Programmer's Manual

```
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® Pathfinder® Ultra® *Gold* 6037EX™ Printer

PAXAR
We Make Your Sales Work(s)™

TC6037EXPM Rev. AA 8/03 Printed in the U.S.A.

Each product and program carries a respective written warranty, the only warranty on which the customer can rely. Paxar reserves the right to make changes in the product and the programs and their availability at any time and without notice. Although Paxar has made every effort to provide complete and accurate information in this manual, Paxar shall not be liable for any omissions or inaccuracies. Any update will be incorporated in a later edition of this manual.

WARNING

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

CANADIAN D.O.C. WARNING

This digital apparatus does not exceed the Class A limits for radio noise emissions from digital apparatus set out in the Radio Interference Regulations of the Canadian Department of Communications.

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe A prescrites dans le Réglement sur le brouillage radioélectrique édicte par le ministère des Communications du Canada.

Trademarks

Monarch®, Pathfinder®, Ultra®, are registered trademarks of Monarch Marking Systems, Inc.
6037EX™ is a trademark of Monarch Marking Systems, Inc.
Paxar is a trademark of Paxar Americas, Inc.
Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.
Windows® and Windows 95®, are registered trademarks of Microsoft in the U.S. and other countries.
CG Triumvirate Bold is a trademark of AGFA Corporation.
Datalight® is a registered trademark of Datalight, Inc.
ROM-DOS™ is a trademark of Datalight, Inc.
SystemSoft® is a registered trademark of SystemSoft Corporation.
IBM® is a registered trademark of International Business Machines Corporation.

Paxar Americas, Inc.
170 Monarch Lane
Miamisburg, OH 45342

TABLE OF CONTENTS

INTRODUCTION

1

Welcome to the software development kit (SDK) for the Monarch® Pathfinder® Ultra® *Gold* 6037EX™ printer. The SDK consists of software and 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 printer applications. These programmers should also be familiar with Monarch's MPCL 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 or higher monitor
- 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++)
- Windows 95® (or higher)-compatible CD-ROM drive
- serial port
- a printer-to-PC cable (part number 124054).

Software Requirements

Your PC needs the following software:

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

SDK Contents

The SDK is located in the directory you specified during installation. It is divided into several sub-directories, as described below.

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

Related Documentation

The following table describes other documentation for the printer.

Item	Description
ROM-DOS User Manual	Information about the printer's ROM-DOS operating system.
Equipment Manual	Information about 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.
Packet Reference Manual	Syntax descriptions of the MPCL printer language.
Datalight Sockets Developer's Guide	Describes the TCP/IP stack and API for use with radio cards.

PRINTER FEATURES

2

There are several printer features that you must understand to write applications. For example, just knowing that 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
- Fonts
- Scanners
- Boot Process
- Windows 95/Network Notes

Display

The printer has an 8-row display. The printers 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.

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

Display Pages

The printer has two display pages (numbered 0-1).

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").

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.

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.

LCD Utility

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.

LCD [*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.

Keypad

The following table describes the keypad's data entry modes. These are described in more detail in the Equipment Manual.

Mode	Acceptable Keys
Numeric/Symbol	**Default.** Numbers and Symbols
Alphabetic	Letters
Special Key	Function Keys

Normally, when entering data, the operator must enter and exit the modes manually. However, with the Kbd functions described in Chapter 4, an application can also change between the modes automatically.

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

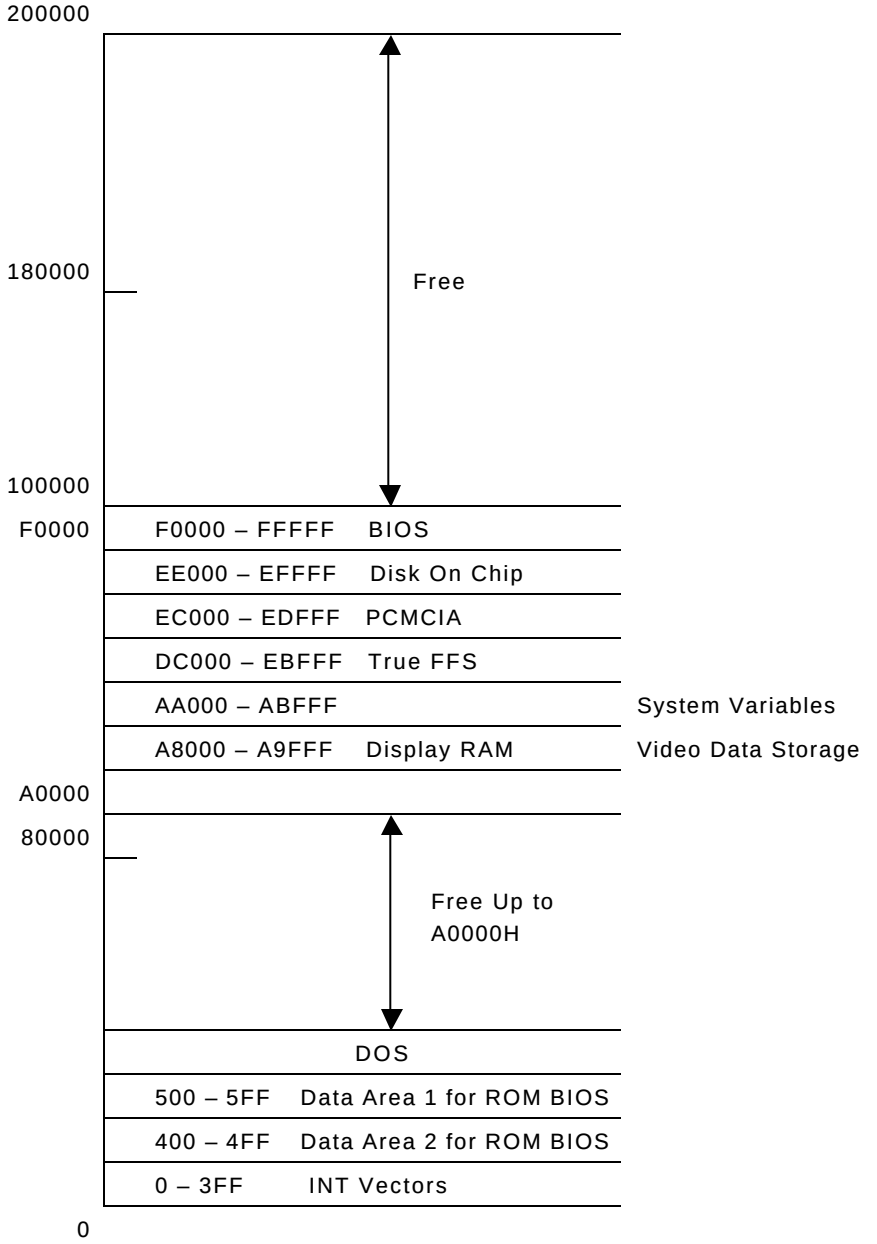
Speaker

Applications can make the printer speaker beep for different lengths of time and frequencies. For example, you might 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 2 MB of memory.

Following is the address map for the printer.



Fonts

The printer has many resident fonts. You must load other fonts separately. Following is a list of these fonts and their IDs.

- Standard (1)
- Reduced (2)
- Bold (3)
- OCRA (4)
- HR1 (5)
- HR2 (6)
- CG Triumvirate Bold® 9 pt. (10)
- CG Triumvirate 6 pt. (11)
- Swiss Bold (50)
- CG Triumvirate Bold (Full Character Set) 6.5 pt. (1000)
- CG Triumvirate Bold (Full Character Set) 8 pt. (1001)
- CG Triumvirate Bold (Full Character Set) 10 pt. (1002)
- CG Triumvirate Bold (Full Character Set) 12 pt. (1003)
- CG Triumvirate Bold (Partial Character Set) 18 pt. (1004)
- CG Triumvirate Bold (Partial Character Set) 22 pt. (1005)
- CG Triumvirate Bold Condensed (Full Character Set) 6.5 pt. (1006)
- CG Triumvirate Bold Condensed (Full Character Set) 8 pt. (1007)
- CG Triumvirate Bold Condensed (Full Character Set) 10 pt. (1008)

- CG Triumvirate Bold Condensed (Full Character Set) 12 pt. (1009)
- CG Triumvirate Bold Condensed (Partial Character Set) 18 pt. (1010)
- CG Triumvirate Bold Condensed (Partial Character Set) 22 pt. (1011)
- Letter Gothic Bold (Full Character Set) 6 pt. (1012)
- Letter Gothic Bold (Full Character Set) 9 pt. (1013)

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.

Using Non-Resident Fonts

To use a non-resident font or a font you have created with the MPCL Toolbox Font Utility, the application must

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

Scanners

Each printer comes with either of two bar code scanners.

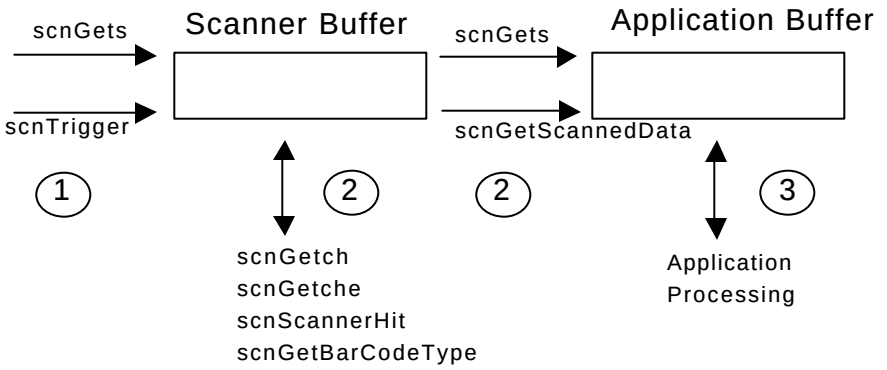
- SE-923
- SE-923HS

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.

NOTE: Every bar code that the SE-923 scanner is configured for delays boot-up by .7 seconds. This does not apply to the SE-923HS scanner.

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 startup screen appears. By default, the display is disabled (nothing other than the startup screen appears). If the LCD command is given, it becomes enabled later in this process.
2. A line moves left to right across the display's bottom to indicate the boot process is working.
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 startup screen never disappears and the printer waits indefinitely for a response to the date prompt.

4. The display is enabled 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).
5. The startup 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

- press 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 pressing Fct 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 a PC

To boot the printer from the PC:

1. Gather the following files on a floppy disk:

   - COMMAND.COM

   - REMDISK.EXE

   - LCD.COM

   - AUTOEXEC.BAT (at a minimum, this file must contain one line containing `LCD Y C`, which enables the display).

2. Enter REMSERV A: at the PC's DOS prompt, where X is the drive where the boot files reside.

3. Turn on the printer and from the Startup Menu, choose #3. The printer boots from the PC.

Startup Menu

To display the Startup Menu:

1. Press the Fct and On/Off keys at the same time.
2. When the logo screen appears, release the Fct key.
3. When the menu appears, press the number of your choice, and then Enter **quickly**. If you don't do it quick enough, the menu scrolls off the screen.

```
6037 Startup Menu
1. Normal Boot
2. Ultra Server
3. Remote Server
4. System Options
```

Normal Boot	Continues the boot process.
Ultra Server	Starts UServe for loading an application (see"Using UServe and UClient").
Remote Server	Runs REMDISK to enable you to boot the printer from the PC (See "Booting from a PC.")
System Options	Displays the following submenu.

```
6037 Startup Menu
1. Diagnostics
2. Print Passthru
3. CMOS Setup
4. F/W Update
5. DOS Prompt
```

Diagnostics	Runs the Diagnostics program. See the Application Notes for more information.
Print Passthru	Starts the Online Passthrough Utility (see "Developing Applications" for more information).
CMOS Setup	Enables a choice of power management modes.
F/W Update	Flashes the firmware in the printer.
DOS Prompt	Bypasses the AUTOEXEC.BAT and CONFIG.SYS files during bootup.

Windows 95/Network Notes

Directory Names

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

Copying Data from the Printer to PC

1. Connect the cable between the PC and the printer.
2. Use UServe and UClient to 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.

DEVELOPING APPLICATIONS 3

Developing applications is a long, detailed process. You cannot just write the application code. You must also understand the whole picture. For example, you must create MPCL packets and assess the application's memory needs.

This chapter describes this process. Following is a summary:

1. Create any needed MPCL packets (if your application prints labels).
2. Write the application.
3. Build (compile and link) the application.
4. Use UServe and UClient to copy files to the printer.
5. Test the application.
6. Train the end users.

Creating MPCL Packets

An application prints labels by submitting MPCL packets to the printer. Refer to the MPCL Packet Reference Manual for more information.

To test your packets without writing an application, use the Online Passthrough Utility:

1. Press the Fct and On/Off keys at the same time.
2. When the logo screen appears, release the Fct key.
3. When the Main menu appears, press 4, then Enter **quickly**. If you don't do it quick enough, the menu scrolls off the screen.
4. When the Startup menu appears, press 2, then Enter. "Online Passthrough running..." appears on the printer display.
5. Connect the PC and printer with a cable and send formats to the printer through the COM port.

To exit Online Passthrough, reboot the printer.

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.

If you plan to include the diagnostics program in the disk image, be sure to write the application so that the end users cannot accidentally get into the program.

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

You must always use MMSULTRA.H.

If you are using Datalight Sockets, include CAPI.H and COMPILER.H.

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

Use the following libraries. Also use LMSDLSKT.LIB with Datalight Sockets.

Name	Printer Support	Scanner Support
LMS6037.LIB	All printing.	Any scanning except where you configure the scanner for a particular bar code.
LMSSCNEN.LIB	None	Scanning where you configure the scanner for a particular bar code.

Using UServe and UClient

To install the application on the printer, you must use UServe and UClient to copy the files to the printer. See Chapter 7 for more information.

Test the Application

It is good practice to test your application when you have loaded it into the printer.

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 procedures (loading supplies, for example) that may vary from the generic descriptions in the Equipment Manual. You must give the Operators and/or System Administrators application-specific instructions, if applicable.

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
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.

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

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

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.

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* or *–1* to prompt the user.

usStockWidth — The supply width in hundredths of an inch. Values are *120*, *150*, *200* or *–1* to prompt the user.

usStockType — The supply type. Values are

MMS_LOW_ENERGY	Paper
MMS_MEDIUM_ENERGY	Fax
MMS_HIGH_ENERGY	Synthetic
-1	Prompt User

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

lpfnSupplyTypePrompt — Enter *0*.

lpfnSupplyPrompt — Enter *0*.

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 MPCL *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 the printer */
  usStatus = pclInit(NULL);
  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);
    }

    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.

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 MPCL *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.

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 "pclFeed" 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 MPCL *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

  sStatus = pclInit(NULL);              // Start Print subsystem
  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


  sStatus = pclInit(NULL);              // Start Print subsystem


  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 MPCL *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

  sStatus = pclInit(NULL);              // Start Print subsystem
  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

  sStatus = pclInit(NULL);          // Start Print subsystem
  if (sStatus != 0)
    printf("Init Failed\nError: %d", sStatus);
  else
  {
    if (!pclBatteryOkToPrint())         // Check battery
      printf("Low battery error");
    else
    {                                   // Open file
      usStatus = pclOpen("B:\\MPCL\\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. This sensor is an option for the printer.

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 MPCL *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

  sStatus = pclInit(NULL);          // Start Print subsystem

  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 MPCL *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

  sStatus = pclInit(NULL);                // Start Print subsystem
  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. An application must call this function before calling any other pcl functions.

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

Syntax

short far pclInit(void);

Parameters

None

Return Values

0 Successful.

Example

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

void main(void)
{
  short sStatus = 0;                    // Status of comm. calls

                                        // Start Print subsystem
  sStatus = pclInit(NULL);
```

```
  if (sStatus == 0)
  {
    /* Branch to printing routine */
    else if (sStatus >= 703 && sStatus <= 793)
     {
       printf("Motion Control Error");
       else
        {
          printf("Error-- press any key when reset.");
          _getch();
          pclClearError();
        }
     }
  }
  pclClose();                          // Close Print subsystem
}
```

pclOpen

Description

Loads MPCL 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 MPCL 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 MPCL *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 to the Print subsystem about the supplies being used.

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 MPCL *Packet Reference Manual* for more information.
Other non-zero	An error occurred. Refer to the MPCL *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.

  status = pclInit(NULL);         // Start Print subsystem
  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 MPCL *Packet Reference Manual* for more information.

Example

See "pclGetOnDemandSensor" for an example.

pclWrite

Description

Writes MPCL packets to the Print subsystem.

You can send no more than one packet at a time. It must be complete.

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 MPCL *Packet Reference Manual* for more information.

Example

See “pclGetOnDemandSensor” for an example.

scnCloseScanner

Description

Disables the 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

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 each scanner.

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 each scanner.

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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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

LMSSCEN.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 each scanner.

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 invalid with LMSSCEN.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 each scanner.

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

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

LMSSCEN.LIB only. Retrieves a pointer to the 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 the 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 the 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 each scanner.

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

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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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

LMSSCEN.LIB only. Saves the 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 the 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 invalid with LMSSCEN.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

LMSSCNEN.LIB only. Saves the 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 on the keypad turns green, the scan was successful. This function works with each 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 printers, values are *0-3*. For 8-row printers, values are *0-7*.
usCol	Column. Values are *0-19*.
usPage	Display page. For 4-row printers, values are *0-3*. For 8-row 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 appear 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 printers, values are *0-3*. For 8-row 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 printers, values are *0-3*. For 8-row 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 printers, values are *0-3*. For 8-row printers, values are *0-7*.
lpusCol	A variable pointer to the column. Values are *0-19*.
sPage	The display page. For 4-row printers, values are *0-3*. For 8-row 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 printers, values are *0-3*. For 8-row 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 printers, values are *0-3*. For 8-row 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 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 appear immediately.

Syntax

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

Parameters

sPage	Display page. For 4-line printers, values are *0-3*. For 8-line 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 appear 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 printers, values are *0-3*. For 8-row 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 appear 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 printers, values are *0-3*. For 8-row 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	Scanner configuration for Codabar bar codes.
CODE128INFO	Scanner configuration for Code 128 bar codes.
CODE39INFO	Scanner configuration for Code 39 bar codes.
CODE93INFO	Scanner configuration for Code 93 bar codes.
D2OF5INFO	Scanner configuration for D 2 of 5 bar codes.
GENERALINFO	General scanner configuration.
I2OF5INFO	Scanner configuration for I 2 of 5 bar codes.
MSIINFO	Scanner configuration for MSI bar codes.
UPCEANINFO	Scanner Configuration for UPC/EAN bar codes.

The data structure names are case-sensitive.

CODABARINFO

The CODABARINFO data structure contains 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

The CODE128INFO data structure contains 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

The CODE39INFO data structure contains 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

The CODE93INFO data structure contains 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

The D2OF5INFO data structure contains 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

The GENERALINFO data structure contains general 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

The I2OF5INFO data structure contains 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

The MSIINFO data structure contains 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*

UPCEANINFO

The UPCEANINFO data structure contains 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 Pulls
- Audio/Visual Feedback

Printing Labels

An application prints labels by submitting MPCL 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 MPCL packets, refer to the *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 print multiple labels in a strip or in a loop. When setting up a method to print multiple labels, you must specify one of the following

- 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 *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 *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 to read bar codes as follows:

1. Enable the scanner with scnScannerOpen or scnOpenScannerShared.
2. If necessary, configure the scanner using the scnGet*xxx* and scnSet*xxx* 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 Pulls

Pressing any key (including pulling 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 pull, the application reads two characters. If the first character is a zero and the second is 0x85, the operator pulled the trigger.

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

Audio/Visual Feedback

It is recommended that in the application, you include some sort of audio or visual feedback indicating that the trigger has been pulled.

For example, after a trigger pull, the application may have the printer beep. Without such feedback, most users keep pulling, not realizing the pull has registered.

USING USERVE AND UCLIENT 7

To prepare the printer to run an application, you must copy the application's files to it from the PC. To do so:

1. Establish a connection between the printer and PC.
2. Copy files between the units.

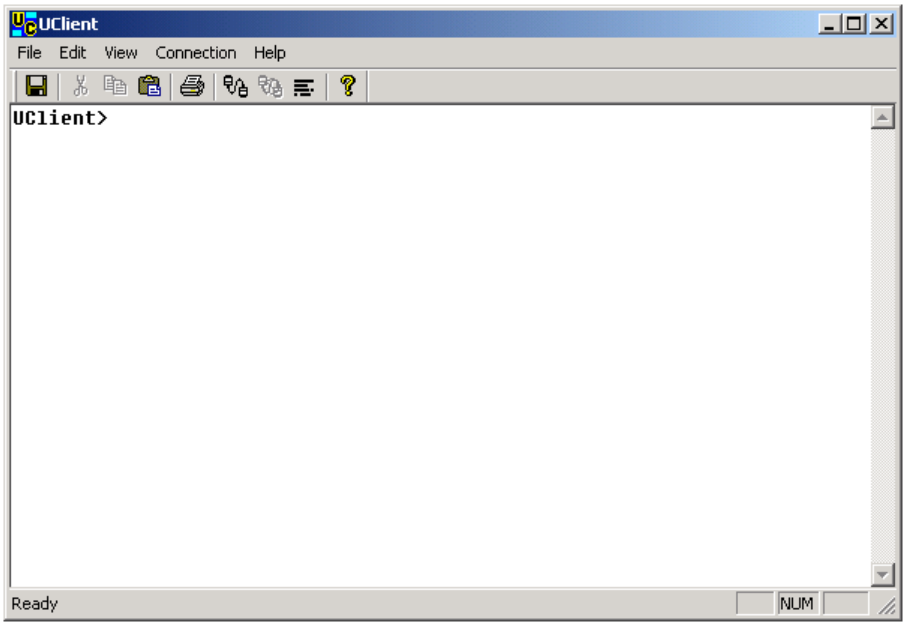
Establishing a PC/Printer Connection

You must establish a connection between the PC and the printer using the UClient and UServe software that runs on the PC and printer, respectively. To establish this connection:

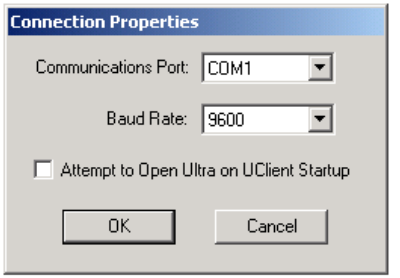
1. Connect the printer and PC with a cable (part number 124054).
2. Turn on the printer and type **USERVE** at the DOS prompt. You will see:

```
USERVE 1.04 READY
_
```

3. Start UClient on the PC. You will see:



4. Select *Properties* from the Connection Menu. You will see:



You set the communication defaults on this screen.

5. Select the PC communications port you are using and the baud rate for the printer and PC to use. Then, click OK.

6. Select *Open* from the Connection menu to open the connection between the two units.

A Note About Connections

For a connection to be open, both sides (printer and PC) have to be ready. If only one side, or neither side, is ready, the connection is closed.

For the printer to be ready, UServe must be running (step 2, in "Establishing a PC/Printer Connection"). For the PC to be ready, UClient must be running, the communication parameters must be set, and Open must have been selected from the Connection (steps 3-6, in "Establishing a PC/Printer Connection").

If one side stops being ready, the other one stays ready. To re-open the connection, you only need to make the one side ready again. The connection between the units immediately opens.

Copying Files

To copy files between the units, you enter commands at the UClient prompt. Note that you copy files *with the PC only* (the one exception is the urun command—see "urun".).

1. Leaving it powered on, the PC/printer connection intact, and UServe running, set aside the printer.

2. Type `ucd d:` at the UClient prompt to change the printer to the d: drive.

NOTE: **Never** copy application files to the c: drive on the printer. Always use the d: drive. This minimizes the possibility of accidentally corrupting system files.

3. Enter commands, one at a time, at the UClient prompt to copy files to the printer. See "Example File Copy" to get an idea of how a file copy might work. UClient Commands" explains these commands in detail.

4. Select *Close* from UClient's Connection menu to sever the PC/printer connection.

5. Select *Exit* from UClient's File menu to exit UClient.

6. Type **X** on the printer's keypad to exit UServe.

Example File Copy

Suppose that the application consists of a number of files in a directory called "App" on the PC:

- AUTOEXEC.BAT
- CONFIG.SYS
- Several .BAT files
- The application's .EXE file.
- FORMAT.FMT
- GR1.GPH
- Several .GPH files
- Several .TXT files

All of them should go in the root D:\ directory.

Following is the sequence of commands you would use to copy these files and create this directory structure on the printer.

NOTE: This command sequence assumes it starts in the c:\ root directory on both the printer and PC.

1. ucd d: — Change the printer's directory from c:\ to d:\.
2. lcd app — Change the PC's directory to c:\app.
3. put *.* — Copy all of the files from c:\app (on the PC) to d:\ on the printer.

UClient Commands

Following are the UClient commands.

Command	Description
close	Closes the communication connection between the PC and printer by taking the PC out of a ready state.
get	Copies a file from the printer to the PC.
help	Displays a list of available commands or help information about a specific command.
lcd	Changes the directory on the PC.
ldel	Deletes a file on the PC.
ldir	Lists details about files and/or directories on the PC, as specified.
lmd	Creates a directory on the PC.
lrd	Deletes a directory on the PC.
lwd	Displays the path to the current directory on the PC.
open	Puts the PC in a communications ready state.
ping	Confirms that the open PC/printer connection is valid.
put	Copies a file from the PC to the printer.
putbios	Copies a BIOS binary file from the PC to the printer.
puttrueffs	Copies a TrueFFS binary file from the PC to the printer.
quit	Exits UClient.
reboot	Restarts the printer.

Command	Description
setbaud	Sets the baud rate for communications between the printer and the PC (overriding the default set from UClient's Connection menu.).
setport	Specifies the port to use on the PC for communication between it and the printer (overriding the default set from UClient's Connection menu).
settimeout	Sets the length of time for the PC to wait for the printer to respond to a command.
ucd	Changes the directory on the printer.
udel	Deletes a file on the printer.
udir	Lists details about files and/or directories on the printer, as specified.
udiskfree	Displays the size of a printer drive and the amount of available space it has.
umd	Creates a directory on the printer.
urd	Deletes a directory on the printer.
urun	Runs a DOS command or program on the printer.
uview	Displays a file's contents on the printer.
uwd	Displays the path to the current directory on the printer.

Syntax Conventions

The syntax in this section uses the following conventions:

Bold Syntax items in bold are the command name.

Italics Items in italics are names of variables.

Italics Items in bold italics are values that a variable can take.

[ ] Items in brackets are optional.

close

Description

Closes the connection between the PC and printer by taking the PC out of a ready state. UServe stays running and the printer remains ready for communications.

Syntax

close

Parameters

None

Example

close

closes the connection between the printer and the PC displays "Connection Closed to Ultra" to confirm it.

get

Description

Copies a file from the printer to the PC.

NOTE: If a file by the same name already exists, it will overwrite the file without warning.

Syntax

get *printerfile*, [*pcfile*]

Parameters

printerfile	The name and path of the source file.
[*pcfile*]	The name and path of the destination file. If you do not specify this parameter, it gets the same name and path on the PC.

Example

get autoexec.bat

copies autoexec.bat from the printer to the PC, giving the file the same name on the PC. The PC displays something similar to the following:

```
Getting File: autoexec.bat [Unknown Size]
  Bytes Received: 537
  Complete
```

help

Description

Displays a list of available commands or help information about a specific command.

Syntax

help [*command*]

Parameters

[*command*] The name of a command.

Example

help lrd

displays the following on the PC:

```
  Command:      LRD
  Description:  local remove directory
  Parameters:   <local directory>
```

lcd

Description

Changes the directory on the PC.

Syntax

lcd *directory*

Parameters

directory The directory to change to. This value can be:

- a path to a particular directory.
- The name of a subdirectory.
- .. (indicating one directory higher than the current one)

Example

lcd c:\backups

changes the current directory to c:\backups and displays this path to confirm.

ldel

Description

Deletes a file on the PC.

Syntax

ldel *filename*

Parameters

filename The name (and optionally, the path) of a file.

Example

ldel autoexec.bat

deletes autoexec.bat on the PC and displays Deleted File: autoexec.bat to confirm it.

ldir

Description

Lists details about files and/or directories on the PC, as specified. Details include date and time of creation (or last update) and size.

Syntax

ldir [*filedirspec*]

Parameters

[*filedirspec*] The name (and optionally, the path) of a file or directory. The default is the current directory.

Example

```
ldir projects
```

displays the contents of the projects directory on the PC, such as the following:

```
.                    03/14/2003      15:18:13      <DIR>
..                   03/14/2003      15:18:13      <DIR>
9850                 01/07/2003      17:34:22      <DIR>
adkpm.pdf            03/14/2003      15:18:13      1931471
adkpmcv.pdf          03/14/2003      14:59:06      1007901
app-a.doc            03/14/2003      11:04:34        33792
app-a.pdf            03/14/2003      11:54:12        85839
cover                03/14/2003      15:08:55      <DIR>
```

lmd

Description

Creates a directory on the PC.

Syntax

lmd *directory*

Parameters

directory The name (and optionally, the path) of a new directory. If the directory already exists, an error occurs.

Example

lmd tc6035pm

creates a directory named tc6035pm on the PC and displays Created directory: tc6035pm to confirm it.

lrd

Description

Deletes a directory on the PC.

Syntax

lrd *directory*

Parameters

directory The name (and optionally, the path) of a directory. If the directory does not exist, an error occurs.

Example

lrd tc6035pm

deletes the tc6035pm directory on the PC and displays Removed directory: tc6035pm to confirm it.

lwd

Description

Displays the path to the current directory on the PC.

Syntax

lwd

Parameters

None

Example

lwd

displays the path to the current directory on the PC. For example, c:\projects\adk.

open

Description

Puts the PC in a communications ready state. If the printer is ready (UServe is running), the connection between the printer and the PC opens.

Syntax

open [*commport*][*, baudrate*]

Parameters

[*commport*]	The communications port on the PC that you have attached the cable to.
[*baudrate*]	The baud rate for communications.

Example

open com1, 9600

opens the com1 port on the PC for communications at 9600 baud.

ping

Description

Confirms that the open PC/printer connection is valid.

Syntax

ping

Parameters

None

Example

ping

confirms that the PC/printer communications are ready. If it is a positive confirmation, the UClient prompt reappears. Otherwise, an error message appears.

put

Description

Copies a file from the PC to the printer. A PC/printer connection must be in place before using this command.

NOTE: If a file by the same name already exists, it will overwrite the file without warning.

Syntax

put *pcfile* [*, printerfile*]

Parameters

pcfile	The name (and optionally, the path) of the file to copy. You can use the * wild card to copy multiple files (i.e., *.txt to copy all .txt files).
[*printerfile*]	The name (and optionally, the path) of the file when placed in the printer. If you do not specify this parameter, the file name is assumed to be the same and the directory to be the current one.

Example

put lookup.txt

copies lookup.txt from the PC to the printer, giving it the same name on the printer. It also displays a listing of the file name and its size, and a count of the bytes sent so far as the copy occurs. When the copy finishes, "Complete" appears. For example:

```
Sending File: lookup.txt [2000 bytes]
  Bytes Sent: 2000
  Complete
```

putbios

Description

Copies a BIOS binary file from the PC to the printer.

Syntax

putbios *filename*

Parameters

filename The name of the BIOS file.

Example

putbios bios.dat

copies bios.dat from the PC to the printer, displaying the file name and status of the command as it does so.

puttrueffs

Description

Copies a TrueFFS binary file from the PC to the printer.

Syntax

puttrueffs *filename*

Parameters

filename The name of the file.

Example

puttrueffs flash.bin

copies flash.bin from the PC to the printer, displaying the file name and status of the command as it does so.

quit

Description

Exits UClient.

Syntax

quit

Parameters

None

Example

quit

Exits the program and returns PC control to Windows.

reboot

Description

Restarts the printer. UServe stops running as a result.

Syntax

reboot

Parameters

None

Example

reboot

restarts the printer. If successful, the UClient prompt reappears.

setbaud

Description

Sets the baud rate for communications between the printer and the PC (overriding the default set from UClient's Connection menu.).

Syntax

setbaud *rate*

Parameters

rate The baud rate. Values are: ***1200***, ***2400***, ***4800***, ***9600***, ***19200***, ***38400***, ***57600***, and ***115200***.

Example

setbaud 9600

sets the baud rate to 9600.

setport

Description

Specifies the port to use on the PC for communication between it and the printer (overriding the default set from UClient's Connection menu.).

Syntax

setport *port*

Parameters

port The communication port on the PC that you have attached the cable to. Values are: ***com1*** – ***com8***.

Example

setport com1

specifies to use the PC's com1 communication port.

settimeout

Description

Sets the length of time for the PC to wait for the printer to respond to a command. The connection between the printer and PC does not have to be active when entering this command.

Syntax

settimeout *length*

Parameters

length The length of the timeout (in seconds).

Example

settimeout 10

sets the PC timeout to 10 seconds. If the command is successful, the UClient prompt reappears.

ucd

Description

Changes the directory on the printer.

Syntax

ucd *directory*

Parameters

directory The name or path of the new directory. Move up one directory (the **..** specification) does not work with this command.

Example

ucd data

changes the printer to the data directory, displaying the UClient prompt when complete.

udel

Description

Deletes a file on the printer. Wild cards are not valid with this command.

Syntax

udel *filename*

Parameters

filename The name of the file to delete.

Example

udel autoexec.bat

deletes autoexec.bat and displays Deleted File: autoexec.bat, confirming the deletion.

udir

Description

Lists details about files and/or directories on the printer, as specified. Details include date and time of creation (or last update) and size.

Syntax

udir [*filedirspec*]

Parameters

[*filedirspec*] A file or directory name (and optionally, the path). The default is the current directory.

Example

udir c:

displays a list of files in the c: root directory. Below is an example.

```
.               01/02/1980      00:17:18        <DIR>
..              01/02/1980      00:17:18        <DIR>
IBMBIO.COM      04/03/2003      10:23:16        63645
IBMDOS.COM      04/03/2003      10:23:16           77
COMMAND.COM     04/01/2003      07:10:00        45742
DOS             01/02/1980      00:17:22        <DIR>
AUTOEXEC.BAT    03/28/2003      08:52:44          537
APP.EXE         04/04/2003      10:14:56        87219
DEBUG.EXE       05/31/1994      06:22:00        15718
DIAG.EXE        04/03/2003      15:16:36        98782
U.EXE           01/04/2003      08:17:56        65367
TP.EXE          04/03/2003      15:12:16         6859
UTIL            02/17/1980      22:22:58        <DIR>
CONFIG.SYS      03/28/2003      08:46:50          602
```

udiskfree

Description

Displays the size of a printer drive and the amount of available space it has.

Syntax

udiskfree *disk*

Parameters

disk The drive to query for space and size. ***c*** and ***d*** are the possible values. A colon (:) is unnecessary after the letter.

Example

udiskfree c

displays something like the following:

```
Drive = C
  Total Size = 1462272 bytes
  Available = 659456 bytes
```

umd

Description

Creates a directory on the printer. If the directory already exists, an error occurs.

Syntax

umd *directory*

Parameters

directory The name (and optionally, the path) of the new directory.

Example

umd backups

creates a subdirectory (to the current directory) called backups on the printer.

urd

Description

Deletes a directory on the printer. If the directory does not exist, an error occurs.

Syntax

urd *directory*

Parameters

directory A directory name (and optionally, the path).

Example

urd backups

deletes the backups subdirectory from the current directory.

urun

Description

Runs a DOS command or program on the printer. UServe runs beneath this offshoot task. When the command or program ends, UServe is still running.

Running this command is the one case of where you would use the printer, rather than only using UClient.

If you run a program (such as Diagnostics) on the printer, Command Failed. Did not receive a response from the Ultra appears on the UClient screen. This error is nothing to worry about. It should not occur, though, when running DOS commands with urun.

Syntax

urun *command*

Parameters

command The command or program name, along with any necessary parameters.

Example

urun diag

runs the printer's diagnostics program as an offshoot task from UServe.

uview

Description

Displays a file's contents on the printer.

Syntax

uview *file*

Parameters

file The name (and optionally, the path) of the file to display.

Example

uview test.txt

displays the contents of test.txt. Following is an example:

test.txt:

This is a test.

Uwd

Description

Displays the path to the current directory on the printer.

Syntax

uwd

Parameters

None

Example

```
uwd
```

displays the path to the current directory on the printer. For example,

```
D:\DATA
```

Troubleshooting

Consult the following table if any errors occur while you are using UClient and UServe. The messages can appear individually or together. They are listed alphabetically.

Message	Solution
Cannot find the requested file.	Verify that the specified file exists and you spelled its name correctly. Then, retry the command.
Command Failed.	Verify that a valid PC/printer connection exists. If so, then verify the syntax and parameters of the command and re-try it.
Connection Closed to Ultra	You tried to close a PC/printer connection that already is closed.
Connection to the Ultra has not been opened	Establish a PC/printer connection. See "Establishing a PC/Printer Connection."
Could not create directory:	There is not enough disk space to create a directory. Free some space before trying again.
Could not delete file:	You specified a non-existent file with the command. Verify the spelling of the name and try again.
Could not establish a connection with the Ultra.	Reboot the printer and re-establish the PC/printer connection. If this error still occurs, have the printer serviced.

Message	Solution
Could not get the file.	Verify that a valid PC/printer connection exists. If so, verify the syntax of the command and retry it.
Could not put the file.	Verify that a valid PC/printer connection exists. If so, verify the syntax of the command. Then, retry the command.
Could not remove directory:	Verify that the directory you specified exists and retry the command.
Could not save file locally.	There is not enough space on the PC to save a file received from the printer. Free some space and retry the command.
Did not receive a response from the Ultra.	Verify that a valid PC/printer connection exists. If so, increase the length of the timeout. Then retry the command. From the user perspective, a command may finish running, but there can still be processing going on behind the scenes before the command is truly finished. A timeout can occur during this timeframe. Therefore, when a timeout error occurs, check to see if the command actually finished or not before trying to correct the problem. For example, if the error occurred on a put, verify that the file copy completed.
Error Running command on Ultra. Returned Error = xxx	A ROM-DOS error occurred on the printer. Retry the command. If the same error occurs, re-boot the printer and/or have it serviced.

Message	Solution
Invalid Parameters for command	Verify the syntax of the command and retry it with all of the parameters.
No Connection is present with Ultra.	Establish a PC/printer connection.
No directory was specified.	Verify the syntax of the command and enter a directory name where specified.
No File Specified.	A file name parameter was left blank. Retry the command, listing a file name.
No Files could be found.	The file you specified in the command does not exist. Verify the correct name and/or spelling and retry the command.
No matching files found.	You specified file names with a wild card, and there are none fitting the specification. Verify the file names and spelling and retry the command.
Open Failed.	UServe is not running on the printer. Start it and establish a PC/printer connection.
Response could not be read from Ultra	Verify that a valid PC/printer connection exists and retry the command.
Request could not be sent to the Ultra.	Verify that a valid PC/printer connection exists and retry the command.
Soft Reboot Failed.	Verify that a valid PC/printer connection exists and retry the command.
Sum check received from the Ultra was invalid.	Verify that a valid PC/printer connection exists and retry the command.

Message	Solution
Supplied Buffer was too small for amount of data.	Call Technical Support at the phone number on the back of this manual.
Ultra returned Negative response.	The printer timed out before it could complete the command. Increase the length of the timeout with the settimeout command and retry the command that timed out.
Unknown Command	Verify the spelling of the command name and re-enter it.
Unknown Drive Letter	Retry the udiskfree command, but specify either ***c*** or ***d*** as the drive letter.
Unknown Error (Error = xxx).	A UServe error occurred. Physically reboot the printer (not with the reboot command), start UServe, and try the command again. If the error still occurs, have the printer serviced.

SAMPLE APPLICATIONS

A

The SDK contains several sample applications for both printers 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.
7	Scans specific bar codes and prints.

Sample 1

Function

Prints with the trigger.

Algorithm

1. Initialize the Print subsystem without allocating font storage memory.
2. Loads a format packet.
3. Performs the following until the operator presses the trigger (to print), Esc (to break out of the loop), or Fct +0 to calibrate supplies.
   - **A.** Checks whether the battery is charged enough for printing.
   - **B.** Loads a batch packet and prints the format.
   - **C.** Waits until the label prints by checking the status repeatedly until it returns something other than "busy."
4. Closes the Print subsystem.

Sample 2

Function

Prints using the on-demand sensor.

Algorithm

1. Initializes the Print subsystem without allocating font storage memory.
2. Loads a format packet. This format defines the feed mode as on-demand.
3. Checks whether the battery is charged enough for printing.
4. Loads a batch packet and prints the format.

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

6. Closes the Print subsystem.

Sample 3

Function

Prints a strip of labels.

Algorithm

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

2. Loads a format packet.

3. Performs the following until an error occurs

   A. Checks whether the battery is charged enough for printing.

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

   C. Builds the batch and prints the label(s).

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

3. Closes the Print subsystem.

Sample 4

Function

Prints using fonts and formats loaded with pclOpen.

Algorithm

1. Initializes the Print subsystem.
2. Loads FORMATS.PCL. This file contains an MPCL format packet.
3. Performs the following until the operator presses the trigger (to print), Esc (to break out of the loop), or Fct +0 to calibrate supplies.
   - **A.** Waits for the operator to press the trigger.
   - **B.** Checks whether the battery is charged enough for printing.
   - **C.** Loads a batch packet.
   - **D.** Waits until the label prints by checking the status repeatedly until it returns something other than "busy."
4. Closes the Print subsystem.

Sample 5

Function

Uses the scanner.

Algorithm

1. Enables the scanner.
2. Waits for the operator to press the trigger. Pressing Esc breaks out of the waiting loop (without performing a scan).
3. Initiates a scan.
4. Disables the scanner.

Sample 6

Function

Scans and prints.

Algorithm

1. Initializes the Print subsystem without allocating font storage memory.
2. Loads a format.
3. Enables the scanner.
4. Performs the following until the operator presses the trigger, Esc (to break out of the loop), or Fct +0 to calibrate supplies.
   - **A.** Initiates a scan.
   - **B.** Checks whether the battery is charged enough for printing.
   - **C.** Loads the batch, prompting for a quantity.
   - **D.** Waits until the label prints by checking the status repeatedly until it returns something other than “busy.”
5. Disables the scanner.
6. Closes the Print subsystem.

Sample 7

Function

Scans and prints.

Algorithm

1. Initializes the Print subsystem without allocating font storage memory.
2. Loads a format.
3. Enables the scanner.

4. Sets the scanner to transmit the System characters and country code with UPCA bar codes.

5. Performs the following until the operator to presses the trigger, Esc (to break out of the loop), or Fct +0 to calibrate supplies.

   A. Initiates a scan.

   B. Checks whether the battery is charged enough for printing.

   C. Prompts the operator to enter a quantity.

   D. Builds the batch dynamically with the entered data.

   E. Loads the batch.

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

6. Disable the scanner.

7. Close the Print subsystem.

GLOSSARY

B

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.
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.
Motion Control subsystem	The printer subsystem that controls how paper feeds through the printer.
MPCL	Monarch Printer Control Language. This language contains commands that drive the printer. Refer to the *Packet Reference Manual* for more information.
Packet	A unit of MPCL commands. For example, to print a particular label, the application writes a particular group of MPCL commands to the print subsystem. This group is enclosed in braces and is known as a packet. Refer to the *Packet Reference Manual* for more information.
Print subsystem	The part of the printer that controls printing.
ROM-DOS	The MS-DOS 6.22-compatible operating system that the printer runs. Datalight manufactures it.

Term	Definition
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

1

A

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

W

For supplies, service, or assistance call toll free:

1-800-543-6650 (USA)
1-800-363-7525 (Canada)
44 1279 786777 (UK)
45 14 67 00 (France)
49 5731 78060 (Germany)
34-93 746 43 10 (Spain)
01 800 300 72927 (Mexico)
55 (47) 338 2396 (Brazil)
61 2 9647 1833 (Australia)
852-2328-9949 (Hong Kong)
94-1-46500 (Sri Lanka)

www.paxar.com