avoid this problem

- use a separate hard drive or floppy disk formatted as a FAT16 system.
- create a partition with a FAT16 system on your FAT32 hard drive.

Then you can use the FAT16 system as the drive shared with your printer.

Syntax

REMSERV *drive* [*options*...]

Parameters

drive The letter of the drive to make available.

options... Any of the following options. In most environments, you can use the defaults for each option.

/B*nnnn*	Specifies to use baud rate *nnnn*. Values for *nnnn* are: *300*, *1200*, *2300*, *4800*, *9600*, *19200*, *38400*, *57600*, and *115k*. The default is 115k.
+	Specifies to use packet transmissions. If you do not specify this option, the default depends on the baud rate. For baud rates over 19200, REMDISK uses packet transmissions. For baud rates at or below 19200, REMDISK uses non-packet transmissions.
/COM1	Specifies to use the COM1 serial communications port.
/S	Specifies to run REMSERV silently without any screen output.
/T*nnnn*	Specifies *nnnn* as the time-out in seconds. Values are *3-3640*.
/?	Displays a short help screen

Help Screen Notes

Note the following help screen discrepancies that appear if you use the /? option:

- The /n before the example should not appear.
- The /H option appears on the screen, but is non-functional. Do not use it.
- You can press any key to exit REMSERV, not only Esc.

Examples

```
REMSERV C:
```

Makes drive C available, using 115k baud and packet transmissions.

```
REMSERV D:/B9600
```

Makes drive D available, using 9600 baud and non-packet transmission.

ROMDISK

Description

To create a disk image, enter the ROMDISK command at the PC's DOS prompt.

ROMDISK displays statistics about the newly-created image file when it is done. Before continuing, review those statistics carefully to ensure the image file was created correctly.

Syntax

ROMDISK [*filespec* [*outfile*] [*options*...]]

Parameters

filespec — The directory on the PC's hard disk that contains the files and optional sub-directories to include in the disk image. *Filespec* becomes the disk image's root directory.

outfile — A fully-qualified path for the disk image file. If the file is in the same directory as ROMDISK.EXE, specify only the file name. The default is ROMDISK.IMG and the current directory.

You must use the drive that the printer recognizes. For example, if the files are on the PC's C: drive, but the printer refers to it as B:, use B: in the path.

options... Any of the following options:

/S	Specifies to include sub-directories of *filespec* in the disk image. Try not to use too many sub-directories to save space.
/T	Specifies to display statistics, but not to create the image file. You might use this option to verify that the files fit in the required space.
/V"label"	Sets the image's volume label to *label*. This label cannot be longer than 11 characters. The default is ROMDISK.

Help Screen Notes

You can view a help screen by entering the ROMDISK command with no parameters. Do not use the following options shown on the screen: /D<seg>, /E, /F#, /H#, /I#, /O, /R#, and /Z#.

Example

```
ROMDISK \APP DISK1.IMG /S /V"MARKDOWN"
```

Places the APP directory's contents (including sub-directories) into the disk image file DISK1.IMG, which has MARKDOWN for a volume label.

SDISK.SYS

Description

To create a virtual RAM disk drive using memory above the lower 640K, use the SDISK.SYS device driver in CONFIG.SYS. The printer preserves this virtual drive (assigned in the order specified in CONFIG.SYS) and its files while the printer is turned off.

To use this driver, you must include it in the disk image.

Syntax

DEVICE=SDISK.SYS[*disksize*[*sectorsize*[*rootentries*]]] [/R]

Parameter	Description
disksize	The RAM disk's size in kilobytes. Values are *64* and *128*. The default is 128.
sectorsize	The sector size in bytes. Values are *128*, *256*, *512*, and *1024*. The default is 512.
rootentries	The number of entries in the root directory. Values are *2-1024*. The default is 64.
/R	Specifies the driver should initialize memory with an empty file system at every startup.

In most cases, you should use the defaults for disksize, sectorsize, and rootentries.

SAMPLE APPLICATIONS

A

The SDK contains several sample applications for you to study as you write your own applications. These samples are located in the SDK's samples sub-directory. This appendix describes these samples.

Following are the samples discussed:

Sample #	Description
1	Prints with the trigger.
2	Prints with the on-demand sensor.
3	Prints a strip of labels.
4	Prints using fonts and formats loaded with pclOpen.
5	Uses the scanner.
6	Scans and prints.

Sample 1

Function

Prints with the trigger.

Algorithm

1. Initializes the Print subsystem using pclInit without allocating font storage memory.
2. Calibrates the supplies with pclCalibrate.
3. Loads fixed format and batch packets with pclWrite. The quantity is 0, so the sample images the format, but does not print it.
4. Waits for the operator to press the trigger.
5. Uses pclBatteryOktoPrint to check whether the battery is charged enough for printing.
6. Loads the batch update packet with pclWrite.
7. Waits until the label prints by checking pclStatus repeatedly until it returns something other than “busy.”
8. Closes the Print subsystem.

Sample 2

Function

Prints using the on-demand sensor.

Algorithm

1. Initializes the Print subsystem using pclInit without allocating font storage memory.
2. Calibrates the supplies with pclCalibrate.
3. Loads a fixed format with pclWrite. This format defines the feed mode as on-demand.

4. Uses pclBatteryOktoPrint to check whether the battery is charged enough for printing.

5. Loads a fixed batch with pclWrite.

6. Waits until the label prints by checking pclStatus repeatedly until it returns something other than “busy.”

7. Closes the Print subsystem with pclClose.

Sample 3

Function

Prints a strip of labels.

Algorithm

1. Initializes the Print subsystem using pclInit without allocating font storage memory.

2. Calibrates the supplies with pclCalibrate.

3. Loads a fixed format with pclWrite.

4. Uses pclBatteryOktoPrint to check whether the battery is charged enough for printing.

5. Prompts the operator to enter a quantity and builds the batch dynamically with that number and other data.

6. Loads a batch with pclWrite.

7. Waits until the labels print by checking pclStatus repeatedly until it returns something other than “busy.”

8. Closes the Print subsystem with pclClose.

Sample 4

Function

Prints using fonts and formats loaded with pclOpen.

Algorithm

1. Initializes the Print subsystem with pclInit, allocating 24,000 bytes of font storage memory.
2. Calibrates the supplies with pclCalibrate.
3. Loads FONTS.PCL with pclOpen. This file contains an MPCLII font packet.
4. Loads FORMATS.PCL with pclOpen. This file contains an MPCLII format packet.
5. Loads a fixed batch packet with pclWrite. The quantity in the batch is 0, so the printer images the format, but does not print it.
6. Waits for the operator to press the trigger.
7. Uses pclBatteryOkToPrint to check whether the battery is charged enough for printing.
8. Loads a batch update packet with pclWrite.
9. Waits until the label prints by checking pclStatus repeatedly until it returns something other than “busy.”
10. Closes the Print subsystem with pclClose.

Sample 5

Function

Uses the scanner.

Algorithm

1. Enables the scanner with scnOpenScanner.
2. Waits for the operator to press the trigger.
3. Initiates a scan with scnGets.
4. Disables the scanner with scnCloseScanner.

Sample 6

Function

Scans and prints.

Algorithm

1. Initializes the Print subsystem using pclInit without allocating font storage memory.
2. Calibrates the supplies with pclCalibrate.
3. Loads a fixed format with pclWrite.
4. Enables the scanner with scnOpenScanner.
5. Waits for the operator to press the trigger or Esc. If the operator presses Esc, continue with step 13.
6. Initiates a scan with scnGets.
7. Uses pclBatteryOktoPrint to check whether the battery is charged enough for printing.
8. Prompts the operator to enter a quantity and press either the trigger or Enter. Or, if the operator presses Esc, jump to step 13.

9. Clears the screen and builds the batch dynamically with the entered number and the scanned data.

10. Loads the batch with pclWrite.

11. Waits until the label prints by checking pclStatus repeatedly until it returns something other than “busy.”

12. Returns to step 5.

13. Disables the scanner.

14. Closes the Print subsystem with pclClose.

GLOSSARY

Following is a list of terms you must be familiar with to write printer applications.

Term	Definition
Application Buffer	The programmer-defined buffer in the application used to save the data from a scan.
Attribute	The way a character appears on the printer's display: in normal or reverse video.
Calibrate	The automated process where the printer determines the size, length and type of the supplies it is using.
Disk Image	A simulated diskette loaded into a flash ROM in the printer. It contains all parts of a diskette, including a file allocation table, data files, and the application.
Display Page	Any of four virtual pages that the printer can display (one at a time) on the physical display. An application can write data to a display page behind-the-scenes, and display it when ready.
Flash ROM	A type of memory you load with the disk image you create. It is read-only, except when you load the disk image. The printer has two flash ROMs.
Motion Control subsystem	The printer subsystem that controls how paper feeds through the printer.
MPCLII	Monarch Printer Control Language II. This language describes the commands that drive the printer. Refer to the MPCLII *Packet Reference Manual* for more information.

Term	Definition
Packet	A unit of MPCLII commands. For example, to print a particular label, the application writes a particular group of MPCLII commands to the print subsystem. This group is enclosed in braces and is known as a packet. Refer to the MPCLII *Packet Reference Manual* for more information.
Print subsystem	The part of the printer that controls printing.
REMDISK	A printer utility that allows the printer to access a PC's disk drive via the printer's serial communications port.
REMSERV	A PC utility that makes a disk drive available for the printer to access via the printer's serial communications port.
ROM-DOS	The MS-DOS 6.22-compatible operating system that the printer runs. Datalight manufactures it.
ROMDISK	A PC utility that creates a disk image to load into the flash ROM.
SDK	Software Development Kit. This kit includes everything you need (libraries, utilities, documentation, etc.) to create printer applications.
Scanner Buffer	The buffer internal to the scanner that contains the bar code data immediately after scanning.
Stock Type	The type of supplies you load in the printer. They can be paper, fax, or synthetic.
Supplies	The media that the printer prints on. For example, it can print labels or tags. Supplies can be made of different stock types. See "Stock Type" in this glossary for more information.
Video Mode	The 20 columns on the printer's display.

INDEX

B

C

D

E

F

G

H

I

K

L

M

R

S

T

U

V

W