

SPT Terminal Series



System Software Manual

SPT Terminal Series System Software Manual

72E-56803-01

Revision A

May 2002



© **2002** by Symbol Technologies, Inc. All rights reserved.

No part of this publication may be reproduced or used in any form, or by any electrical or mechanical means, without permission in writing from Symbol. This includes electronic or mechanical means, such as photocopying, recording, or information storage and retrieval systems. The material in this manual is subject to change without notice.

The software is provided strictly on an “as is” basis. All software, including firmware, furnished to the user is on a licensed basis. Symbol grants to the user a non-transferable and non-exclusive license to use each software or firmware program delivered hereunder (licensed program). Except as noted below, such license may not be assigned, sublicensed, or otherwise transferred by the user without prior written consent of Symbol. No right to copy a licensed program in whole or in part is granted, except as permitted under copyright law. The user shall not modify, merge, or incorporate any form or portion of a licensed program with other program material, create a derivative work from a licensed program, or use a licensed program in a network without written permission from Symbol. The user agrees to maintain Symbol’s copyright notice on the licensed programs delivered hereunder, and to include the same on any authorized copies it makes, in whole or in part. The user agrees not to decompile, disassemble, decode, or reverse engineer any licensed program delivered to the user or any portion thereof.

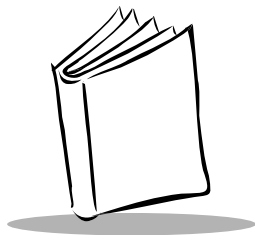
Symbol reserves the right to make changes to any software or product to improve reliability, function, or design.

Symbol does not assume any product liability arising out of, or in connection with, the application or use of any product, circuit, or application described herein.

No license is granted, either expressly or by implication, estoppel, or otherwise under any Symbol Technologies, Inc., intellectual property rights. An implied license only exists for equipment, circuits, and subsystems contained in Symbol products.

Symbol, Spectrum One, and Spectrum24 are registered trademarks of Symbol Technologies, Inc. Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

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



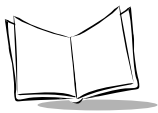
Contents

About This Guide

Chapter Descriptions	vii
Notational Conventions	viii
Related Documents	viii
Service Information	viii
Symbol Support Center	ix

Chapter 1. SPT Scanners

Section Descriptions	1-1
Using the Scan Manager Shared Library	1-2
Using the API	1-2
Using the Scan Demo Application	1-6
Scanner Commands	1-6
Introduction	1-6
Returned Status Definitions	1-7
Scanner Commands	1-8
Barcode Parameter Functions	1-37
Introduction	1-37
Returned Status Definitions	1-37
Barcode Types	1-38
Codabar Barcode Parameter Functions	1-39
Code 32 Barcode Parameter Functions	1-44
Code 39 Barcode Parameter Functions	1-47
General Barcode Parameter Functions	1-52
I 2 of 5 Barcode Parameter Functions	1-63
MSI Plessey Barcode Parameter Functions	1-66
UPC/EAN Barcode Parameter Functions	1-71
Hardware Parameter Functions	1-83
Introduction	1-83
Returned Status Definitions	1-83



Hardware Parameter Functions	1-84
Power Considerations	1-118
scanBatteryErrorEvent	1-118
Sudden Loss of Power	1-118
Backlighting	1-118
Other Power Notes	1-119
Sample Scanning Application	1-119
Writing the Code	1-120

Chapter 2. MSR 3000

Section Descriptions	2-2
MSR 3000 Features	2-2
Library Globals	2-5
Using The MSR Manager Shared Library	2-7
Using the API	2-7
MSR Commands	2-8
Introduction	2-8
Return Codes	2-8
MSR 3000 Command Descriptions	2-9
Application Templates	2-44
MSR 3000 Configurator	2-45
Introduction	2-45
File menu commands	2-45
View menu commands	2-45
Help Menu Commands	2-46
New Command (File Menu)	2-47
Open Command (File Menu)	2-48
Exit Command (File Menu)	2-49
Toolbar Command (View Menu)	2-50
Toolbar	2-51
Status Bar Command (View Menu)	2-52
Status Bar	2-53
Help Topic Command (Help Menu)	2-54
About Command (Help Menu)	2-55
Context Help Command	2-56
Scroll Bars	2-57
Move Command (Control Menu)	2-58
Size Command (Control Menu)	2-59
Minimize Command (Control Menu)	2-60
Maximize Command (Control Menu)	2-61
Close Command (Control Menu)	2-62
Configurator Properties Buttons	2-63
Using the Configurator to Set the MSR 3000	2-63

Introduction	2-63
A Simple Application Program Sample	2-68
Include Files	2-68
PilotMain Routine	2-69
AppStart Function	2-71
MainFormHandleEvent Function	2-73
AppStop Function	2-79

Chapter 3. Printers for Palm Computing Platform

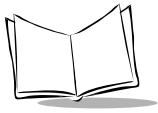
Introduction	3-1
Application Programming Interface (API)	3-2
API Architectural Overview	3-2
Section Descriptions	3-4
System Requirements	3-4
Conventions Used in this Manual	3-4
API Function Calls	3-5
Introduction	3-5
Returned Status Definitions	3-5
Print Commands	3-6
General Purpose Interface Functions	3-7
High-level API Calls	3-18
Lower-level API Calls	3-26
Data Structures	3-29
Sample Application	3-32
Code Samples	3-33

Chapter 4. Spectrum24

Introduction	4-1
ESSID and BSSID	4-1
Wired and Wireless Network Connections	4-2
Spectrum24/NetLib Design and Implementation Considerations	4-5
Spectrum24 Radio API	4-8
The Programmer's Interface	4-8
Spectrum24 Data Structures	4-10
Spectrum24 Constants	4-13
Spectrum24 Network Interface Settings	4-14

Appendix A. ASCII Equivalents

Appendix B. Scan Manager Parameter Definitions



Appendix C. Data Editing Overview for Magnetic Stripe Reader

Introduction	C-1
Functions	C-1
Rearrange the Data	C-1
Insert Character Strings into the Output Data Record	C-1
Duplicate Fields	C-1
Select Output Fields	C-2
Output Method	C-2
Fields	C-2
Formulas	C-3
Added Field	C-3
Search Method	C-3
Operation	C-4

Appendix D. Common Magnetic Card Encoding Formats for MSR

Credit Card Format	D-1
Track1 Format:	D-1
Track2 Format:	D-2
California Driver's License Format (DMV)	D-3
Track1 Format:	D-3
Track2 Format:	D-4
Track3 Format:	D-5
Driver's License Format Recommended by AAMVA	D-6
Track1 Format:	D-6
Track2 Format:	D-8
Track3 Format:	D-10

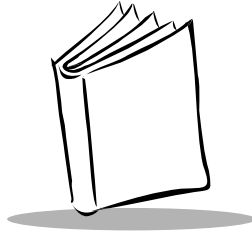
Appendix E. Supported Printers

Variables in Printcap Strings	E-2
Printing Rectangles	E-3
Portable Label Printers	E-4
Commercial Printers	E-10
Using Forms with Legacy Printers	E-12
Writing the form to the printer	E-13

Glossary

Index

Feedback



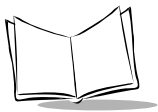
About This Guide

The SPT Terminal Series System Software Manual provides information necessary to develop applications for Symbol SPT terminals.

Chapter Descriptions

Topics covered in this guide are as follows:

- [Chapter 1, SPT Scanners](#), provides information for developers who want to create scan-aware applications for the terminal. The chapter assumes that you are familiar with the CodeWarrior development environment.
- [Chapter 2, MSR 3000](#), provides information for use in developing applications to enable magnetic stripe reading on the Symbol Technologies SPT Terminals.
- [Chapter 3, Printers for Palm Computing Platform](#), is for developers who want to create print applications for the Palm III or Symbol Palm Terminal (SPT). It assumes that you are familiar with the CodeWarrior development environment.
- [Chapter 4, Spectrum24](#), provides an overview of Spectrum24[®] wireless operation, and information for developers about the Spectrum24 programming interface.
- [Appendix A, ASCII Equivalents](#), contains a list of the scan value, hex value, full ASCII code, and keystrokes for each barcode.
- [Appendix B, Scan Manager Parameter Definitions](#), contains a list of the parameters available to developers, and the parameter default values.
- [Appendix C, Data Editing Overview for Magnetic Stripe Reader](#), describes the data editing feature of the API, which allows you to edit the data which has been read from a magnetic card before sending it to the application.
- [Appendix D, Common Magnetic Card Encoding Formats for MSR](#), provides a listing of the commonly used magnetic card formats for use in your application development.



- [Appendix E, Supported Printers](#), contains a list of supported printer names and models.

Notational Conventions

The following conventions are used in this document:

- Italics are used to highlight specific items in the general text, and to identify chapters and sections in this and related documents.
- Bullets (•) indicate:
 - action items
 - lists of alternatives
 - lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Documents

- *SPT 1800 Series Quick Reference Guide*, p/n 72-51336-xx.
- *SPT 1800 Series Product Reference Guide*, p/n 72-51337-xx.
- *Spectrum24 Site Survey Utility User Guide*, p/n 72-39283-xx

Service Information

If you have a problem with your equipment, contact the [Symbol Support Center](#) for your region. See [page ix](#) for contact information. Before calling, have the model number, serial number, and several of your bar code symbols at hand.

Call the Support Center from a phone near the equipment so that the service person can try to talk you through your problem. If the equipment is found to be working properly and the problem is symbol readability, the Support Center will request samples of your bar codes for analysis at our plant.

If your problem cannot be solved over the phone, you may need to return your equipment for servicing. If that is necessary, you will be given specific directions.

Note: *Symbol Technologies is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty. If the original shipping container was not kept, contact Symbol to have another sent to you.*

Symbol Support Center

For service information, warranty information or technical assistance contact or call the Symbol Support Center in:

United States ¹

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
1-800-653-5350

United Kingdom

Symbol Technologies
Symbol Place
Winnersh Triangle, Berkshire RG41 5TP
United Kingdom
0800 328 2424 (Inside UK)
+44 118 945 7529 (Outside UK)

Australia

Symbol Technologies Pty. Ltd.
432 St. Kilda Road
Melbourne, Victoria 3004
1-800-672-906 (Inside Australia)
+61-3-9866-6044 (Outside Australia)

Denmark/Danmark

Symbol Technologies AS
Dr. Neergaardsvej 3
2970 Hørsholm
7020-1718 (Inside Denmark)
+45-7020-1718 (Outside Denmark)

Canada

Symbol Technologies Canada, Inc.
2540 Matheson Boulevard East
Mississauga, Ontario, Canada L4W 4Z2
905-629-7226

Asia/Pacific

Symbol Technologies Asia, Inc.
230 Victoria Street #04-05
Bugis Junction Office Tower
Singapore 188024
337-6588 (Inside Singapore)
+65-337-6588 (Outside Singapore)

Austria/Österreich

Symbol Technologies Austria GmbH
Prinz-Eugen Strasse 70 / 2.Haus
1040 Vienna, Austria
01-5055794-0 (Inside Austria)
+43-1-5055794-0 (Outside Austria)

Europe/Mid-East Distributor Operations

Contact your local distributor or call
+44 118 945 7360

**Finland/Suomi**

Oy Symbol Technologies
Kaupintie 8 A 6
FIN-00440 Helsinki, Finland
9 5407 580 (Inside Finland)
+358 9 5407 580 (Outside Finland)

Germany/Deutschland

Symbol Technologies GmbH
Waldstrasse 66
D-63128 Dietzenbach, Germany
6074-49020 (Inside Germany)
+49-6074-49020 (Outside Germany)

Latin America Sales Support

7900 Glades Road
Suite 340
Boca Raton, Florida 33434 USA
1-800-347-0178 (Inside United States)
+1-561-483-1275 (Outside United States)

Netherlands/Nederland

Symbol Technologies
Kerkplein 2, 7051 CX
Postbus 24 7050 AA
Varsseveld, Netherlands
315-271700 (Inside Netherlands)
+31-315-271700 (Outside Netherlands)

France

Symbol Technologies France
Centre d'Affaire d'Antony
3 Rue de la Renaissance
92184 Antony Cedex, France
01-40-96-52-21 (Inside France)
+33-1-40-96-52-50 (Outside France)

Italy/Italia

Symbol Technologies Italia S.R.L.
Via Cristoforo Columbo, 49
20090 Trezzano S/N Navigilo
Milano, Italy
2-484441 (Inside Italy)
+39-02-484441 (Outside Italy)

Mexico/México

Symbol Technologies Mexico Ltd.
Torre Picasso
Boulevard Manuel Avila Camacho No 88
Lomas de Chapultepec CP 11000
Mexico City, DF, Mexico
5-520-1835 (Inside Mexico)
+52-5-520-1835 (Outside Mexico)

Norway/Norge

Symbol's registered and mailing address:
Symbol Technologies Norway
Hoybratenveien 35 C
N-1055 OSLO, Norway

Symbol's repair depot and shipping address:
Symbol Technologies Norway
Enebakkveien 123
N-0680 OSLO, Norway

+47 2232 4375

South Africa

Symbol Technologies Africa Inc.
Block B2
Rutherford Estate
1 Scott Street
Waverly 2090 Johannesburg
Republic of South Africa
11-809 5311 (Inside South Africa)
+27-11-809 5311 (Outside South Africa)

Sweden/Sverige

"Letter" address:

Symbol Technologies AB
Box 1354
S-171 26 SOLNA
Sweden

Visit/shipping address:

Symbol Technologies AB
Solna Strandväg 78
S-171 54 SOLNA
Sweden

Switchboard: 08 445 29 00 (domestic)

Call Center: +46 8 445 29 29 (international)

Support E-Mail:

Sweden.Support@se.symbol.com

¹Customer support is available 24 hours a day, 7 days a week.

If you purchased your Symbol product from a Symbol Business Partner, contact that Business Partner for service.

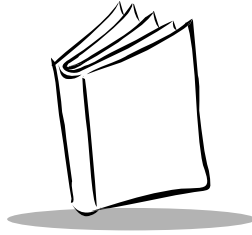
For the latest version of this guide go to:<http://www.symbol.com/manuals>.

Spain/España

Symbol Technologies S.L.
C/ Peonias, 2
Edificio Piovera Azul
28042 Madrid, Spain
91 324 40 00 (Inside Spain)
+34 91 324 40 00 (Outside Spain)



SPT Terminal Series System Software Manual



Chapter 1

SPT Scanners

Section Descriptions

- [*Using the Scan Manager Shared Library*](#)—A high-level overview of the code that creates a typical scanning application, and a description of a simple scanning application that lists the function calls that should be included in a typical scanning application.
- [*Scanner Commands*](#)—A list of the commands that operate the scanner.
- [*Barcode Parameter Functions*](#)—A list of the parameter functions that set the scan parameters associated with specific types of barcodes.
- [*Hardware Parameter Functions*](#)—A list of the parameter functions that set the parameters associated with the scanning hardware.
- [*Power Considerations*](#)—A description of how Scan Manager functions affect the levels of power available to the scanner hardware.
- [*Sample Scanning Application*](#)—A demo application included with the Scan Manager SDK that exercises nearly all of the API.



Using the Scan Manager Shared Library

Using the API

The Scan Manager software development kit (SDK) is used by third-party developers to create scanner-enabled applications for the terminal. The Scan Manager shared library API allow terminal applications to control and receive data from the scanner hardware.

A typical application uses the Scan Manager shared library to do the following, in the order listed below:

1. Open the scanner.
2. Enable the scanner to initiate scans through either the hardware or the application.
3. Handle any decoded data or error messages received from the decoder.
4. Shut down the scanner.

Refer to [Sample Scanning Application](#) for a detailed walk-through of SScan, a sample scanner-enabled application.

The following snippets of code are a simple construct of a typical third-party application:

```
#include "PalmOS.h"           // all the system toolbox headers
#include <Menu.h>
...
...
#include "ScanMgrDef.h"        // Scan Manager constant definitions
#include "ScanMgrStruct.h"     // Scan Manager structure definitions
#include "ScanMgr.h"           // Scan Manager API function definitions
...
#include "SscandemoRsc.h"      // application resource defines
#include "Utils.h"             // miscellaneous utility functions
#define SCANDATA_WIDTH 145
Boolean extend;
Int16 extenedDataLength
```



```
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
```

```
{
    // Check for a normal launch.
    if (cmd == sysAppLaunchCmdNormalLaunch)
    {
        Err error = STATUS_OK;

        // Set up Scan Manager and the initial (Main) form.
        StartApplication();

        // Start up the event loop.
        EventLoop();

        // Close down Scan Manager, decoder
        StopApplication();
    }

    return(0);
}
```

```
/******
```

```
*
```

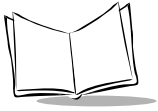
```
* FUNCTION:   StartApplication
```

```
*
```

```
* DESCRIPTION: This routine sets up the initial state of the
```

```
*
```

```
    application.
```



*

*****/

```
static void StartApplication(void)  
{  
    Err error;  
  
    // Call up the main form.  
    FrmGotoForm( MainForm );  
  
    // Now, open the scan manager library  
    error = ScanOpenDecoder();  
  
    // Set decoder parameters we care about...  
    // enable scanning  
    ScanCmdScanEnable();  
    // allow software-triggered scans  
    ScanSetTriggeringModes( HOST );  
    // Enable any barcodes to be scanned  
    ScanSetBarcodeEnabled( barUPCA, true );  
    ScanSetBarcodeEnabled( barUPCE, true );  
    ScanSetBarcodeEnabled( barUPCE1, true );  
    ScanSetBarcodeEnabled( barEAN13, true );  
    ScanSetBarcodeEnabled( barEAN8, true );  
    ScanSetBarcodeEnabled( barBOOKLAND_EAN, true);  
    ScanSetBarcodeEnabled( barCOUPON, true);  
  
    // We've set our parameters...
```

```
// Now call "ScanCmdSendParams" to send them to the decoder
ScanCmdSendParams( No_Beep);
}
```

```

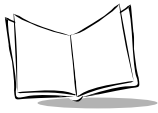
/*****
*
* FUNCTION:   StopApplication
*
* DESCRIPTION: This routine does any cleanup required, including
*                 shutting down the decoder and Scan Manager shared
*                 library.
*
*****/

```

```
static void StopApplication(void)
{
    // Disable the scanner and Close Scan Manager shared library
    ScanCmdScanDisable();
    ScanCloseDecoder();
}
```

To start the scanner:

1. Call the **ScanOpenDecoder()** function to open the Scan Manager shared library, and to initialize the scanner. You *must* call this function first, before any other function in the shared library can be called.
2. Use the appropriate Scan Manager functions to set any of the other scanner parameters, such as barcode formats. The specified parameters are *only set locally*. To send the new parameters to the scanner, you *must* call **ScanCmdSendParams()**. The new parameters remain in effect until you or another application changes them, or **ScanCmdParamDefaults()** is called.
3. Call the **ScanCmdScanEnable()** function to allow scanning to be performed.



To set the scan trigger:

Call the **ScanSetTriggeringModes()** function to identify the type of trigger that will initiate scans. The typical application passes this function to the **LEVEL** parameter.

To handle scanner data and errors:

1. In your event handling code, respond to any **scanDecodeEvent** by storing or displaying the decoded data.
2. Respond to error conditions (such as **scanBatteryErrorEvent**) by alerting the user or performing appropriate recovery routines.

To shut down the scanner:

1. Call the **ScanCmdScanDisable()** function to shut down the scanner.
2. Call the **ScanCloseDecoder()** function at the conclusion of the program. If you don't, you'll get system errors and unexpected results.

Using the Scan Demo Application

Scan Demo is a demo application included with the Scan Manager shared library. Scan Demo exercises nearly all of the API, and shows you how to:

- Use the API to set and get scanner parameters
- Handle decoded scanner data
- Handle scan errors and a low-battery condition

This demo application also allows you to use the terminal's graphical interface to display and change scanner settings. Refer to the Scan Manager library for the location of Scan Demo.

Scanner Commands

Introduction

The Scan Manager API in this section give you commands to manipulate the scanner. Using these commands, an application should perform the following functions:

- Enable or disable scanning
- Start a decode
- Turn the LED on or off

- Sound any of the defined beep patterns
- Set the scanner into “aim” (laser-pointer) mode
- Get version information for the various terminal software components

Returned Status Definitions

The scanner commands in this chapter may return one of the status codes described in [Table 1-1](#).

Table 1-1. Returned Status Codes

STATUS CODE	DEFINITION
Any non-negative value (0 to 32767)	Parameter value.
STATUS_OK	The function's parameters were verified. If a function must wait for an ACK from the scanner, STATUS_OK indicates that the ACK was received.
NOT_SUPPORTED	The last packet received from the scanner generated either a NAK_DENIED or NAK_BAD_CONTEXT status. This usually indicates that the specified parameter is not supported by this scanner, or the scanner was unable to comply with the request.
COMMUNICATIONS_ERROR	Either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not verified through an ACK, and therefore, is questionable.
BAD_PARAM	One or more of the function call parameters supplied by the user was not in the expected range.
BATCH_ERROR	The limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with ScanSet are responsible for generating a batch command to establish scanner parameters. The parameters are not sent to the scanner until the ScanCmdSendParams function is called, at which time a new batch is started.
ERROR_UNDEFINED	An error condition exists that is not specifically associated with the scanner or its communications.



Scanner Commands

Table 1-2 lists the scanner commands described in this chapter.

Table 1-2. Scanner Commands

FUNCTION	PAGE
<i>ScanCloseDecoder</i>	1-9
<i>ScanCmdAimOff</i>	1-10
<i>ScanCmdAimOn</i>	1-11
<i>ScanCmdBeep</i>	1-12
<i>ScanCmdGetAllParams</i>	1-14
<i>ScanCmdLedOff</i>	1-15
<i>ScanCmdLedOn</i>	1-16
<i>ScanCmdParamDefaults</i>	1-17
<i>ScanCmdScanDisable</i>	1-18
<i>ScanCmdScanEnable</i>	1-19
<i>ScanCmdSendParams</i>	1-20
<i>ScanCmdStartDecode</i>	1-21
<i>ScanCmdStopDecode</i>	1-22
<i>ScanCmdTrigSledOff</i>	1-23
<i>ScanCmdTrigSledOn</i>	1-24
<i>ScanGetAimMode</i>	1-25
<i>ScanGetDecodedData</i>	1-26
<i>ScanGetExtendedDecodedData</i>	1-29
<i>ScanGetDecoderVersion</i>	1-30
<i>ScanGetLedState</i>	1-31
<i>ScanGetScanEnabled</i>	1-32
<i>ScanGetScanManagerVersion</i>	1-33
<i>ScanGetScanPortDriverVersion</i>	1-34
<i>ScanGetTrigSledMode</i>	1-35
<i>Barcode Parameter Functions</i>	1-37

ScanCloseDecoder

Purpose	Closes the Scan Manager shared library and frees up system resources.	
Prototype	<code>Int16 ScanCloseDecoder (void);</code>	
Returned Status	<code>Zero</code>	No errors closing shared library
	<code>Non-zero</code>	Error closing shared library
Comments	Must be called by all applications that call the ScanOpenDecoder function. Failure to do so will cause system errors and unpredictable results.	
See Also	ScanOpenDecoder	



ScanCmdAimOff

Purpose Takes the scanner out of the “aim” mode (also known as “laser pointer” mode).

Prototype `Int16 ScanCmdAimOff (void);`

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanCmdAimOn*](#)

ScanCmdAimOn

Purpose Places the scanner into its “aim” mode (also known as “laser pointer” mode).

Prototype Int16 ScanCmdAimOn (void);

Returned Status STATUS_OK

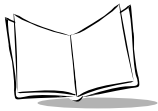
BAD_PARAM Error

COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

Comments This function call only tells the scanner that you want to use the scanner laser for aiming, not decoding. To execute the aim, the user must press the scanner’s trigger, or a [ScanCmdStartDecode](#) command must be sent.

See Also [ScanCmdAimOff](#)



ScanCmdBeep

Purpose	Executes the specified beep sequence.	
Prototype	<code>Int16 ScanCmdBeep (BeepType beep);</code>	
Parameters	<code>-> beep</code>	Must be one of the following values ONE_SHORT_HIGH TWO_SHORT_HIGH THREE_SHORT_HIGH FOUR_SHORT_HIGH FIVE_SHORT_HIGH ONE_SHORT_LOW TWO_SHORT_LOW THREE_SHORT_LOW FOUR_SHORT_LOW FIVE_SHORT_LOW ONE_LONG_HIGH TWO_LONG_HIGH THREE_LONG_HIGH FOUR_LONG_HIGH FIVE_LONG_HIGH ONE_LONG_LOW TWO_LONG_LOW THREE_LONG_LOW FOUR_LONG_LOW FIVE_LONG_LOW FAST_WARBLE SLOW_WARBLE MIX1 MIX2 MIX3 MIX4

DECODE_BEEP
 BOOTUP_BEEP
 PARAMETER_DEFAULTS_BEEP

Returned Status	STATUS_OK	
	BAD_PARAM	Error
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error



ScanCmdGetAllParams

Purpose	Retrieves the current parameters from the scanner.	
Prototype	<code>Int16 ScanCmdGetAllParams(UInt8 *ptr, UInt16 maxlength);</code>	
Parameters	<code>-> ptr[]</code>	Array where the scanner's parameter information is deposited.
	<code>-> maxlength</code>	Maximum size (in bytes) of the parameter values stored in the <code>ptr[]</code> array.
Returned Status	Number of bytes copied into <code>ptr[]</code> .	
	<code>COMMUNICATIONS_ERROR</code>	Error
	<code>NOT_SUPPORTED</code>	Error
Comments	<p>The location of the array where the parameters are stored begins with <code>ptr[0]</code>. The parameters are returned as data pairs consisting of (<code>parameter_number</code> and <code>parameter_value</code>). You must parse through the data pairs and associate each <code>parameter_number</code> with a specific scanner capability. If the number of bytes you specify in <code>maxlength</code> is less than the number of scanner parameters retrieved, the remaining parameters are lost. To make sure you retrieve all of the parameters, set <code>MemPtr</code> to at least 256 bytes.</p> <p>As you use <code>ScanSet</code> commands to set the decoder's parameters, the new parameters will not be reflected in the <code>ptr[]</code> array. You must update your own parameter storage when you change parameters.</p>	

ScanCmdLedOff

Purpose Immediately turns off the scanner's green LED.

Prototype `Int16 ScanCmdLedOff (void);`

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanCmdLedOn*](#)



ScanCmdLedOn

Purpose Immediately turns on the scanner's green LED.

Prototype `Int16 ScanCmdLedOn (void);`

Returned Status STATUS_OK

BAD_PARAM Error

COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

Comments The LED stays on until the **ScanCmdLedOff** command is sent.

See Also [*ScanCmdLedOff*](#)

ScanCmdParamDefaults

Purpose Sets all parameters to the factory-installed defaults.

Prototype `Int16 ScanCmdParamDefaults (void);`

Returned Status `STATUS_OK`

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error



ScanCmdScanDisable

Purpose Prevents the scanner from activating the laser when the trigger is pressed or a [ScanCmdStartDecode](#) command is received.

Prototype `Int16 ScanCmdScanDisable (void);`

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [ScanCmdScanEnable](#)

ScanCmdScanEnable

Purpose Permits the scanner to activate the laser when the trigger is pressed or a [ScanCmdStartDecode](#) command is received.

Prototype `Int16 ScanCmdScanEnable (void);`

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [ScanCmdScanDisable](#)



ScanCmdSendParams

Purpose Sends to the scanner any parameters changed by your application. Also can initiate a beep when the parameters have been successfully changed.

Prototype `Int16 ScanCmdSendParams(BeepType beep);`

Parameters `-> beep` Set this parameter to one of the BeepType values listed in the ScanMgrDef.h header file. If you do not want a beep, send the NO_BEEP parameter.

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

Comments This function transmits the scanner parameter values set by other functions. If you do not call ScanCmdSendParams after you have called all of your “set” functions, the settings will not take effect.

The values you set are permanent and will persist until either the terminal is reset or until you perform a [ScanCmdParamDefaults](#) command.

The beep parameter is the sound the beeper should make when the parameters have been successfully changed.

ScanCmdStartDecode

Purpose	Instructs the scanner to turn on the laser and begin decoding a barcode.	
Prototype	<code>Int16 ScanCmdStartDecode (void);</code>	
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error
Comments	This command only initiates a scanning session if the trigger mode is set to Host (see ScanGetTriggeringModes). If the scanner was previously set to aim mode by the ScanCmdAimOn command, this command initiates a laser pointer operation. The laser remains on for the value set in ScanGetLaserOnTime x 10.	
See Also	ScanCmdStopDecode	



ScanCmdStopDecode

Purpose Instructs the scanner to abort a decode attempt.

Prototype `Int16 ScanCmdStopDecode (void);`

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanCmdStartDecode*](#)

ScanCmdTrigSledOff

Purpose	Disables Trigger Sled.	
Prototype	<code>Int16 ScanCmdTrigSledOff ();</code>	
Returned Status	<code>STATUS_OK</code>	
	<code>NOT_SUPPORTED</code>	Error
Comments	This function call disables the Trigger Sled. SPT 1550 does not support Trigger Sled Mode.	
See Also	<i>ScanCmdTrigSledOn</i>	

To scan bar codes using the Snap-on Trigger Sled:

1. Start your scanning application.
2. Enable the Trigger Sled through the API or through the *Symbol Preference* panel.
3. Aim the scanner at the bar code.
4. Press the trigger button on the Trigger Sled handle. Make sure the red scan beam spans the entire bar code. The green scan LED lights and a beep sounds to indicate a successful decode.



ScanCmdTrigSledOn

Purpose	Enables Trigger Sled.	
Prototype	<code>Int16 ScanCmdTrigSledOn();</code>	
Returned Status	<code>STATUS_OK</code>	
	<code>NOT_SUPPORTED</code>	Error
Comments	This function call enables the Trigger Sled. SPT 1550 does not support Trigger Sled Mode.	
See Also	<i>ScanCmdTrigSledOff</i>	

To scan bar codes using the Snap-on Trigger Sled:

1. Start your scanning application.
2. Enable the Trigger Sled through the API or through the *Symbol Preference* panel.
3. Aim the scanner at the bar code.
4. Press the trigger button on the Trigger Sled handle. Make sure the red scan beam spans the entire bar code. The green scan LED lights and a beep sounds to indicate a successful decode.

ScanGetAimMode

Purpose	Identifies whether the scanner is in “normal mode” or “aim” mode (for use as a laser pointer).	
Prototype	<code>Int16 ScanGetAimMode (void);</code>	
Returned Status	Zero	normal mode
	Non-zero	Aim mode
See Also	<i>ScanCmdAimOn</i>	
	<i>ScanCmdAimOff</i>	



ScanGetDecodedData

Purpose	Retrieves the decoded data from the last scan. Also fills in the DECODE_DATA_STRUCT structure with barcode type, length, and checksum information.	
Prototype	<code>Int16 ScanGetDecodedData (MESSAGE *ptr);</code>	
Parameters	<code>-> ptr</code>	A pointer to the user-allocated DECODE_DATA_STRUCT where the decoded data is to be placed.
	<code>-> ptr -> length</code>	Number of characters in the decoded data string.
	<code>-> ptr -> data</code>	Contains the decoded data.
	<code>-> ptr -> data</code>	Start of the packet.
	<code>[ptr -> length</code>	Checksum.
	<code>-> ptr -> type</code>	The type of barcode that was decoded:

BCTYPE_NOT_APPLICABLE
BCTYPE_BOOKLAND_EAN
BCTYPE_COUPON_CODE
BCTYPE_CODABAR
BCTYPE_CODE32
BCTYPE_CODE39
BCTYPE_CODE39_FULL_ASCII
BCTYPE_CODE93
BCTYPE_CODE128
BCTYPE_D2OF5
BCTYPE_EAN8
BCTYPE_EAN8_2
SUPPLEMENTALS
BCTYPE_EAN8_5
SUPPLEMENTALS
BCTYPE_EAN13_5
SUPPLEMENTALS
BCTYPE_EAN13
BCTYPE_EAN13_2
SUPPLEMENTALS
BCTYPE_EAN128
BCTYPE_I2OF5
BCTYPE_IATA2OF5
BCTYPE_ISBT128
BCTYPE_MSI_PLESSEY
BCTYPE_TRIOPTIC_CODE39
BCTYPE_UPCA
BCTYPE_UPCA_2
SUPPLEMENTALS
BCTYPE_UPCA_5
SUPPLEMENTALS
BCTYPE_UPCE0
BCTYPE_UPCE0_2
SUPPLEMENTALS
BCTYPE_UPCE0_5
SUPPLEMENTALS
BCTYPE_UPCE1
BCTYPE_UPCE1_2
SUPPLEMENTALS
BCTYPE_UPCE1_5
SUPPLEMENTALS



Returned Status	STATUS_OK	
	BAD_PARAM	Error
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error
Comments	Typically, an application calls this function in response to an EVENT_DECODE_DATA type of event.	
See Also	<i>ScanGetExtendedDecodedData</i>	

ScanGetExtendedDecodedData

Purpose	Retrieves the decoded data larger than 255 bytes (multipacket data) from the last scan.	
Prototype	<pre>Int16 ScanGetExtendedDecodedData (Int16 length, Int16 *type. UInt8 *extendedData);</pre>	
Parameters	length	passed to the function by the application, and is the size of the buffer pointed to by *buf.
	extendedData	pointer to the buffer to place the decoded data.
	type	pointer to an int, and will contain the bar code type after the API is successfully called.
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error
Comments	Typically, an application calls this function in response to an EVENT_DECODE_DATA type of event and when extended data flag is set to indicate that extended data has been decoded. The length of the data is also sent out from the scanner manager to the application. Note that scan angle selection is not supported and ScanCmdAimOn is not supported.	
See Also	<i>ScanGetDecodedData</i>	



ScanGetDecoderVersion

Purpose	Retrieves the ASCII revision string of the scanner's decode software. Also copies the string to a user-specified location.	
Prototype	<pre>Int16 ScanGetDecoderVersion (Char* ptr, UInt16 max_length);</pre>	
Parameters	-> ptr	A pointer to a user-allocated char array. This function places the revision into the array, null terminated.
	-> max_length	Maximum number of characters to be copied to ptr[].
Returned Status	Length of the revision string.	
	BAD_PARAM	Error
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error
Comments	The application should call this function after receiving a REVISION_REPLY_EVENT.	

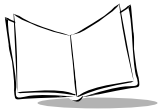
ScanGetLedState

Purpose Indicates whether the green LED is currently on or off.

Prototype `Int16 ScanGetLedState (void);`

Returned Status	Zero	OFF
	Non-zero	ON

See Also [*ScanCmdLedOff*](#)
[*ScanCmdLedOn*](#)



ScanGetScanEnabled

Purpose Indicates whether the scanner is currently enabled.

Prototype `Int16 ScanGetScanEnabled (void);`

Returned Status	Zero	DISABLED
	Non-zero	ENABLED

See Also [*ScanCmdScanEnable*](#)
[*ScanCmdScanDisable*](#)

ScanGetScanManagerVersion

Purpose	Copies the ASCII version string for the Scan Manager software into a user-specified location.	
Prototype	<code>Int16 ScanGetScanManagerVersion (Char* ptr, UInt16 max_length);</code>	
Parameters	<code>-> ptr</code>	A pointer to a user-allocated char array. This function places the version into the array, null terminated.
	<code>-> max_length</code>	Maximum number of characters to be copied to ptr[].
Returned Status	Length of the revision string.	
	<code>NOT_SUPPORTED</code>	Error
See Also	<i>ScanGetScanPortDriverVersion</i>	



ScanGetScanPortDriverVersion

Purpose Copies the ASCII version string for the scan port driver software into a user-specified location.

Prototype `Int16 ScanGetScanPortDriverVersion (Char* ptr, UInt16 max_length);`

Parameters

-> ptr	A pointer to a user-allocated char array. This function places the version into the array, null terminated.
--------	---

-> max_length	Maximum number of characters to be copied to ptr[].
---------------	---

Returned Status Length of the revision string.

NOT_SUPPORTED	Error
---------------	-------

See Also [*ScanGetScanManagerVersion*](#)

ScanGetTrigSledMode

Purpose Identifies whether the Trigger Sled Mode is enabled or not.

Prototype `Int16 ScanGetTrigSledMode();`

Returned Status	Zero	Disabled
	Non-zero	Enabled

Comments SPT 1550 does not support Trigger Sled Mode.

See Also [*ScanCmdTrigSledOff*](#), [*ScanCmdTrigSledOn*](#)



ScanOpenDecoder

Purpose	Loads and initializes the Scan Manager shared library, and initializes the scanner.
Prototype	<code>Int16 ScanOpenDecoder (void);</code>
Returned Status	<code>DECODER_ALREADY_OPEN</code> —The function was previously called without a corresponding call to the <code>ScanCloseDecoder</code> function. <code>STATUS_OK</code>
Comments	Must be called by all applications before any of the other functions in the Scan Manager shared library can be used. Also include a call to the <code>ScanCloseDecoder</code> function.
See Also	<i>ScanCloseDecoder</i>

Barcode Parameter Functions

Introduction

The Scan Manager functions described in this chapter give you the ability to control how the scanner handles various types of barcodes. These functions allow your application to control the following types of settings:

- Which specific barcode types will be decoded
- Which specific barcode lengths will be decoded
- Which conversions will be performed on the decoded data
- Whether to decode Universal Product Code (UPC) preamble and supplemental data
- How many times a barcode is to be scanned to ensure an accurate decode (redundancy)

The Scan Manager software places events into your application's event queue to notify you of pertinent scanner events. The following scanner events, at a minimum, should be handled by your application:

- Decode Event
- Scanning Error

Returned Status Definitions

The function calls listed in this chapter may return one of the status codes described in [Table 1-3](#).

Table 1-3. Returned Status Codes

STATUS CODE	DEFINITION
Any non-negative value (0 to 32767)	Parameter value.
STATUS_OK	The function's parameters were verified. If a function must wait for an ACK from the scanner, STATUS_OK indicates that the ACK was received.



Table 1-3. Returned Status Codes

STATUS CODE	DEFINITION
NOT_SUPPORTED	The last packet received from the scanner generated either a NAK_DENIED or NAK_BAD_CONTEXT status. This usually indicates that the specified parameter is not supported by this scanner, or the scanner was unable to comply with the request.
COMMUNICATIONS_ERROR	Either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not verified through an ACK, and therefore, is questionable.
BAD_PARAM	One or more of the function call parameters supplied by the user was not in the expected range.
BATCH_ERROR	The limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with scanSet are responsible for generating a batch command to establish scanner parameters. The parameters are not sent to the scanner until the ScanCmdSendParams function is called, at which time a new batch is started.
ERROR_UNDEFINED	An error condition exists that is not specifically associated with the scanner or its communications.

Barcode Types

Table 1-4 lists the barcode types that can be enabled by the parameter functions in this chapter.

Table 1-4. Barcode Types

BARCODE TYPE	PAGE
Codabar Barcode Parameter Functions	1-39
Code 32 Barcode Parameter Functions	1-44
Code 39 Barcode Parameter Functions	1-47
General Barcode Parameter Functions	1-52
I 2 of 5 Barcode Parameter Functions	1-63

Table 1-4. Barcode Types (continued)

BARCODE TYPE	PAGE
<i>MSI Plessey Barcode Parameter Functions</i>	1-66
<i>UPC/EAN Barcode Parameter Functions</i>	1-71

The actual parameter functions for each barcode type are listed in the appropriate section.

Codabar Barcode Parameter Functions

[Table 1-5](#) lists the Codabar barcode parameter functions described in this section.

Table 1-5. Codabar Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetCIsiEditing</i>	1-40
<i>ScanGetNotisEditing</i>	1-41
<i>ScanSetCIsiEditing</i>	1-42
<i>ScanSetNotisEditing</i>	1-43



ScanGetClsiEditing

Purpose	Identifies whether the start and stop characters are being stripped from a 14-character Codabar symbol, and a space is being inserted after the first, fifth, and tenth characters.	
Prototype	int ScanGetClsiEditing (void);	
Returned Status	Zero	DISABLE
	>Zero	ENABLE
	COMMUNICATIONS_ERROR	if an error occurs
	NOT_SUPPORTED	if an error occurs
See Also	<i>ScanSetClsiEditing</i>	

ScanGetNotisEditing

Purpose	Identifies whether the start and stop characters are being stripped from a 14-character Codabar symbol.	
Prototype	int ScanGetNotisEditing (void);	
Returned Status	Zero	DISABLE
	>Zero	ENABLE
	COMMUNICATIONS_ERROR	if an error occurs
	NOT_SUPPORTED	if an error occurs
See Also	<i>ScanSetNotisEditing</i>	



ScanSetClsiEditing

Purpose When enabled, strips the start and stop characters from a 14-character Codabar symbol, and inserts a space after the first, fifth, and tenth characters.

Prototype `int ScanSetClsiEditing (Boolean bEnable);`

Parameters `-> bEnable` Must be one of the following values:
True = ENABLE
False = DISABLE

Returned Status `STATUS_OK`

 `>BAD_PARAM` Error

 `BATCH_ERROR` Error

See Also [*ScanGetNotisEditing*](#)

ScanSetNotisEditing

Purpose When enabled, strips the start and stop characters from a 14-character Codabar symbol.

Prototype `int ScanSetNotisEditing (Boolean bEnable);`

Parameters `-> bEnable` Must be one of the following values:
True = ENABLE
False = DISABLE

Returned Status `STATUS_OK`

 `>BAD_PARAM` Error

 `BATCH_ERROR` Error

See Also [*ScanGetNotisEditing*](#)



Code 32 Barcode Parameter Functions

Table 1-6 lists the Code 32 barcode parameter functions described in this section.

Table 1-6. Code 32 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetCode32Prefix</i>	1-45
<i>ScanSetCode32Prefix</i>	1-46

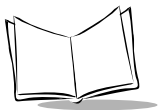
ScanGetCode32Prefix

Purpose Identifies whether the character 'A' is being appended to the beginning of decode data that is in Code 32 format.

Prototype `Int16 ScanGetCode32Prefix (void);`

Returned Status	<code>Zero</code>	<code>DISABLE</code>
	<code>>Zero</code>	<code>ENABLE</code>
	<code>COMMUNICATIONS_ERROR</code>	<code>Error</code>
	<code>NOT_SUPPORTED</code>	<code>Error</code>

See Also [*ScanSetCode32Prefix*](#)



ScanSetCode32Prefix

Purpose Determines whether the character “A” is to be appended to the beginning of decode data that is in Code 32 format.

Prototype `Int16 ScanSetCode32Prefix (Boolean bEnable);`

Parameters `-> bEnable` Must be one of the following values:
True = ENABLE
False = DISABLE

Returned Status `STATUS_OK`

 `BAD_PARAM` Error

 `BATCH_ERROR` Error

See Also [*ScanGetCode32Prefix*](#)

Code 39 Barcode Parameter Functions

Table 1-7 lists the Code 39 barcode parameter functions described in this section.

Table 1-7. Code 39 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetCode39CheckDigitVerification</i>	1-48
<i>ScanGetCode39FullAscii</i>	1-49
<i>ScanSetCode39CheckDigitVerification</i>	1-50
<i>ScanSetCode39FullAscii</i>	1-51



ScanGetCode39CheckDigitVerification

Purpose Identifies whether a Code 39 symbol is complying with specified algorithms.

Prototype `Int16 ScanGetCode39CheckDigitVerification (void);`

Returned Status `ENABLE`

`DISABLE`

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanSetCode39CheckDigitVerification*](#)

ScanGetCode39FullAscii

Purpose Identifies whether an ASCII character code is being assigned to letters, punctuation marks, numerals, and most keyboard control keystrokes.

Prototype `Int16 ScanGetCode39FullAscii (void);`

Returned Status	<code>Zero</code>	<code>DISABLE</code>
	<code>>Zero</code>	<code>ENABLE</code>
	<code>COMMUNICATIONS_ERROR</code>	<code>Error</code>
	<code>NOT_SUPPORTED</code>	<code>Error</code>

See Also [*ScanSetCode39FullAscii*](#)



ScanSetCode39CheckDigitVerification

Purpose Determines whether a Code 39 symbol is to comply with specified algorithms.

Prototype `Int16 ScanSetCode39CheckDigitVerification (
 UInt16 check_digit);`

Parameters `-> check_digit` Must be one of the following values:
 ENABLE
 DISABLE

Returned Status `STATUS_OK`

 `BAD_PARAM` Error

 `BATCH_ERROR` Error

Comments Only those Code 39 symbols that include a modulo 43 check digit are
 decoded when this parameter is enabled.

See Also [*ScanGetCode39CheckDigitVerification*](#)

ScanSetCode39FullAscii

Purpose	Determines whether an ASCII character code is to be assigned to letters, punctuation marks, numerals, and most keyboard control keystrokes.	
Prototype	<code>Int16 ScanSetCode39FullAscii (Boolean bEnable);</code>	
Parameters	<code>-> full_ascii</code>	Must be onf of the following values: True = ENABLE False = DISABLE
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	Code 39 Full ASCII interprets the barcode special character (\$ + % /) preceding a Code 39 character, and assigns an ASCII character value to the pair. For example, when Code 39 Full ASCII is enabled and a +B is scanned, it is interpreted as b; %J as 7; and \$H emulates the keystroke BACKSPACE. Scanning ABC\$M outputs the keystroke equivalent of ABC ENTER. Do not enable Code 39 Full ASCII and Trioptic Code 39 at the same time.	
See Also	<i>ScanGetCode39FullAscii</i>	



General Barcode Parameter Functions

Table 1-8 lists the general barcode parameter functions described in this section.

Table 1-8. General Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetBarcodeEnabled</i>	1-53
<i>ScanGetBarcodeLengths</i>	1-54
<i>ScanGetConvert</i>	1-56
<i>ScanGetTransmitCheckDigit</i>	1-57
<i>ScanSetBarcodeEnabled</i>	1-58
<i>ScanSetBarcodeLengths</i>	1-59
<i>ScanSetConvert</i>	1-61
<i>ScanSetTransmitCheckDigit</i>	1-62

ScanGetBarcodeEnabled

Purpose Determines whether the specified barcode type is currently enabled for decoding.

Prototype `Int16 ScanGetBarcodeEnabled (BarType barcodeType);`

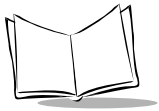
Returned Status The enabled state of the specified barcode type:

Zero=DISABLE
 >zero=ENABLE
 barBOOKLAND_EAN
 barCODABAR
 barCODE39
 barCODE93
 barCODE128
 barCOUPON
 barD2OF5
 barEAN8
 barEAN13
 barI2OF5
 barISBT128
 barMSI_PLESSEY
 barTRIOPTICCODE39
 barUCC_EAN128
 barUPCA
 barUPCE
 barUPCE1

COMMUNICATION_ERROR Error

NOT_SUPPORTED Error

See Also [*ScanSetBarcodeEnabled*](#)



ScanGetBarcodeLengths

Purpose Identifies the number of human-readable symbols in the specified format that are being decoded.

Prototype `Int16 ScanGetBarcodeLengths (
 BarType barcodeType,
 UInt16* pLengthType,
 UInt16* pLength1,
 UInt16* pLength2);`

Returned Status barcodeType will be filled with one of the following values:

barCODABAR
barCODE39
barCODE93
barD25
barI2of5
barMSI_PLESSEY

pLengthType will be filled with one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH
If applicable, pLength1 will be used to return length1.
If applicable, pLength2 will be used to return length2.

STATUS_OK

BAD_PARAM Error

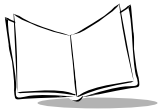
BATCH_ERROR Error

COMMUNICATION_ERROR Error

NOT_SUPPORTED Error

Comments If pLengthType is ONE_DISCRETE_LENGTH, ignore the value returned in pLength2. If pLengthType is ANY_LENGTH, ignore the values returned in pLength1 and pLength2.

See Also [ScanSetBarcodeLengths](#)



ScanGetConvert

Purpose	Identifies whether decoded data is being converted to the specified format before transmission.	
Prototype	<code>Int16 ScanGetConvert (ConvertType conversion);</code>	
Parameters	<code>-> conversion</code>	Must be one of the following values: UPCEtoUPCA UPCE1toUPCA EAN8toEAN13 CODE39toCODE32 I2OF5toEAN13
Returned Status	<code>Zero</code>	DISABLE
	<code>>Zero</code>	ENABLE
	<code>COMMUNICATIONS_ERROR</code>	Error
	<code>NOT_SUPPORTED</code>	Error
See Also	ScanSetConvert	

ScanGetTransmitCheckDigit

Purpose Identifies whether the specified code is being transmitted with a check digit.

Prototype `Int16 ScanGetTransmitCheckDigit (barType barcodeType);`

Parameters `-> barcodeType` Must be one of the following values:
 barUPCA
 barUPCE
 barUPCE1
 barCODE39
 bar12OF5
 barMSI_PLESSEY

Returned Status The barcode format specified in the
 `ScanSetTransmitCheckDigit` function call.

`TRANSMIT_CHECK_DIGIT`

`DO_NOT_TRANSMIT_CHECK_DIGIT`

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanSetTransmitCheckDigit*](#)



ScanSetBarcodeEnabled

Purpose	Dictates whether the specified barcode type is to be enabled for decoding.	
Prototype	<code>Int16 ScanSetBarcodeEnabled (BarType barcodeType, Boolean bEnable);</code>	
Parameters	<code>-> barcodeType</code>	Must be one of the following values: barBOOKLAND_EAN barCODABAR barCODE39 barCODE93 barCODE128 barD2OF5 barEAN8 barEAN13 barI2OF5 barISBT128 barMSI_PLESSEY barTRIOPTICCODE39 barUCC_EAN128 barUPCA barUPCE barUPCEANCOUPONCODE barUPCE1
	<code>-> bEnable</code>	Must be one of the following values: True=ENABLE False=DISABLE
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
See Also	<i>ScanGetBarcodeEnabled</i>	

ScanSetBarcodeLengths

Purpose	Determines the number of human-readable symbols in the specified format that are to be decoded.	
Prototype	<pre>Int16 ScanSetBarcodeLengths (BarType barcodeType, UInt16 lengthType, UInt16 length1, UInt16 length2);</pre>	
Parameters	-> barcodeType	Must be one of the following values: barCODABAR barCODE39 barCODE93 barD25 barI2of5 barMSI_PLESSEY
	-> lengthType	Must be one of the following values: ONE_DISCRETE_LENGTH TWO_DISCRETE_LENGTHS LENGTH_WITHIN_RANGE ANY_LENGTH
	-> length1, length2	The discrete lengths you wish to decode, or the range of barcode lengths you wish to decode. These lengths are ignored if the ANY_LENGTH parameter is set.
Returned Status	STATUS_OK	



BAD_PARAM	Error
BATCH_ERROR	Error

Comments

The number of human-readable characters in the specified barcode format (including check digits) that are to be decoded may be set for:

- One discrete length: Decode only those codes that contain a selected length. For example, if you select `ONE_DISCRETE_LENGTH` and pass a length value of 14, only the barcode symbols containing 14 characters are decoded. Codes that contain two discrete lengths (`length2`) are ignored.
- Two discrete lengths: Decode only those codes that contain two selected lengths. For example, if you select `TWO_DISCRETE_LENGTHS` and pass length values of 2 and 14, only the barcode symbols containing 2 or 14 characters are decoded.
- Lengths within a specified range: Decode those codes that contain a specified range of characters. If you select `LENGTH_WITHIN_RANGE` and pass length values of 4 and 12, only the barcode symbols containing between 4 and 12 characters are decoded.
- Any length: Decode specified barcode symbols containing any number of characters. The length values that you pass are ignored. Codes that contain one discrete length or two discrete lengths are ignored.
If Code 39 Full ASCII is enabled, try to use the `LENGTH_WITHIN_RANGE` or `ANY_LENGTH` options.

See Also

[ScanGetBarcodeLengths](#)

ScanSetConvert

Purpose	Converts decoded data to the specified format before transmission.	
Prototype	<pre>Int16 ScanSetConvert (ConvertType conversion, Boolean bEnable);</pre>	
Parameters	-> conversion	Must be one of the following values: UPCEtoUPCA UPCE1toUPCA EAN8toEAN13 CODE39toCODE32 I2OF5toEAN13
	-> bEnable	Must be one of the following values: True=ENABLE False=DISABLE
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	BATCH_ERROR	Error
Comments	<p>Converting UPC-E to UPC-A—To transmit UPC-E (zero suppressed) decoded data, select DISABLE. After being converted, the data follows UPC-A format conventions, and is affected by UPC-A programming selections (such as preamble, check digit).</p> <p>Converting UPC-E1 to UPC-A—To transmit UPC-E1 (zero suppressed) decoded data, select DISABLE. After being converted, the data follows UPC-A format conventions and is affected by UPC-A programming selections (such as, preamble or check digit).</p> <p>Converting EAN-8 to EAN-13—When EAN Zero Extend is disabled, this parameter has no effect on barcode data.</p> <p>Converting I 2 of 5 to EAN-13—The I 2 of 5 code must be enabled, one length must be set to 14, and the code must have a leading zero and a valid EAN-13 check digit.</p>	
See Also	ScanGetConvert	



ScanSetTransmitCheckDigit

Purpose Determines whether the specified code is to be transmitted with a check digit.

Prototype Int16 ScanSetTransmitCheckDigit (
BarType barcodeType,
UInt16 check_digit);

Parameters

-> barcodeType	Must be one of the following values: barUPCA barUPCE barUPCE1 barCODE39 bar12OF5 barMSI_PLESSEY
-> check_digit	Must be one of the following values: TRANSMIT_CHECK_DIGIT DO_NOT_TRANSMIT_CHECK_DIGIT

Returned Status STATUS_OK

BAD_PARAM Error

BATCH_ERROR Error

Comments Check digits are used by the scanner to validate that the correct data has been decoded. In UPC code, the check digit's value is based on the other data in the barcode.

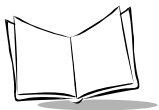
See Also [*ScanGetTransmitCheckDigit*](#)

1 2 of 5 Barcode Parameter Functions

Table 1-9 lists the 1 2 of 5 barcode parameter functions described in this section.

Table 1-9. 1 2 of 5 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGet12of5CheckDigitVerification</i>	1-64
<i>ScanSet12of5CheckDigitVerification</i>	1-65



ScanGetI2of5CheckDigitVerification

Purpose Identifies whether an I 2 of 5 symbol is complying with specified algorithms.

Prototype `Int16 ScanGetI2of5CheckDigitVerification (void);`

Returned Status `DISABLE`

`OPCC_CHECK_DIGIT`

`USS_CHECK_DIGIT`

`COMMUNICATIONS_ERROR` **Error**

`NOT_SUPPORTED` **Error**

See Also [*ScanSetI2of5CheckDigitVerification*](#)

ScanSetI2of5CheckDigitVerification

Purpose Determines whether an I 2 of 5 symbol is to comply with specified algorithms.

Prototype `Int16 ScanSetI2of5CheckDigitVerification (UInt16
check_digit);`

Parameters `-> check_digit` Must be one of the following values:
DISABLE
USS_CHECK_DIGIT
OPCC_CHECK_DIGIT

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`BATCH_ERROR` Error

Comments The I 2 of 5 symbol must comply with one of the following algorithms:

- Optical Product Code Council (OPCC)
- Uniform Symbology Specification (USS)

See Also [*ScanGetI2of5CheckDigitVerification*](#)



MSI Plessey Barcode Parameter Functions

Table 1-10 lists the MSI Plessey barcode parameter functions described in this section.

Table 1-10. MSI Plessey Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetMsiPlesseyCheckDigit Algorithm</i>	1-67
<i>ScanGetMsiPlesseyCheckDigits</i>	1-68
<i>ScanSetMsiPlesseyCheckDigit Algorithm</i>	1-69
<i>ScanSetMsiPlesseyCheckDigits</i>	1-70

ScanGetMsiPlesseyCheckDigit Algorithm

Purpose Determines whether MSI Plessey-encoded symbols with two check digits are being verified a second time before being transmitted.

Prototype `Int16 ScanGetMsiPlesseyCheckDigitAlgorithm (void);`

Returned Status `MOD10_MOD11`

`MOD10_MOD10`

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanSetMsiPlesseyCheckDigit Algorithm*](#)



ScanGetMsiPlesseyCheckDigits

Purpose Determines the number of check digits that are being inserted at the end of MSI Plessey-encoded data.

Prototype `Int16 ScanGetMsiPlesseyCheckDigits (void);`

Returned Status `ONE_CHECK_DIGIT`

`TWO_CHECK_DIGITS`

`COMMUNICATIONS_ERROR` **Error**

`NOT_SUPPORTED` **Error**

See Also [*UPC/EAN Barcode Parameter Functions*](#)

ScanSetMsiPlesseyCheckDigit Algorithm

Purpose	Determines whether MSI Plessey-encoded symbols with two check digits are to be verified a second time before being transmitted.	
Prototype	<pre>Int16 ScanSetMsiPlesseyCheckDigitAlgorithm (UInt16 algorithm);</pre>	
Parameters	-> algorithm	Must be one of the following values: MOD10_MOD11 MOD10_MOD10
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	BATCH_ERROR	Error
See Also	<i>ScanGetMsiPlesseyCheckDigit Algorithm</i>	



ScanSetMsiPlesseyCheckDigits

Purpose Determines the number of check digits that are to be inserted at the end of MSI Plessey-encoded data.

Prototype `Int16 ScanSetMsiPlesseyCheckDigits (UInt16 check_digits);`

Parameters `-> check_digits` Must be one of the following values:
ONE_CHECK_DIGIT
TWO_CHECK_DIGITS

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`BATCH_ERROR` Error

Comments The check digits at the end of the barcode verify the integrity of the data. At least one check digit is always required. Check digits are not automatically transmitted with the data.

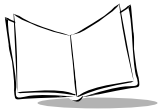
See Also [*ScanGetMsiPlesseyCheckDigits*](#)

UPC/EAN Barcode Parameter Functions

Table 1-11 lists the UPC/EAN (European Article Numbering) barcode parameter functions described in this section:

Table 1-11. UPC/EAN Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
<i>ScanGetDecodeUpcEanRedundancy</i>	1-72
<i>ScanGetDecodeUpcEanSupplementals</i>	1-73
<i>ScanGetEanZeroExtend</i>	1-74
<i>ScanGetUpcEanSecurityLevel</i>	1-75
<i>ScanGetUpcPreamble</i>	1-76
<i>ScanSetDecodeUpcEanRedundancy</i>	1-77
<i>ScanSetDecodeUpcEanSupplementals</i>	1-78
<i>ScanSetEanZeroExtend</i>	1-79
<i>ScanSetUpcEanSecurityLevel</i>	1-80
<i>ScanSetUpcPreamble</i>	1-82



ScanGetDecodeUpcEanRedundancy

Purpose When the autodiscriminate UPC/EAN supplementals parameter is selected in the ScanSetDecodeUpcEanRedundancy function, it identifies the number of times a symbol without supplementals is decoded before being transmitted.

Prototype `Int16 ScanGetDecodeUpcEanRedundancy (void);`

Returned Status Integer in the range [0...20].

COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

See Also [*ScanGetDecodeUpcEanRedundancy*](#)

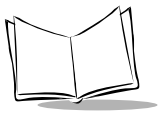
ScanGetDecodeUpcEanSupplementals

Purpose Identifies how UPC or EAN code that includes supplemental characters is being decoded.

Prototype `Int16 ScanGetDecodeUpcEanSupplementals (void);`

Returned Status `DECODE_SUPPLEMENTALS`
`IGNORE_SUPPLEMENTALS`
`AUTODISCRIMINATE_SUPPLEMENTALS`
`COMMUNICATIONS_ERROR` Error
`NOT_SUPPORTED` Error

See Also [*ScanSetDecodeUpcEanSupplementals*](#)



ScanGetEanZeroExtend

Purpose Determines whether five leading zeros are being added to decoded EAN-8 symbols.

Prototype `Int16 ScanGetEanZeroExtend (void);`

Returned Status	<code>Zero</code>	<code>DISABLE</code>
	<code>>Zero</code>	<code>ENABLE</code>
	<code>COMMUNICATIONS_ERROR</code>	<code>Error</code>
	<code>NOT_SUPPORTED</code>	<code>Error</code>

See Also [*ScanSetEanZeroExtend*](#)

ScanGetUpcEanSecurityLevel

Purpose Identifies the number of times the barcode is scanned before being decoded.

Prototype `Int16 ScanGetUpcEanSecurityLevel (void);`

Returned Status

<code>SECURITY_LEVEL0</code>	
<code>SECURITY_LEVEL1</code>	
<code>SECURITY_LEVEL2</code>	
<code>SECURITY_LEVEL3</code>	
<code>COMMUNICATIONS_ERROR</code>	Error
<code>NOT_SUPPORTED</code>	Error

See Also [*ScanSetUpcEanSecurityLevel*](#)



ScanGetUpcPreamble

Purpose Identifies whether the specified UPC code is being transmitted with lead-in characters.

Prototype `Int16 ScanGetUpcPreamble (BarType barcodeType);`

Parameters `-> barcodeType` Must be one of the following values:
barUPCA
barUPCE
barUPCE1

Returned Status One of the following values:
NO_PREAMBLE
SYSTEM_CHARACTER
SYSTEM_CHARACTER_COUNTRY_CODE

COMMUNICATIONS_ERROR Error
NOT_SUPPORTED Error

See Also [*ScanSetUpcPreamble*](#)

ScanSetDecodeUpcEanRedundancy

Purpose	With the autodiscriminate UPC/EAN supplementals option selected, it adjusts the number of times a symbol without supplementals is to be decoded before being transmitted.	
Prototype	<pre>Int16 ScanSetDecodeUpcEanRedundancy (UInt16 supplemental_redundancy);</pre>	
Parameters	-> supplemental_redundancy	Must be an integer in the range [2...20].
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	BATCH_ERROR	Error
Comments	The range is from two to 20 times. Five or above is recommended when decoding a mix of UPC/EAN symbols with and without supplementals, and the autodiscriminate option is selected.	
See Also	ScanGetDecodeUpcEanRedundancy	



ScanSetDecodeUpcEanSupplementals

Purpose Determines how UPC or EAN code that includes supplemental characters is to be decoded.

Prototype `Int16 ScanSetDecodeUpcEanSupplementals (UInt16 supplementals);`

Parameters `-> supplementals` Must be one of the following values:
DECODE_SUPPLEMENTALS
IGNORE_SUPPLEMENTALS
AUTODISCRIMINATE_
SUPPLEMENTALS

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`BATCH_ERROR` Error

Comments Supplementals are two or five characters added to code according to specific format conventions (for example, UPC A+2, UPC E+2, EAN 8+2). Three options are available:

- If you select the `decode_supplementals` parameter, UPC/EAN symbols that don't have supplemental characters are not decoded.
- If you select the `ignore_supplementals` parameter, UPC/EAN symbols that have supplemental characters are decoded, and the supplemental characters are ignored.
- If you select the `autodiscriminate_supplementals` parameter, you can adjust the number of times a symbol is scanned to ensure that both the barcode and the supplementals are correctly decoded. If you use `autodiscriminate`, consider setting redundancy to greater than five.

See Also [*ScanGetDecodeUpcEanSupplementals*](#)

ScanSetEanZeroExtend

Purpose	When enabled, adds five leading zeros to decoded EAN-8 symbols.	
Prototype	<code>Int16 ScanSetEanZeroExtend (Boolean bEnable);</code>	
Parameters	<code>-> bEnable</code>	Must be one of the following values: True=ENABLE False=DISABLE
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	This function call makes EAN-8 symbols compatible to EAN-13 symbols.	
See Also	<i>ScanGetUpcEanSecurityLevel</i>	



ScanSetUpcEanSecurityLevel

Purpose	Selects the number of times the barcode is to be scanned before being decoded.	
Prototype	<code>Int16 ScanSetUpcEanSecurityLevel (UInt16 security_level);</code>	
Parameters	<code>-> security_level</code>	Must be one of the following values: SECURITY_LEVEL0 SECURITY_LEVEL1 SECURITY_LEVEL2 SECURITY_LEVEL3
Returned Status	<code>STATUS_OK</code>	

BAD_PARAM	Error
BATCH_ERROR	Error

Comments

The SPT scanner offers four levels of decoding security for UPC/EAN barcodes. Security levels determine the number of times linear barcodes (such as Code 39 or I 2 of 5) are scanned before being decoded. Higher security levels are needed for decreasing barcode quality. Data must be decoded the same twice in a row for the scan to be considered good. As security levels increase, the scanner's aggressiveness decreases, so be sure to choose only that level of security necessary for any given application.

- Security Level 0—The default setting. Allows the scanner to operate in its most aggressive state, while providing sufficient security for decoding in-spec barcodes.
- Security Level 1—As barcode quality levels diminish, certain characters (1, 2, 7, or 8) become prone to misdecodes. Select this security level if you are experiencing misdecodes because of poorly printed barcodes, and the misdecodes are limited to these characters.
- Security Level 2—Select this security level if you are experiencing misdecodes of poorly printed barcodes, and the misdecodes are not limited to characters 1, 2, 7, or 8.
- Security Level 3—Select this security level if you have tried Security Level 2 and are still experiencing misdecodes. This security level significantly impairs the decoding ability of the scanner. If this level of security is necessary, try to improve the barcode's quality.

See Also

[*ScanGetUpcEanSecurityLevel*](#)



ScanSetUpcPreamble

Purpose Determines whether the specified UPC code is to be transmitted with lead-in characters.

Prototype `Int16 ScanSetUpcPreamble (BarType barcodeType, Int16 preamble);`

Parameters	<code>-> barcodeType</code>	Must be one of the following values: barUPCA barUPCE barUPCE1
	<code>-> preamble</code>	Must be one of the following values: NO_PREAMBLE SYSTEM_CHARACTER SYSTEM_CHARACTER_COUNTRY_CODE

Returned Status	STATUS_OK	
	BAD_PARAM	Error
	BATCH_ERROR	Error

Comments Three options are given for transmitting lead-in characters (preamble) added to UPC-A symbols:

- Transmit system character only
- Transmit system character and country code ("0" for USA)
- Do not transmit the preamble

The preamble is considered part of the symbol.

See Also [*ScanGetUpcPreamble*](#)

Hardware Parameter Functions

Introduction

The Scan Manager functions in this section give you ability to set up the scanner. With these functions, an application can perform the following:

- Set scan angle and aim duration
- Set triggering mode
- Set beep durations and frequencies
- Set redundancy and security levels
- Identify and manipulate barcode data

Returned Status Definitions

The hardware parameter functions may return one of the status codes described in [Table 1-12](#).

Table 1-12. Returned Status Codes

STATUS CODE	DEFINITION
Any non-negative value (0 to 32767)	Parameter value.
STATUS_OK	The function's parameters were verified. If a function must wait for an ACK from the scanner, STATUS_OK indicates that the ACK was received.
NOT_SUPPORTED	The last packet received from the scanner generated either a NAK_DENIED or NAK_BAD_CONTEXT status. This usually indicates that the specified parameter is not supported by this scanner, or the scanner was unable to comply with the request.
COMMUNICATIONS_ERROR	Either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not verified through an ACK, and therefore, is questionable.
BAD_PARAM	One or more of the function call parameters supplied by the user was not in the expected range.

**Table 1-12. Returned Status Codes (continued)**

STATUS CODE	DEFINITION
BATCH_ERROR	The limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with <code>ScanSet</code> are responsible for generating a batch command to establish scanner parameters. The parameters are not sent to the scanner until the ScanCmdSendParams function is called, at which time a new batch is started.
ERROR_UNDEFINED	An error condition exists that is not specifically associated with the scanner or its communications.

Hardware Parameter Functions

Table 1-13 lists the hardware parameter functions described in this chapter.

Table 1-13. Hardware Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetAimDuration	1-86
ScanGetBeepAfterGoodDecode	1-87
ScanGetBeepDuration	1-88
ScanGetBeepFrequency	1-89
ScanGetBidirectionalRedundancy	1-90
ScanGetDecodeLedOnTime	1-91
ScanGetLaserOnTime	1-92
ScanGetLinearCodeTypeSecurityLevel	1-93
ScanGetPrefixSuffixValues	1-94
ScanGetAngle	1-95
ScanGetScanDataTransmissionFormat	1-96
ScanGetTransmitCodeIdCharacter	1-97
ScanGetTriggeringModes	1-98
ScanIsPalmSymbolUnit	1-99

Table 1-13. Hardware Parameter Functions (continued)

PARAMETER FUNCTION	PAGE
<i>ScanSetAimDuration</i>	1-100
<i>ScanSetAngle</i>	1-101
<i>ScanSetBeepAfterGoodDecode</i>	1-102
<i>ScanSetBeepDuration</i>	1-103
<i>ScanSetBeepFrequency</i>	1-104
<i>ScanSetBidirectionalRedundancy</i>	1-105
<i>ScanSetDecodeLedOnTime</i>	1-106
<i>ScanSetLaserOnTime</i>	1-107
<i>ScanSetLinearCodeTypeSecurityLevel</i>	1-108
<i>ScanSetPrefixSuffixValues</i>	1-110
<i>ScanSetScanDataTransmissionFormat</i>	1-111
<i>ScanSetTransmitCodeIdCharacter</i>	1-112
<i>ScanSetTriggeringModes</i>	1-117



ScanGetAimDuration

Purpose Identifies the amount of time the aiming pattern is seen before a scan begins.

Prototype `Int16 ScanGetAimDuration (void);`

Returned Status Integer in the range [0...99], representing a time period of 0.0 to 9.9 seconds, in 0.1-second increments.

COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

See Also [*ScanGetAimDuration*](#)

ScanGetBeepAfterGoodDecode

Purpose Identifies whether the unit has been set to beep after a good decode.

Prototype `Int16 ScanGetBeepAfterGoodDecode (void);`

Returned Status	Zero	DISABLE
	> Zero	ENABLE
	COMMUNICATIONS_ERROR	Error
	NOT_SUPPORTED	Error

See Also [*ScanSetBeepAfterGoodDecode*](#)



ScanGetBeepDuration

Purpose	Identifies the duration of the beep for the specified beep types.	
Prototype	<code>Int16 ScanGetBeepDuration (DurationType type);</code>	
Parameters	<code>-> type</code>	Must be one of the following values: DECODE SHORT MEDIUM LONG
Returned Status	<code>STATUS_OK</code>	
See Also	<i>ScanSetBeepDuration</i>	

ScanGetBeepFrequency

Purpose	Gets the frequency of the beeper for the specified beep types.	
Prototype	Int16 ScanGetBeepFrequency (FrequencyType beep_type);	
Parameters	-> beep_type	Must be one of the following values: DECODE LOW MEDIUM HIGH
Returned Status	STATUS_OK	
See Also	<i>ScanSetBeepFrequency</i>	



ScanGetBidirectionalRedundancy

Purpose Identifies whether a barcode must be successfully scanned in both directions before being decoded.

Prototype `Int16 ScanGetBidirectionalRedundancy (void);`

Returned Status `ENABLE`

`DISABLE`

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanSetBidirectionalRedundancy*](#)

ScanGetDecodeLedOnTime

Purpose Identifies the amount of time the LED is to be turned on when a successful scan is performed.

Prototype `Int16 ScanGetDecodeLedOnTime (void);`

Returned Status Integer in the range [0...100], representing a time period of 0.0 to 10.0 seconds, in 0.1-second increments.

`COMMUNICATIONS_ERROR` Error

`NOT_SUPPORTED` Error

See Also [*ScanSetDecodeLedOnTime*](#)



ScanGetLaserOnTime

Purpose	Identifies the maximum scanner processing time allowed during a scan.		
Prototype	<code>Int16 ScanGetLaserOnTime (void);</code>		
Returned Status	Integer in the range [5...99], representing a time period of 0.5 to 9.9 seconds, in 0.1-second increments.		
	<code>COMMUNICATIONS_ERROR</code>		Error
	<code>NOT_SUPPORTED</code>		Error
See Also	<i>ScanSetLaserOnTime</i>		

ScanGetLinearCodeTypeSecurityLevel

Purpose Identifies the number of times the barcode is scanned before being decoded.

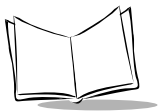
Prototype `Int16 ScanGetLinearCodeTypeSecurityLevel (void);`

Returned Status SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3
SECURITY_LEVEL4

COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

See Also Comment for [ScanSetPrefixSuffixValues](#)



ScanGetPrefixSuffixValues

Purpose Identifies any prefix or suffixes appended to the scanned data.

Prototype

```
Int16 ScanGetPrefixSuffixValues (  
    Char* pPrefix,  
    Char* pSuffix_1,  
    Char* pSuffix_2);
```

Returned Status

MemPtr[0] returns prefix
MemPtr[1] returns suffix_1
MemPtr[2] returns suffix_2

STATUS_OK

BAD_PARAM

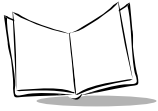
COMMUNICATIONS_ERROR Error

NOT_SUPPORTED Error

See Also [ScanSetPrefixSuffixValues](#) for prefix/suffix values

ScanGetAngle

Purpose	Identifies the scanner's field of view.		
Prototype	<code>Int16 ScanGetAngle (void);</code>		
Returned Status	<code>SCAN_ANGLE_WIDE</code>		
	<code>SCAN_ANGLE_NARROW</code>		
	<code>COMMUNICATIONS_ERROR</code>		Error
	<code>NOT_SUPPORTED</code>		Error
See Also	<i>ScanSetAngle</i>		



ScanGetScanDataTransmissionFormat

Purpose Identifies the scan data transmission format.

Prototype `Int16 ScanGetScanDataTransmissionFormat (void);`

Returned Status

DATA_AS_IS	
DATA_SUFFIX1	
DATA_SUFFIX2	
DATA_SUFFIX1_SUFFIX2	
PREFIX_DATA	
PREFIX_DATA_SUFFIX1	
PREFIX_DATA_SUFFIX2	
PREFIX_DATA_SUFFIX1_SUFFIX2	
COMMUNICATIONS_ERROR	Error
NOT_SUPPORTED	Error

See Also [*ScanSetScanDataTransmissionFormat*](#)

ScanGetTransmitCodeIdCharacter

Purpose Determines whether a character has been selected to identify the scanned barcode's code type and the method selected.

Prototype `Int16 ScanGetTransmitCodeIdCharacter (void);`

Returned Status

AIM_CODE_ID_CHARACTER	
DISABLE	
SYMBOL_CODE_ID_CHARACTER	
COMMUNICATIONS_ERROR	Error
NOT_SUPPORTED	Error

See Also [*ScanSetTransmitCodeIdCharacter*](#)



ScanGetTriggeringModes

Purpose	Identifies the type of scan engine trigger.		
Prototype	<code>Int16 ScanGetTriggeringModes (void);</code>		
Returned Status	HOST		
	LEVEL		
	PULSE		
	COMMUNICATIONS_ERROR		Error
	NOT_SUPPORTED		Error
See Also	<i>ScanSetTriggeringModes</i>		

ScanIsPalmSymbolUnit

Purpose	Identifies whether the application is running on an SPT device (Palm organizer with scanner hardware and software).	
Prototype	<code>Int16 ScanIsPalmSymbolUnit (void);</code>	
Returned Status	<code>Zero</code>	Unit is not an SPT device
	<code>Non-zero</code>	Unit is an SPT device
	<code>COMMUNICATIONS_ERROR</code>	Error
	<code>NOT_SUPPORTED</code>	Error
Comments	Use this call when your software needs to run on both an unmodified Palm III device and an SPT device.	



ScanSetAimDuration

Purpose Sets the amount of time the aiming pattern is to be seen before a scan begins.

Prototype `Int16 ScanSetAimDuration (UInt16 aim_duration);`

Parameters `-> aim_duration` Must be an integer in the range [0...99],
representing a time period of 0.0 to 9.9 seconds.

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`BATCH_ERROR` Error

Comments This function is invoked when the trigger is pressed or a [ScanCmdStartDecode](#) command is received. This function call does not apply to the aim signal or to the [ScanCmdAimOn](#) command.
The `aim_duration` parameter is programmable in 0.1-second increments. If a value of 0 is set for `aim_duration`, the aim pattern is disabled.

See Also [ScanGetAimDuration](#)

ScanSetAngle

Purpose	Sets the scanner's field of view.	
Prototype	<code>Int16 ScanSetAngle (UInt16 scanner_angle);</code>	
Parameters	<code>-> scanner_angle</code>	Must be one of the following values: SCAN_ANGLE_WIDE SCAN_ANGLE_NARROW
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	A SCAN_ANGLE_WIDE field of view allows the scanner to decode more barcode characters at the same time.	
See Also	<i>ScanGetAngle</i>	



ScanSetBeepAfterGoodDecode

Purpose	Determines whether the unit is to beep after a good decode.	
Prototype	<code>Int16 ScanSetBeepAfterGoodDecode (Boolean bEnableBeep);</code>	
Parameters	<code>-> bEnableBeep</code>	Must be one of the following values: True=ENABLE False=DISABLE
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	When bEnableBeep is disabled, the beep still operates during parameter menu scanning, and indicates error conditions.	
See Also	<i>ScanGetBeepAfterGoodDecode</i>	

ScanSetBeepDuration

Purpose	Sets the duration of the beep for the specified beep types.	
Prototype	<pre>Int16 ScanSetBeepDuration (DurationType type, Int16 beep_duration);</pre>	
Parameters	-> type	Must be one of the following values: DECODE SHORT MEDIUM LONG
	-> beep_duration	A numeric beep length in milliseconds (ms).
Returned Status	STATUS_OK	
	BAD_PARAM	Error
Comments	Default durations are:	
	Decode Short Medium Long	90 ms 70 ms 90 ms 240 ms
	The acceptable range for any of these durations is 0 to 10,000 ms.	
See Also	ScanGetBeepDuration	



ScanSetBeepFrequency

Purpose Sets the frequency of the beeper for the specified beep types.

Prototype `Int16 ScanSetBeepFrequency (`
`FrequencyType type,`
`Int16 beep_freq);`

Parameters

<code>-> type</code>	Must be one of the following values: DECODE FREQUENCY LOW FREQUENCY MEDIUM FREQUENCY HIGH FREQUENCY
<code>-> beep_freq</code>	A numeric beep frequency in hertz (Hz).

Returned Status

<code>STATUS_OK</code>	
<code>BAD_PARAM</code>	Error

Comments Default durations are:

Decode frequency	3000 Hz
Low frequency	1500 Hz
Medium frequency	3000 Hz
High frequency	7500 Hz

The acceptable range for any of these frequencies is 0 to 15,000 Hz.

See Also [*ScanGetBeepFrequency*](#)

ScanSetBidirectionalRedundancy

Purpose	Requires that a barcode be successfully scanned in both directions before being decoded.	
Prototype	<code>Int16 ScanSetBidirectionalRedundancy (UInt16 redundancy);</code>	
Parameters	<code>-> redundancy</code>	Must be one of the following values: ENABLE DISABLE
	<code>-> beep_freq</code>	A numeric beep frequency in hertz (Hz).
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	This parameter is only valid when the ScanSetPrefixSuffixValues function call has been enabled.	
See Also	ScanGetBidirectionalRedundancy	



ScanSetDecodeLedOnTime

Purpose Sets the amount of time the LED will be turned on when a successful scan is performed.

Prototype `Int16 ScanSetDecodeLedOnTime (UInt16 led_on_time);`

Parameters `-> led_on_time` Must be an integer in the range [0...99],
representing a time period of 0.0 to 9.9 seconds.

Returned Status `STATUS_OK`

 `BAD_PARAM` Error

 `BATCH_ERROR` Error

Comments If a value of 0 is set for `led_on_time`, the LED will not be turned on. The
`led_on_time` parameter is programmable in 0.1-second increments.

See Also [*ScanGetDecodeLedOnTime*](#)

ScanSetLaserOnTime

Purpose	Sets the maximum scanner processing time to be allowed during a scan.	
Prototype	<code>Int16 ScanSetLaserOnTime (UInt16 laser_on_time);</code>	
Parameters	<code>-> led_on_time</code>	Must be an integer in the range [5...99], representing a time period of 0.5 to 9.9 seconds.
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	<p>Your application should use the hardware trigger, instead of the ScanCmdStartDecode command, to initiate a scan. However, if the scanner was previously set to laser pointer mode by the ScanCmdAimOn command and the laser is activated by the ScanCmdStartDecode command, the laser remains on for laser_on_time x 10 seconds.</p> <p>The laser_on_time parameter is programmable in 0.1-second increments.</p>	
See Also	ScanGetLaserOnTime	



ScanSetLinearCodeTypeSecurityLevel

Purpose Selects the number of times the barcode is to be scanned before being decoded.

Prototype `Int16 ScanSetLinearCodeTypeSecurityLevel (UInt16 security_level);`

Parameters `-> security_level` Must be one of the following values:
SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3
SECURITY_LEVEL4

Returned Status STATUS_OK

BAD_PARAM Error

BATCH_ERROR Error

Comments Security levels determine the number of times linear barcodes (such as Code 39 or I 2 of 5) are scanned before being decoded.

Security levels do not apply to code 128 function calls.

Higher security levels are needed for decreasing barcode quality. As security levels increase, the scanner's aggressiveness decreases, so be sure to choose only that level of security necessary for any given application.

Linear Security Level 1: The following code types must be successfully read twice before being decoded:

CODE TYPE	LENGTH
Codabar	All
MSI Plessey	4 or less
D 2 of 5	8 or less
I 2 of 5	8 or less

Linear Security Level 2: The following code types must be successfully read twice before being decoded:

CODE TYPE	LENGTH
All	All

Linear Security Level 3: Code types other than the following must be successfully read twice before being decoded. The following codes must be read three times:

CODE TYPE	LENGTH
MSI Plessey	4 or less
D 2 of 5	8 or less
I 2 of 5	8 or less

Linear Security Level 4: The following code types must be successfully read three times before being decoded:

CODE TYPE	LENGTH
All	All

See Also

[ScanGetLinearCodeTypeSecurityLevel](#)



ScanSetPrefixSuffixValues

Purpose	Appends a prefix or one or two suffixes to scanned data.	
Prototype	<pre>Int16 ScanSetPrefixSuffixValues (Int8 prefix, Int8 suffix_1, Int8 suffix_2);</pre>	
Parameters	-> prefix suffix_1 suffix_2	The desired ASCII values.
Returned Status	STATUS_OK	
	BAD_PARAM	Error
	BATCH_ERROR	Error
Comments	Before setting the prefix/suffix values, set the Scan Data Transmission Format.	
See Also	ScanGetPrefixSuffixValues ScanSetScanDataTransmissionFormat Appendix A, ASCII Equivalents for prefix/suffix values	

ScanSetScanDataTransmissionFormat

Purpose Changes the scan data transmission format.

Prototype `Int16 ScanSetScanDataTransmissionFormat (UInt16 transmission_format);`

Parameters `-> transmission_format` Must be one of the following values:
 DATA_AS_IS
 DATA_SUFFIX1
 DATA_SUFFIX_2
 DATA_SUFFIX1_
 SUFFIX2
 PREFIX_DATA
 PREFIX_DATA_SUFFIX1
 PREFIX_DATA_SUFFIX2
 PREFIX_DATA_SUFFIX1_
 SUFFIX2

Returned Status `STATUS_OK`

`BAD_PARAM` Error

`BATCH_ERROR` Error

See Also [*ScanGetScanDataTransmissionFormat*](#)



ScanSetTransmitCodeIdCharacter

Purpose	Selects a character that identifies the scanned barcode's code type.	
Prototype	<code>Int16 ScanSetTransmitCodeIdCharacter (UInt16 code_id);</code>	
Parameters	<code>-> code_id</code>	Must be one of the following values: SYMBOL_CODE_ID_CHARACTER AIM-CODE_ID_CHARACTER_DISABLE
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	<p>The code ID character is useful when the scanner is decoding more than one code type. The code ID character is inserted between the prefix and the decoded symbol.</p> <p>The user may select:</p> <ul style="list-style-type: none">• No code ID character• Symbol Code ID character• AIM Code ID character <p>The Symbol Code ID characters are listed and defined in Table 1-14.</p> <p>The definitions for each AIM Code ID character contains a three-character string (in the format jcm). These characters are defined in Table 1-15.</p> <p>The Code characters are listed in Table 1-16.</p> <p>The Modifier characters are listed in Table 1-17.</p>	
See Also	<i>ScanGetTransmitCodeIdCharacter</i>	

Table 1-14. Symbol Code ID Characters

CODE	DEFINITION
A	UPC-A, UPC-E, UPC-E1, EAN-8, EAN-13
B	Code 39, Code 32
C	Codabar
D	Code 128 or ISBT 128
E	Code 93
F	Interleaved 2 of 5
G	Discrete 2 of 5 or Discrete 2 of 5 IATA
J	MSI Plessey
K	UCC/EAN-128
L	Bookland EAN
M	Trioptic Code 39
N	Coupon Code

Table 1-15. AIM Code ID Characters

CODE	DEFINITION	REFER TO
]]	Flag character (ASCII 93)	N/A
c	Code character	Table 1-16
m	Modifier character	Table 1-17

Table 1-16. Code Characters

CODE	DEFINITION
A	Code 39, Code 32
C	Code 128 or ISBT 128
E	UPC-A, UPC-E, UPC-E1, EAN-8, EAN-13, UCC/EAN-128
F	Codabar
G	Code 93



Table 1-16. Code Characters (continued)

CODE	DEFINITION
I	Interleaved 2 of 5
M	MSI Plessey
S	Discrete 2 of 5 and Discrete 2 of 5 IATA
X	Bookland EAN, Trioptic Code 39, Coupon Code

Table 1-17. Modifier Characters

BARCODE TYPE	MODIFIER CHAR	OPTION	EXAMPLE
Code 39	0	Decoder has not checked any check characters or performed a full ASCII processing	A full ASCII barcode with check character W, A+I+MI+D+W, is transmitted as J]A7AimId
	1	Decoder has checked one check character	
	3	Decoder has checked and stripped one check character	
	4	Decoder has performed a full ASCII character conversion	
	5	Decoder has performed a full ASCII character conversion and checked one check character	
	7	Decoder has performed a full ASCII character conversion, and checked and stripped one check character	
Trioptic Code 39	0	No options	A Trioptic barcode 412356 is transmitted as J]X0412356
Code 128	0	Standard data packet with no function code 1 character in the first symbol position	A Code 128 barcode with a function code 1 character in the first position, FNCI AimId, is transmitted with an AIM ID of J]C1

Table 1-17. Modifier Characters (continued)

BARCODE TYPE	MODIFIER CHAR	OPTION	EXAMPLE
Code 128 (cont'd)	1	Function code 1 character in the first symbol position	
	2	Function code 1 character in the second symbol position	
I 2 of 5	0	No check digit processing	An I 2 of 5 barcode 4123 without a check digit being checked is transmitted as J104123
	1	Decoder has checked the check digit	
	3	Decoder has stripped the check digit before transmission	
Codabar	0	No check digit processing	A Codabar barcode 4123 without a check digit being checked is transmitted as JF04123
	1	Decoder has checked the check digit	
	3	Decoder has stripped the check digit before transmission	
Code 93	0	No options	A Code 93 barcode 012345678905 is transmitted as JG0012345678905
MSI Plessey	0	Single check digit checked	An MSI Plessey barcode 4123 with a single check digit checked is transmitted as JM04123
	1	Two check digits checked	
	2	Single check digit checked and stripped before transmission	

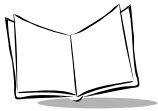


Table 1-17. Modifier Characters (continued)

BARCODE TYPE	MODIFIER CHAR	OPTION	EXAMPLE
	3	Two check digits checked and stripped before transmission	
D 2 of 5	0	No options	A D 2 of 5 barcode 4123 is transmitted as JS04 123
UPC/EAN	0	Standard packet in full EAN country code format: 13 digits for UPC-A and UPC-E (not including supplemental data)	A UPC-A barcode 012345678905 is transmitted as JE00 12345678905
	1	Two-digit supplemental data only	
	2	Five-digit supplemental data only	
	4	EAN-8 data packet	
Bookland EAN	0	No options, always transmit 0	A Bookland barcode 123456789X is transmitted as JX0 123456789X

ScanSetTriggeringModes

Purpose	Sets the type of scan engine trigger.	
Prototype	<code>Int16 ScanSetTriggeringModes (UInt16 triggering_mode);</code>	
Parameters	<code>-> triggering_mode</code>	<p>Must be one of the following values:</p> <p>LEVEL—Only the terminal Scan trigger initiates the scan; the laser is turned off when the trigger is released or the decode was good.</p> <p>PULSE—Only the terminal Scan trigger initiates the scan; the laser is turned off when the value set in ScanSetLaserOnTime is reached or when the decode was good.</p> <p>HOST—The terminal Scan trigger or the application's ScanCmdStartDecode command initiates the scan; the laser is turned off when the trigger is released, or when the value set in ScanSetLaserOnTime is reached, the ScanCmdStopDecode command is called, or the decode was good.</p>
Returned Status	<code>STATUS_OK</code>	
	<code>BAD_PARAM</code>	Error
	<code>BATCH_ERROR</code>	Error
Comments	If the scanner was previously set to aim mode by the ScanCmdAimOn command, each mode functions as described above, except the laser will be on for the value set in ScanSetLaserOnTime x 10, and decoding is disabled.	
See Also	ScanGetTriggeringModes ScanSetLaserOnTime ScanCmdStartDecode ScanCmdStopDecode ScanCmdAimOn	



Power Considerations

The power-consumption characteristics of the SPT device are different than those of a normal Palm III device, and it is important to keep this in mind when writing your application. The normal low-battery alert is displayed whenever the battery voltage falls below the acceptable operating level. However, a scan operation requires a different power threshold. When battery levels fall below this threshold (approximately 2.3 volts) an attempted scan will fail, and the scanner is disabled. Your application must alert the end-user in this situation, explaining why the scan failed, and directing them to install fresh batteries.

scanBatteryErrorEvent

Your code needs to handle the `scanBatteryErrorEvent`. The Scan Manager application generates this event whenever it detects the low-battery condition. Consult one of the two sample programs (ScanDemo or SScan) for an example of how to handle this event. Be sure to catch this event in all of your event handlers that might be in effect when a scan operation is attempted. For example, the ScanDemo program catches the `scanBatteryErrorEvent` in `ApplicationHandleEvent` so that it is handled in whatever form being displayed.

Sudden Loss of Power

If the terminal is put into sleep mode (through the unit's on/off button) while a scan-aware application is running, the state of the scanner will be preserved when the unit is turned back on. If the terminal is put into sleep mode while a scan is in progress, the scan will be aborted before the unit goes to sleep.

If the end-user removes the batteries while a scan-aware application is running (and the unit is not in sleep mode), Scan Manager removes power to the scanner and tries to maintain its current settings. However, this is not recommended, and you may see unpredictable results.

Backlighting

If your application controls or relies on the Palm device's backlighting feature, be aware that Scan Manager turns backlighting off at the outset of a scan operation. It also restores backlighting after the scan is completed.

Other Power Notes

Certain decoder settings affect power consumption, and therefore affect battery life. The “laser pointer” mode set in `ScanCmdAimOn` draws a lot of power. Selecting pulse mode rather than trigger mode generally draws more power. Also, increasing values from the defaults for the following increases power consumption somewhat:

- `LaserOnTime`
- `DecodeLedTime`
- `DecodeBeepDuration`
- `AimDuration`

The scanner draws no power until `ScanOpenDecoder()` is called. It stops drawing power when `ScanDecoderClose()` is called. Therefore, if you need the scanner’s capabilities only during certain portions of your application, you may want to issue the `ScanOpenDecoder` call before you enter that portion of the application, and `ScanCloseDecoder` when you exit that portion of the application. For example, you may need the scanner for entering data in fields on only one form of your application.

To avoid excessive voltage draw, the Scan Manager software puts the terminal into sleep mode while the laser is on. You should be careful to preserve this functionality. For this reason, it is recommended that you do not pass a timeout value to “EvtGetEvent.” (For example, do not generate a `nilEvent` every few milliseconds in a scanning situation). Doing so could cause the terminal to come out of sleep mode at one of your timeout intervals while the laser is firing.

Finally, to reduce instantaneous power draw, your application should avoid opening the IR Exchange Manager or HotSync port while the scanner is open.

Sample Scanning Application

The Scan Manager application described in this chapter is called *SScan*. It is a sample scan-aware application that demonstrates the basics of building a scan-aware application. The sections in this chapter describe, at a high level, the components in the *Sscan* application. The Scan Manager library also includes a detailed application, called *Scan Demo*, that exercises nearly all of the API. Refer to the Scan Manager library for the location of *Scan Demo*.



Writing the Code

Include Files

The following three `#include` statements provide you with the Scan Manager interface definitions, including the API functions, constants, and data structures.

```
#include "ScanMgrDef.h"// Scan Manager constant definitions
```

```
#include "ScanMgrStruct.h"// Scan Manager structure definitions
```

```
#include "ScanMgr.h" // Scan Manager API function definitions
```

PilotMain Routine

The `PilotMain` function is a standard Palm organizer application. It contains the code for handling a normal application launch (`sysAppLaunchCmdNormalLaunch`) by calling three other functions: `StartApplication`, `EventLoop`, and `StopApplication`.

```
/*
 *
 * FUNCTION:      PilotMain
 *
 * DESCRIPTION:   This function is the equivalent of a main()
 *                function in standard C. It is called by the
 *                Emulator to begin execution of this application.
 *
 * PARAMETERS:   cmd - command specifying how to launch the
 *                application.
 *                cmdPBP - parameter block for the command.
 *                launchFlags - flags used to configure the launch.
 *
 * RETURNED:     Any applicable error code.
 *
 */
*****/
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
    // Check for a normal launch.
    if (cmd == sysAppLaunchCmdNormalLaunch)
```

```

{
    Err error = STATUS_OK;

    // Set up Scan Manager and the initial (Main) form.
    StartApplication();

    // Start up the event loop.
    EventLoop();

    // Close down Scan Manager, decoder
    StopApplication();
}

return(0);
}

```

The StartApplication Function

Sscan's `StartApplication` function demonstrates what you need to do at the outset of your program to properly initialize the scanner.

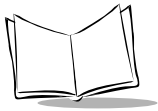
The first thing the `StartApplication` function does is call `ScanIsPalmSymbolUnit`, which tells the application whether it is running on a device that contains scanner hardware and software. This function is useful when your application needs to run on both an unmodified Palm III device and on a SPT device. Based on the result of this call, you can continue either as a normal application or as a scan-aware application.

Before calling any other Scan Manager library function, you **must** call `ScanOpenDecoder`. This function:

- Loads the Scan Manager shared library
- Powers on the decoder
- Initiates communication between the application and the scanner unit

Be sure to check the return value of the `ScanOpenDecoder` call. If it does not return a value of `STATUS_OK`, do **not** proceed with other Scan Manager calls.

If your application successfully performs `ScanOpenDecoder`, you may configure the decoder to suit your application's needs. This could involve enabling the scanner, setting the trigger mode, and enabling the appropriate barcode types. The Sscan application enables the scanner by calling `ScanCmdScanEnable`.



Next, it calls the **ScanSetTriggeringModes** function with a parameter of **HOST** to configure the triggering mode so that software-initiated scanning can be performed. Finally, several UPC and EAN barcode types (or symbologies) are enabled by the function **ScanSetBarcodeEnabled**.

For these parameters to actually take effect, you must call the **ScanCmdSendParams** function. All **ScanSet...** functions must be set with this function call. You only need to call **ScanCmdSendParams** once, after you have set all of your parameters.

NOTE: You are not required to call **ScanCmdSendParams** after you call other **ScanCmd...** functions, such as **ScanCmdScanEnable**. **ScanCmd...** functions take effect automatically.


```

/*****
*
* FUNCTION:      StartApplication
*
* DESCRIPTION:   This routine sets up the initial state of the
*                application. Set the Main Form as the initial form
*                to display. Checks to make sure we're running on
*                Symbol hardware, then calls ScanOpenDecoder to
*                initialize the Scan Manager. If successful, then we
*                proceed with setting decoder parameters that we care
*                about for this application. ScanCmdSendParams is
*                called to send our params to the decoder.
*
* PARAMETERS:    None.
*
* RETURNED:      Nothing.
*
*****/
static void StartApplication(void)
{
    Err error;

    // Call up the main form.
    FrmGotoForm( MainForm );

    // Make sure we're running on Symbol hardware before
    // attempting to
    // open the decoder or call any other Scan Manager functions.
    if (ScanIsPalmSymbolUnit())
    {

        // Now, open the scan manager library
        error = ScanOpenDecoder();

        if (!error)
        {

            // Set decoder parameters we care about...
            ScanCmdScanEnable(); // enable scanning

```



```
        ScanSetTriggeringModes( HOST ); //allow software-
            triggered scans (from our Scan button)
        ScanSetBarcodeEnabled( barUPCA, true ); // Enable any
            barcodes to be scanned
        ScanSetBarcodeEnabled( barUPCE, true );
        ScanSetBarcodeEnabled( barUPCE1, true );
        ScanSetBarcodeEnabled( barEAN13, true );
        ScanSetBarcodeEnabled( barEAN8, true );
        ScanSetBarcodeEnabled( barBOOKLAND_EAN, true);
        ScanSetBarcodeEnabled( barCOUPON, true);

        // We've set our parameters...
        // Call "ScanCmdSendParams" to send them to the decoder
        ScanCmdSendParams( No_Beep);
    }
}
```

The MainFormHandleEvent Function

After calling `StartApplication`, `PilotMain` calls `EventLoop`, which initiates the standard event-processing routine familiar to Palm organizer application developers. From the standpoint of scan-aware application developers, the most interesting code in `Sscan` is the `MainFormHandleEvent` function, which is the event handler for `Sscan`'s main form.

```
/*
*****
*
* FUNCTION:      MainFormHandleEvent
*
* DESCRIPTION:   Handles processing of events for the Main Form.
                  The following events are handled:
                  frmOpenEvent and menuEvent - standard handling
                  scanDecodeEvent - indicates that a scan was
                                      completed
                  scanBatteryErrorEvent - indicates batteries too
                                      low to scan
                  ctlSelectEvent - for Scan button on the main form
*
* PARAMETERS:   event - the most recent event.
*
* RETURNED:     True if the event is handled, false otherwise.
*
*****
*/
```

```

*****/
static Boolean MainFormHandleEvent(EventPtr event)
{
    Boolean      bHandled = false;
    UInt16       extendedDataFlag;

    switch( event->eType )
    {
        case frmOpenEvent:
            MainFormOnInit();
            bHandled = true;
            break;

        case menuEvent:
            MainFormHandleMenu(event->data.menu.itemID);
            bHandled = true;
            break;

        case scanDecodeEvent:
            // A decode has been performed.
            // Use the decoder API to get the decoder data
            // into our memory
            // Get barcode parameters from the registers
            extendedDataFlag= ((ScanEventPtr)event)
            ->scanData.scanGen.data1;
            extendedDataLength =
            (Int16) (((ScanEventPtr)event) -
            >scanData.scanGen.data2);

            extend = extendedDataFlag & EXTENDED_DATA_FLAG;

            OnDecoderData();

            bHandled = true;
            break;

        case scanBatteryErrorEvent:
        {
            Char szTemp[10];
            StrIToA( szTemp, ((ScanEventPtr)event)

```



```
        ->scanData.batteryError.batteryLevel );
        FrmCustomAlert( BatteryErrorAlert, szTemp, NULL,
            NULL );
        bHandled=true;
        break;
    }

    case ctlSelectEvent:
    {
        if (ScanIsPalmSymbolUnit())
        {
            // Scan Button
            if (event->data.ctlEnter.controlID ==
                MainSCANButton)
            {
                ScanCmdStartDecode();
                bHandled = true;
            }
        }
        break;
    }

    case fldChangedEvent;
        UpdateScrollbar();
        bHandled = true;
        break;

    case sclRepeatEvent:
        ScrollLines(event->data.sclRepeat.newvalue - event ->
            data.sclRepeat.value, false);
        break;

    case keyDownEvent;
    {
        if (event->data.keyDown.chr ==pageUpChr) {
            PageScroll (winUp);
            bHandled = true;
        }else if (event->data.keyDown.chr ==pageDownChr) {
            PageScroll (winDown);
            bHandled = true;
        }
        break;
    }
```

```
        {  
    } //end switch  
    return(bHandled);  
}
```



```

/*****
*
* FUNCTION:      MainFormOnInit
*
* DESCRIPTION:   This routine sets up the initial state of the main
*                form
*
* PARAMETERS:    None.
*
* RETURNED:      Nothing.
*
*****/
static void MainFormOnInit()
{
    FormPtr pForm = FrmGetActiveForm();
    if( pForm )
    {
        // initialize the barcode type and barcode data fields
        SetFieldText(MainBarTypeField, "No Data", 20, false );
        SetFieldText(MainScandataField, "No Data", 80, false );
        FrmDrawForm( pForm );
    }
}

/*****
*
* FUNCTION:      MainFormHandleMenu
*
* DESCRIPTION:   This routine handles menu selections off of the
*                main form
*
* PARAMETERS:    None.
*
* RETURNED:      Nothing.
*
*****/

void MainFormHandleMenu( UInt16 menuSel )
{
    switch( menuSel )

```

```

{
    // Options menu
    case OptionsResetDefaults:
        if (ScanIsPalmSymbolUnit()) {
            ScanCmdScanDisable();

            if (ScanCmdParamDefaults () ==Status_OK)
                ScanCmdScanEnable ();
            //enable scanning
        }
        break;

    case OptionsAbout:
        OnAbout();
        break;
}
}

/*****
*
* FUNCTION:      OnDecoderData
*
* DESCRIPTION:   Called when the app receives a scanDecodeEvent,
                  which signals that a decode operation has been
                  completed. Calls the Scan Manager function
                  "ScanGetDecodedData" to get the scan data and
                  barcode type from the last scan. Fills in the
                  controls on the main form that display this
                  information.
*
* RETURNED:      True if the event is handled, false otherwise.
*
*****/

Boolean OnDecoderData()
{
    static Char BarTypeStr[80]=" ";
    MESSAGE    decodeDataMsg;
    Int16      status;
    MemHandle  hExtendedData;

```



```
UInt8 *pExtendedData;
Int16     extendedDataType;
UInt16     numlines;
if (extend) {
    hExtendedData = MemHandleNew ( extendedDataLength);
    pExtendedData = (UInt8 *) MemHandleLock
        (hExtendedData);
    status = ScanGetExtendedDecodedData
        (extendedDataLength, &extendedDataType,
        pExtendedData);
}
else {
    status = ScanGetDecodedData ( &decodeDataMsg);
    extendedDataType = decodeDataMsg.type
    extendedDataLength = decodeDataMsg.length;
    hExtendedData = MemHandleNew (extendedDataLength +1)
    pExtendedData = (UInt8 *) MemHandleLock
        (hExtendedData);
    pExtendedData [extendedDataLength] = '\\0';
    MemMove (&pExtendedData [0], &decodeDataMsg.data [0],
        extendedDataLength+1);
}
if (status == STATUS_OK ) // if we successfully got the decode
                        // data from the API...
    FieldPTR    pField;
    //call a function to translate barcode type into a
    //string, and display it
    ScanGetBarTypeStr (extendedDataType, BarTypeStr, 30);
    //in Utils.c
    SetFieldText (MainBarTypeField, BarTypeStr, 30, true);

    //Check to see if this scan was a "No Read Data"
    //(indicated by type of zero)
    if (extendedDataType ==0)
    {
        SetFieldText (MainScandataField, "No Scan", 30,
            true);
    }
    else
    {
```



```

        // Place the barcode data into the field and
        //display
        /*Set up data display field to display the
        memory*/

    pField = (FieldPtr)GetObjectPtr(MainScandataField);

    pField->attr.editable = true;
    FldDelete (pField, 0, FldGetTextLength (pField));
    //clear out old data

    if (extendedDataLength -> FldGetMaxChars (pField))
        FldSetMaxChars (pField, extendedDataLength);

    FldEraseField (pField); //hide field so we don't see the
                            //data scroll in
    FldInsert(pField, (char*) (&pExtendedData [0],
        extendedDataLength);
    //move to top of scroll bar
    numlines = FldCalcFieldHeight ((char*)&pExtendedData
    [0], SCANDATA_WIDTH);
    ScrollLines (-numlines, false); //scroll to top of data
    FldDrawField(pField);           // show field
    pField->attr.editable = false;
    }
}
UpdateScrollbar();

MemHandleUnlock (hExtendedData);
MemHandleFree (hExtendedData);

return(0);
}

```

MainFormHandleEvent handles five specific events. Most are standard Palm events that are probably already familiar to you. However, two are events that scan-aware applications will need to handle.

- **scanDecodeEvent** is a special event issued by the Scan Manager software to your application. It signals to your code that a scan (either successful or



unsuccessful) has been completed. In response to the `scanDecodeEvent`, you can call the `ScanGetDecodedData` Scan Manager function, which gives you the results from the most recent scan. This is illustrated in the `OnDecoderData` function shown previously.

- `scanBatteryErrorEvent` is another special event issued by Scan Manager to your application. You receive this event whenever a scan operation fails because of low battery levels. When this condition occurs, the scanner is disabled until the batteries are replaced or recharged. Because of this, and because this condition occurs sooner than the normal low-battery warning of a Palm organizer, it is important that you execute some code to alert end-users. The Sscan application does this by issuing an alert.
- `frmOpenEvent` is a standard event that most Palm organizer developers are familiar with. Sscan calls `MainFormOnInit` to initialize and draw the main form.
- `menuEvent` is another standard Palm event. The `menuEvent` code in Sscan allows you to issue a decoder command to reset all of the decoder parameters to their defaults. It also allows you to display an About form with all of the version information for your SPT scanner software.
- `ctlSelectEvent` is an event received by the application in response to the user pushing a button. In Sscan, it is in response to the “Scan” button on the main form. Upon receiving this event, a scan is initiated by calling the `ScanCmdStartDecode` Scan Manager API function.

The StopApplication Function

The `StopApplication` function is called at the conclusion of `PilotMain`. This function first uses `ScanIsPalmSymbolUnit` to make sure the application is running on an SPT device. We recommend this check only if your software might be running on both unmodified Palm III devices and SPT devices. If Sscan is running on an SPT device, we call `ScanCmdScanDisable` to disable scanning. This is not required, merely suggested.

Finally, you must call the `ScanCloseDecoder` function before exiting your program. This function powers down the decoder and closes the Scan Manager shared library. Failure to call `ScanCloseDecoder` can cause unpredictable system problems.

```

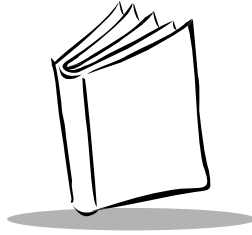
/*****
*
* FUNCTION:      StopApplication
*
* DESCRIPTION:   This routine does any cleanup required, including
*                shutting down the decoder and Scan Manager shared
*                library.
*
* PARAMETERS:    None.
*
* RETURNED:      Nothing.
*
*****/

static void StopApplication(void)
{
    if (ScanIsPalmSymbolUnit())
    {
        // Disable the scanner and Close Scan Manager shared
        // library
        ScanCmdScanDisable();
        ScanCloseDecoder();
    }
}

```



SPT Terminal Series System Software Manual



Chapter 2

MSR 3000

The *MSR 3000* chapter provides information for use in developing applications to enable magnetic stripe reading on the Symbol Palm Terminals.

The MSR 3000 is an external Magnetic Card Reader for the SPT. The Software Development System enables application software developers to easily use and control all the basic and advanced functions of the MSR 3000.

The Software Development System consists of two parts:

Configurator:	Provides an easy-to-use graphical user interface (GUI) for selecting features and setting up the MSR 3000. It runs under Windows 95, 98 and Windows NT 4.0 or above.
MSR Manager Shared Library	A “C” library of all the functions to use and control the MSR 3000. The parameters of the library functions can be generated by the MSR 3000 Configurator, or defined directly by the developer.

This chapter describes both of these tools in detail.

This chapter assumes that you are familiar with the CodeWarrior™ for Palm OS development environment.



Section Descriptions

- [Using The MSR Manager Shared Library](#), describes the use of the API, which enables applications on the SPT terminal to control and receive data from the MSR 3000.
- [MSR Commands](#), describes each API command in detail.
- [MSR 3000 Configurator](#), describes the use of the Configurator, a windows-based tool which enables the developer to easily set up the MSR for use with the API.
- [Using the Configurator to Set the MSR 3000](#), describes how to use the Configurator tool to generate a header file and combine the header file with the application running on the SPT terminal.
- [A Simple Application Program Sample](#), provides an easy to follow application program to use as a reference in your application development.

MSR 3000 Features

All features of the MSR 3000 can be configured with the configurator tool or directly via the shared library interface. The Configurator creates a CodeWarrior include file for the selected MSR 3000 settings, and the application developer can then use the include file and call `MsrSendSet()` function to set up MSR 3000. See [MSR 3000 Configurator](#) for details.

Buffer Mode

Two Buffer Modes are supported on the MSR 3000:

- **Unbuffered Mode** MSR 3000 sends data as soon as the data is available. When using the unbuffered mode, the application program needs to be ready to receive data.
- **Buffered Mode** The application program first sends an “Arm to Read” command to enable the magstripe reading. The user swipes a card, the decoded data is stored in the MSR 3000 data buffer and the MSR 3000 is disarmed. The application program then sends a “Get Trackx” command to retrieve the data from the buffer. All setting functions names `MsrSetXxx` disarm the current read. An application must issue an “Arm to Read again before swiping a card.

If an application is designed to control the card swipe and card data read by itself, then the application developer should use buffered mode. Otherwise, unbuffered mode is recommended.

Terminator/Pre-amble/Post-amble

If Data Edit is disabled, simple message formatting can be done using the Terminator, the Pre-ample and the Post-amble features. This allows a user-definable character string(s) to be added to the data returned by the MSR 3000. Pre-ample is added to the beginning of the data, post-ample is added to the end of data and terminator, and the terminator is added to the end of data. A formatted message block would have the following arrangement:

```
{Pre-ample}{Message Data}{Terminator}{Post-ample}
```

LRC Character

LRC is a check character following the end sentinel in an magnetic stripe card. This option allows the MSR 3000 to be set to either send or not send the LRC character.

Track Selection

There is a maximum of three tracks on a magnetic stripe card which contain encoded data. This feature allows you to specify which track(s) to read.

Track Separator

This option allows the application to select the character used to separate data decoded by multiple track magnetic stripe readers. The Track Separator can be any ASCII character.

Data Edit

The MSR 3000 has very strong Data Editing features. The basic concept is to extract only the data fields (such as Name, Account Number, EXP Date, Address, Age, etc.) required by the application program. MSR 3000 then sends these fields in the order specified by the application program. The application developer does not need to have any knowledge of the specific magstripe format. The application developer can use the configurator tool to make their selections. The Configurator creates a header file for all the data edit commands. These features make the high-level application software development a lot easier. See [Appendix C, Data Editing Overview for Magnetic Stripe Reader](#) for detail.

Special Magnetic Card Format Support

To support special magnetic card formats, the MSR 3000 provides two unique features; the Generic Decoder and the Raw Decoder.

Generic Decoder

The Generic Decoder supports a flexible magnetic card format structure. The Generic Decoder can handle special requirements encoded by the ISO standard 5 or 7 bit data



formats and is more efficient than the raw data decoder, which should support most other special requirements.

The developer can define the following parameters for each track in the generic decoder:

- Bit Format: 5 bits with parity or 7 bits with parity
- Start Sentinel
- End Sentinel
- Special reserved characters. Using this feature the developer can redefine the character for any position in the ISO 7 bits or 5 bits character set table. The maximum number of reserved characters is six.

Raw Data Decoder

The Raw Data Decoder sends magnetic data in raw data format so the application program can perform complicated decoding. With this feature, raw data, which are bit level data in the card, can be sent to the application program for further processing. Two ASCII characters represent each raw data byte: the first ASCII character is for the high digit of the hex code, and the second ASCII character is for the low digit of the hex code. For example, the two ASCII characters "4" and "1" represent raw data 41h (01000001).

Track selection is invalid for the raw data decoder; that is, all encoded data of three tracks in the card is sent.

Track identification is sent before data message for each track, when decoder mode is Raw Data Decoder. Track 1 identification is hex 01, track 2 identification is hex 02 and track 3 identification is hex 03.

Library Globals

```

// maximum characters in a card

#define          MAX_CARD_DATA          400

// maximum characters for pre-amble and post-amble

#define          MAX_PRE_POST_SIZE      10

// maximum added field number

#define          MAX_AFLD_NUM           6

// maximum added field length

#define          MAX_AFLD_LEN           6

// maximum data edit send command number

#define          MAX_SCMD_NUM           4

// maximum length in a data edit send command

#define          MAX_SCMD_LEN           40

// maximum characters in whole data edit send command

#define          MAX_SCMD_CHAR          110

// maximum flexible field number

#define          MAX_FFLD_NUM           16

// maximum length in a flexible field setting command

#define          MAX_FFLD_LEN           20

// maximum characters in whole flexible field setting command

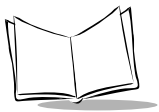
#define          MAX_FFLD_CHAR          60

// maximum reserved character to define

#define          MAX_RES_CHAR_NUM       6

// maximum track number

```



```
#define          MAX_TRACK_NUM          3

// characters for track format

#define          TRACK_FORMAT_LEN       5


// structure of reserved character

typedef          struct  ReservedChar {

    Byte          format;

    char          SR_Bits;

    char          SR_Chars;

} ReservedChar;


Typedef          struct  MSR_Setting {

    Byte          Buffer_mode;

    Byte          Terminator;

    char          Preamble[MAX_PRE_POST_SIZE+1];

    char          Postamble[MAX_PRE_POST_SIZE+1];

    Byte          Track_selection;

    Byte          Track_separator;

    Byte          LRC_setting;

    Byte          Data_edit_setting;

    Byte          Decoder_mode;

    Byte          Track_format[MAX_TRACK_NUM][TRACK_FORMAT_LEN];

ReservedChar     Reserved_chars[MAX_RES_CHAR_NUM];
```

```

        char            Added_field[MAX_AFLD_NUM] [MAX_AFLD_LEN+1];

        char            Send_cmd[MAX_SCMD_NUM] [ MAX_SCMD_LEN] ;

        char            Flexible_field[MAX_FFLD_NUM] [MAX_FFLD_LEN]

    } MSR_Setting;

typedef    MSRSetting*    MSRSetting_Ptr;

typedef    char *          MSRCardInfo_Ptr;

typedef    ReservedChar *ReservedChar_Ptr;

```

Using The MSR Manager Shared Library

Using the API

The MSR Manager shared library API allows SPT application programs to control and receive data from the MSR 3000 Magnetic Stripe Reader.

A typical application program uses the MSR Manager shared library to do the following:

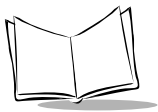
- Open the MSR
- Set the MSR
- Handle MSR data or error messages received from the MSR 3000 MSR.
- Close the MSR.

See [A Simple Application Program Sample](#) for a detailed walk-through of a simple MSR application program.

Using the MSR Demo Application Program

MSR Demo is a demo application program included with the MSR Manager shared library. It includes all of the API, and demonstrates:

- Using the API to set and get MSR 3000 parameters
- Handling MSR data



- Handling errors.

This demo application program also uses the SPT graphic interface to display and change MSR 3000 settings.

MSR Commands

Introduction

The MSR Manager API provides commands to manipulate the MSR 3000.

Return Codes

The MSR commands may return one of the following status codes.

Status Code	Meaning	Suggested Action
MsrMgrNormal	Normal	
MsrMgrErrGlobal	Global parameter error, library global variable error on SPT.	Reset SPT.
MsrMgrErrParam	Invalid parameter in function.	Check the parameters.
MsrMgrErrNotOpen	Shared library is not open.	Open the library before invoke any function call.
MsrMgrErrStillOpen	From MSRLibClose() if the library is still open by others.	
MsrMgrErrMemory	SPT memory error occurred.	Reset SPT.
MsrMgrErrSize	Card information from MSR 3000 MSR overflow.	Check the card and application. Information in a card should not exceed MAX_CARD_DATA (400 (105 for track1, 64 for track2 and 109 for track3) characters, and application should read card information after receiving a MsrDataReadyEvt.
MsrMgrErrNAK	Firmware NAK answer, MSR 3000 reports wrong command was received	check the command or function.
MsrMgrErrTimeout	Waiting timeout.	check the connection between SPT and MSR 3000.

Status Code	Meaning	Suggested Action
MsrMgrErrROM	MSR 3000 ROM check error.	replace MSR 3000 unit.
MsrMgrErrRAM	MSR 3000 RAM check error.	reset MSR 3000.
MsrMgrErrEEPROM	MSR 3000 EEPROM check error.	reset MSR 3000.
MsrMgrErrRes	Error response from MSR 3000.	reset MSR 3000.
MsrMgrErrChecksum	Check sum error.	Reset MSR 3000.
MsrMgrBadRead	Read was failed on selected tracks and buffered mode only.	Swipe the card again and check card format.
MsrMgrLowBattery	Battery voltage is too low to enable MSR 3000.	Recharge SPT battery.
MsrMgrNoData	No data for selected tracks on buffered mode.	
serErrBadPort	Cradle port does not exist.	Check Palm and its OS.
serErrTimeOut	Unable to send or receive data within the specified timeout period.	Check the connection between the SPT and MSR 3000.
serErrAlreadyOpen	SPT Cradle port already has an installed foreground owner.	Check the application.
memErrNotEnoughSpace	No enough memory available on the SPT.	Reset SPT.

MSR 3000 Command Descriptions

MSR Event

MsrDataReadyEvt – this event will be received by an application to indicate that there is data ready to receive from the MSR 3000.

Note: *This command is available in unbuffered mode only.*



MsrOpen

Purpose	Load and initialize the MSR 3000 Manager Library, and return the versions of the shared library and the MSR 3000 attached.	
Prototype	<code>Err MsrOpen (UInt refNum, unsigned long *msrVerP, unsigned long *libVerP)</code>	
Parameters	<code>refNum</code>	library reference number from SysLibLoad or SysLibFind
	<code>MsrVerP</code>	pointer to a MSR 3000 version number
	<code>LibVerP</code>	pointer to a shared library version number
Return	<code>MsrMgrNormal</code>	open successful
	<code>MsrMgrErrGlobal</code>	global parameter error
	<code>MsrMgrErrMemory</code>	memory error occurred
	<code>MsrMgrErrNAK</code>	firmware NAK answer
	<code>MsrMgrLowBattery</code>	battery voltage too low to enable MSR 3000
	<code>MsrMgrErrRes</code>	error response from MSR 3000 MSR
	<code>serErrTimeOut</code>	handshake timeout
	<code>serErrBadPort</code>	port does not exist
	<code>serErrAlreadyOpen</code>	port was open

MemErrNotEnoughSpace insufficient memory

Comments

This is first library function to be called by the application program. This function creates and initializes library globals, and enable power to the MSR. Default serial port settings are set to MSR. It also tests communication with the MSR and gets the version number of the MSR 3000.

This function takes approximately 1 second.

As this function controls power to the MSR 3000, care should be taken with its use. To conserve batteries, we recommend opening the MSR only when serviced, and closing it when not needed. This must be balanced with the 1 second it takes to open the MSR 3000.

Version number follows the system versioning scheme; 0xMMmfsbbb, where MM is major version, m is minor version, f is bug fix, s is stage: 3-release, 2-beta, 1-alpha, 0-development, bbb is build number for non-releases.

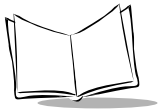
e.g. V1.12b3 would be 0x01122003, V2.00a2 would be 0x02001002 and V1.01 would be 0x01013000.

Example

```
error = MsrOpen(GMsrMgrLibRefNum, &versionNo, &libversion);
```

See Also

MsrClose



MsrClose

Purpose	Close the MSR Manager Library, and free resources.	
Prototype	<code>Err MsrClose (UInt refNum)</code>	
Parameters	<code>refNum</code>	library reference number
Return	<code>MsrMgrNormal</code>	close successful
	<code>MsrMgrErrGlobal</code>	global parameter error
	<code>MsrMgrErrMemory</code>	memory error occurred
	<code>SerErrBadPort</code>	this port does not exist
Comments	Frees the library globals and removes power to MSR.	
Example	<code>error = MsrClose(GMsrMgrLibRefNum) ;</code>	
See Also	<code>MsrOpen</code>	

MsrSetDefault

Purpose	Set MSR 3000 with default settings.	
Prototype	<code>Err MsrSetDefault (UInt refNum)</code>	
Parameters	<code>refNum</code>	library reference number from SysLibLoad or SysLibFind
Return	<code>MsrMgrNormal</code>	setting successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets MSR with following default values:	
	Buffer Mode	Unbuffered
	Terminator	CR/LF
	Preamble	None
	Postamble	None
	Track Selection	All Three Tracks
	Track Separator	CR
	LRC	Do Not Send LRC
	Data Edit Setting	Disabled
Example	<code>error = MsrSetDefault (GMsrMgrLibRefNum) ;</code>	
See Also	<code>MsrClose</code>	



MsrGetSetting

Purpose	Get MSR current settings from MSR 3000.	
Prototype	<code>Err MsrGetSetting (UInt refNum, MSR_Setting *userMSRP)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>userMSRP</code>	pointer to a variable for user MSR setting
Return	<code>MsrMgrNormal</code>	get current setting successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Gets MSR current settings from MSR 3000.	
Example	<code>error = MsrGetSetting(GMsrMgrLibRefNum, &appSetting);</code>	
See Also	<code>MsrSendSetting</code>	

MsrSendSetting

Purpose	Send user settings to the MSR 3000.	
Prototype	<code>Err MsrSendSetting (UInt refNum, MSR_Setting userMSRP)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>userMSR</code>	variable for user MSR setting
Return	<code>MsrMgrNormal</code>	send user setting successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sends user setting to MSR 3000.	
Example	<pre>error = MsrSendSetting(GMsrMgrLibRefNum, user_MsrSetting);</pre>	
See Also	<code>MsrGetSetting</code>	

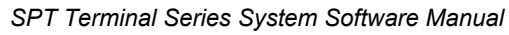


MsrGetVersion

Purpose	Get MSR 3000 and software library version.	
Prototype	<pre>Err MsrGetVersion (UInt refNum, unsigned long *msrVerP, unsigned long *libVerP)</pre>	
Parameters	<code>refNum</code>	library reference number
	<code>msrVerP</code>	pointer to MSR 3000 version
	<code>libVerP</code>	pointer to MSR shared software library
Return	<code>MsrMgrNormal</code>	get status successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	<p>Gets versions of the MSR 3000 and MSR shared software library. Version number follows the system versioning scheme. 0xMMmfsbbb, where MM is major version, m is minor version, f is bug fix, s is stage: 3-release,2-beta,1-alpha,0-development, bbb is build number for non-releases. e.g. V1.12b3 would be 0x01122003, V2.00a2 would be 0x02001002 and V1.01 would be 0x01013000.</p>	
Example	<pre>error = MsrGetVersions(GmsrMgrLibRefNum, &msrVer, &libVer);</pre>	
See Also	<code>MsrOpen</code> <code>MsrGetStatus</code>	

MsrGetStatus

Purpose	Get status of last magnetic stripe read.	
Prototype	<code>Err MsrGetStatus (UInt refNum, BytePtr statusP)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>StatusP</code>	pointer to MSR status
Return	<code>MsrMgrNormal</code>	get status successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Gets MSR status of last swipe. Status byte format is as the following:	
	Bit5	1: track3 decoding was successful 0: error
	Bit4	1: track2 decoding was successful 0: error
	Bit3	1: track1 decoding was successful 0: error
	Bit2	1: track3 has sampling data 0: error
	Bit1	1: track2 has sampling data 0: error



```
1: track3 has sampling data
0: error
```

Example

```
error = MsrGetStatus(GmsrMgrLibRefNum, &status);
```

MsrSelfDiagnose

MsrSelfDiagnose

Purpose	Initiate MSR 3000 self test and return results.	
Prototype	<code>Err MsrSelfDiagnose (UInt refNum)</code>	
Parameters	<code>refNum</code>	library reference number
Return	<code>MsrMgrNormal</code>	self diagnose successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>MsrMgrErrROM</code>	ROM error.
	<code>MsrMgrErrRAM</code>	RAM error.
	<code>MsrMgrErrEEPROM</code>	EEPROM error.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Starts MSR self-diagnostic test and returns check result. It takes approximately 2 seconds.	
Example	<code>error = MsrSelfDiagnose(GMsrMgrLibRefNum);</code>	
See Also	<code>MsrGetStatus</code>	

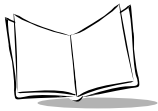


MsrSetBufferMode

Purpose	Set buffer mode of MSR 3000.	
Prototype	<code>Err MsrSetBufferMode (UInt refNum, Byte mode)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>mode</code>	buffer Mode to set MSR
Return	<code>MsrMgrNormal</code>	set buffer mode successful.
	<code>MsrMgrErrParam</code>	not allowed mode value.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Set Unbuffered Mode or Buffered Mode. Mode should be one of the following: <ul style="list-style-type: none">• <code>MsrBufferedMode</code>• <code>MsrUnbufferedMode</code>	
Example	<pre>error = MsrSetBufferMode(GmsrMgrLibRefNum, MsrUnbufferedMode);</pre>	
See Also	<code>MsrGetSetting</code>	

MsrArmtoRead

Purpose	Enable MSR to be ready for a card swipe in buffered mode.	
Prototype	<code>Err MsrArmtoRead (UInt refNum)</code>	
Parameters	<code>refNum</code>	library reference number
Return	<code>MsrMgrNormal</code>	arm to read successful.
	<code>MsrMgrErrParam</code>	mode neither Unbuffered nor Buffered mode.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Clears the buffer in the MSR and enables the MSR to be ready for a card swipe. This function is for the Buffered Mode only.	
Example	<code>error = MsrArmtoRead(GmsrMgrLibRefNum);</code>	
See Also	<code>MsrGetSetting</code>	

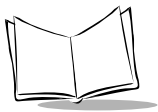


MsrSetTerminator

Purpose	Set terminator setting to MSR 3000.	
Prototype	<code>Err MsrSetTerminator (UInt refNum, Byte setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	terminator to set MSR
Return	<code>MsrMgrNormal</code>	set terminator successful.
	<code>MsrMgrErrParam</code>	not allowed setting value.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	The Terminator is a designated character which comes at the end of the last track of data, to separate card reads. This function set CR/LF, CR, LF or None as terminator.	
	The setting should be one of the following: <ul style="list-style-type: none">• <code>MsrTerminatorCRLF</code>• <code>MsrTerminatorCR</code>• <code>MsrTerminatorLF</code>• <code>MsrTerminatorNone</code>	
Example	<pre>error = MsrSetTerminator(GmsrMgrLibRefNum, MsrTerminatorCRLF);</pre>	
See Also	<code>MsrGetSetting</code>	

MsrSetPreamble

Purpose	Set a preamble to MSR 3000.	
Prototype	<code>Err MsrSetPreamble (UInt refNum, char *setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	preamble string to set MSR 3000 MSR.
Return	<code>MsrMgrNormal</code>	set preamble successful.
	<code>MsrMgrErrParam</code>	string too long.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets preamble string. String should less than MAX_PRE_POST_SIZE.	
Example	<code>error = MsrSetPreamble(GmsrMgrLibRefNum, "preamble\n");</code>	
See Also	<code>MsrGetSetting</code>	
	<code>MsrSetPostamble</code>	



MsrSetPostamble

Purpose	Set a postamble to MSR 3000.	
Prototype	<code>Err MsrSetPostamble (UInt refNum, char *setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	postamble string to set MSR 3000 MSR.
Return	<code>MsrMgrNormal</code>	set postamble successful.
	<code>MsrMgrErrParam</code>	string too long.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets postamble string. String should less than MAX_PRE_POST_SIZE.	
Example	<code>error = MsrSetPostamble(GmsrMgrLibRefNum, "postamble\n");</code>	
See Also	<code>MsrGetSetting</code>	

MsrSetTrackSelection

Purpose	Select tracks to be decoded to MSR 3000.	
Prototype	<code>Err MsrSetTrackSelection (UInt refNum, Byte setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	tracks information to get from MSR
Return	<code>MsrMgrNormal</code>	set track selection successful.
	<code>MsrMgrErrParam</code>	not allowed mode value.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets tracks to be decoded to the MSR 3000. Setting should be one of the following: <code>MsrAnyTrack</code> <code>MsrTrack1Only</code> <code>MsrTrack2Only</code> <code>MsrTrack3Only</code> <code>MsrTrack1Track2</code> <code>MsrTrack1Track3</code> <code>MsrTrack2Track3</code> <code>MsrAllThreeTracks</code>	
Example	<pre>error = MsrSetTrackSelection (GmsrMgrLibRefNum, MsrAnyTrack) ;</pre>	
See Also	<code>MsrGetSetting</code>	



MsrSetTrackSeparator

Purpose	Send track separator to MSR 3000.	
Prototype	<code>Err MsrSetTrackSeparator (UInt refNum, Byte setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	separator for tracks to set MSR
Return	<code>MsrMgrNormal</code>	set track separator successful.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Track separator is a designated character which separates data tracks.	
Example	<code>error = MsrSetTrackSeparator (GmsrMgrLibRefNum, '\n');</code>	
See Also	<code>MsrGetSetting</code>	

MsrSetLRC

Purpose	Send LRC mode to MSR 3000.	
Prototype	<code>Err MsrSetLRC (UInt refNum, Byte setting)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	LRC mode to set MSR
Return	<code>MsrMgrNormal</code>	set LRC successful.
	<code>MsrMgrErrParam</code>	added string too long.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	LRC is a check character following the end sentinel. The setting should be one of the following: <code>MsrSendLRC</code> <code>MsrNoLRC</code>	
Example	<code>error = MsrSetLRC (GmsrMgrLibRefNum, MsrNoLRC) ;</code>	
See Also	<code>MsrGetSetting</code>	

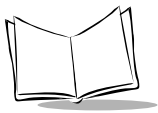


sMsrSetDataEdit

Purpose	Send data edit mode.	
Prototype	<code>Err MsrSetDataEdit (UInt refNum, Byte mode)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>mode</code>	Data edit mode to set MSR
Return	<code>MsrMgrNormal</code>	set data edit successful.
	<code>MsrMgrErrParam</code>	not allowed mode value.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets data edit mode. Mode should be one of the following: <code>MsrDisbaleDataEdit</code> <code>MsrDataEditMatch</code> <code>MsrDataEditUnmatch</code>	
Example	<pre>error = MsrSetDataEdit (GmsrMgrLibRefNum, MsrDisableDataEdit);</pre>	
See Also	<code>MsrGetSetting</code>	

MsrSetAddedField

Purpose	Set added fields to MSR 3000.	
Prototype	<pre>Err MsrSetAddedField (UInt refNum, char setting[MAX_AFLD_NUM] [MAX_AFLD_LEN+1])</pre>	
Parameters	refNum	library reference number
	setting	added fields strings to set MSR 3000 MSR.
Return	MsrMgrNormal	set added fields successful.
	MsrMgrErrParam	string too long.
	MsrMgrErrNAK	firmware NAK answer.
	MsrMgrErrRes	error response from MSR 3000.
	serErrTimeOut	handshake timeout.
	serErrBadPort	this port does not exist.
Comments	Set added fields strings for data edit. String should less than MAX_AFLD_LEN.	
Example	<pre>"fld1", "fld2\n", {0x0A, 0x00}, ""}; error = MsrSetAddedField(GmsrMgrLibRefNum, addedField);</pre>	
See Also	MsrGetSetting	



MsrSetDataEditSend

Purpose	Set data edit send commands to MSR 3000.	
Prototype	<pre>Err MsrSetDataEditSend (UInt refNum, char setting[MAX_SCMD_NUM] [MAX_SCMD_LEN])</pre>	
Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	send command setting for data edit.
Return	<code>MsrMgrNormal</code>	set added fields successful.
	<code>MsrMgrErrParam</code>	wrong send command.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	<p>Set data edit send commands.</p> <p><code>setting</code> is <code>[ccsmd] [dmvsmd][aamvasmd] [flexsmd]</code> <code>ccsmd</code> = {<code>field_len</code>}{Hex E0}{<code>field</code>}{<code>field</code>} {Hex FF}..., default is {Hex 00}{Hex FF} <code>dmvsmd</code> = {<code>field_len</code>}{Hex E1}{<code>field</code>}{<code>field</code>}{Hex FF}..., default is {Hex 00}{Hex FF} <code>aamvasmd</code> = {<code>field_len</code>}{Hex E2}{<code>field</code>}{<code>field</code>}{Hex FF}..., default is {Hex 00}{Hex FF} <code>flexsmd</code> = {<code>field_len</code>}{Hex E3}{<code>field</code>}{<code>field</code>}{Hex FF}..., default is {Hex 00}{Hex FF}</p> <p><code>field_len</code> is the number of bytes from {Hex Ex} to the {<code>field</code>} before {Hex FF}. <code>field</code> is a one byte field identifier. The highest three bits are used to identify the track number, and the lowest 5 bits is a unique field number.</p>	

track7,track6,track5	Track number
000	Added field number
001	Track1 field number
010	Track2 field number
011	Track3 field number
111	11111 (Successive field)
1xx	xxxxx (Reserved)

The valid field number for Credit Card on track1 and track2 is 0~9, 0~6. And there is no valid field number on track3.

The valid field number for California Driver License Card on each track is 0~7, 0~7 and 0~19.

The valid field number for AAMVA Card on each track is 0~6, 0~8 and 0~17.

The valid field number for added field is 0~5.

The valid field number for flexible field is 0~15.

The successive symbol 0xFF is a '~' symbol, so user need not write successive field one by one.

Example

```
char scmd [MAX_SCMD_NUM] [MAX_SCMD_LEN] = {
    {0x00, 0xFF},
    {0x00, 0xFF},
    {0x00, 0xFF},
    {0x00, 0xFF}};
error = MsrSetDataEditSend(GmsrMgrLibRefNum, scmd);
```

See Also

MsrGetSetting



MsrSetFlexibleField

Purpose	Set added fields to MSR 3000.	
Prototype	Err MsrSetFlexibleField (UInt refNum, char setting[MAX_FFLD_NUM] [MAX_FFLD_LEN])	
Parameters	refNum	library reference number
	setting	flexible field setting to set MSR 3000.
Return	MsrMgrNormal	set flexible fields successful.
	MsrMgrErrParam	wrong setting of flexible field.
	MsrMgrErrNAK	firmware NAK answer.
	MsrMgrErrRes	error response from MSR 3000.
	serErrTimeOut	handshake timeout.

serErrBadPort this port does not exist.

Comments

Sets flexible fields for data edit.

<Flexible_Field> is [length_match][string_match][search_before]
[search_between][search_after]...

length_match = {field_len} {hex F0} {track_no} {minimum length} {maximum length}

string_match = {field_len} {hex F1} {track_no}, {offset} {string_len} {string}

search_before = {field_len} {hex F2} {track_no}, {field_no} {times} {string_len} {string}

search_between = {field_len} {hex F3} {track_no} {field_no} {times1} {string1_len} {string1} {times2} {string2_len} {string2}

search_after = {field_len} {hex F4} {track_no} { field_no} {times1} {offset} {length2} {length1} {string1}

track_no = High three bits of the byte, 001*****|010*****|011*****

field_no = Low five bits of the byte, ***00000~***11111

field_len is the number of bytes from {Hex Fx} to the end of command.

field is a one byte field identifier. The highest three bits are used to identify the track number, and the lowest 5 bits is a unique field number.

If FlexFld[0][0] is 0, then MsrSetFlexibleField() means clear all flexible field.

Example

```
char flexFld [MAX_FFLD_NUM][MAX_FFLD_LEN] = {{0x00}};
error = MsrSetFlexibleField(GmsrMgrLibRefNum, flexFld);
```

See Also

MsrGetSetting



MsrGetDataBuffer

Purpose	Request card data information for the Buffered Mode.	
Prototype	<code>Err MsrGetDataBuffer (UInt refNum, MSRCardInfo_Ptr userCardInfoP, Byte get_Type)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>UserCardInfoP</code>	pointer to variable for user Card information
	<code>get_Type</code>	Type to get data
Return	<code>MsrMgrNormal</code>	No bad read and at least one good read on the selected track.
	<code>MsrMgrErrParam</code>	wrong setting of flexible field.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000, or call MsrGetDataBuffer without "Arm To Read" or "Have not swiped a card yet".
	<code>MsrMgrBadRead</code>	bad read on any one of selected track.
	<code>MsrMgrNoData</code>	no card data at all selected tracks
	<code>serErrTimeout</code>	handshake timeout.

`serErrBadPort` cradle port does not exist

Comments

Request card data Information. This function is for the Buffered Mode only. `get_Type` should be one of the following:

- `MsrGetAllTracks`
- `MsrGetTrack1`
- `MsrGetTrack2`
- `MsrGetTrack3`

Application should call `MsrArmtoRead()` first, before swiping a card and call `MsrGetDataBuffer()` to get the card information. `MsrReadMsrBuffer()` itself will issue a `MsrArmtoRead()`, wait a specified time for user to swipe a card and then get the card information. If the SPT goes to sleep before calling `MsrGetDataBuffer()`, the data buffer is lost.

Example

```
error = MsrGetDataBuffer(GmsrMgrLibRefNum, buff,
MsrGetAllTracks);
```

See Also

`MsrReadMSRBuffer`
`MsrReadMSRUnbuffer`



MsrReadMSRBuffer

Purpose	Request Card Information until receiving data or time out.	
Prototype	<code>Err MsrReadMSRBuffer (UInt refNum, MSRCardInfo_Ptr userCardInfoP, UInt waitTime)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>UserCardInfoP</code>	pointer to variable for user Card information
	<code>waitTime</code>	timeout in 100ms units
Return	<code>MsrMgrNormal</code>	No bad read and at least one good read on the selected track.
	<code>MsrMgrErrParam</code>	wrong setting of flexible field.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>MsrMgrBadRead</code>	Bad read on any one of selected tracks.
	<code>MsrMgrNoData</code>	No card data at all selected tracks.
	<code>serErrTimeOut</code>	handshake timeout.

`serErrBadPort` this port does not exist.

Comments

Issues “Arm to Read”, then requests card information until receiving data or time out. This function is for the Buffered Mode only.

Application should call `MsrArmtoRead()` first, before swipe a card and call `MsrGetDataBuffer()` to get the card information. `MsrReadMsrBuffer()` itself will issue a `MsrArmtoRead()`, wait a specified time for user to swipe a card and then get the card information.

Example

```
error = MsrReadMSRBuffer(GmsrMgrLibRefNum, buff, 20);  
// request and read card information, timeout is 2 seconds
```

See Also

`MsrGetDataBuffer`
`MsrReadMSRUnbuffer`



MsrReadMSRUnbuffer

Purpose	Request card data Information for the unbuffered mode.	
Prototype	<code>Err MsrReadMSRUnbuffer (UInt refNum, MSRCardInfo_Ptr userCardInfoP)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>UserCardInfoP</code>	pointer to variable for user card information
Return	<code>MsrMgrNormal</code>	No error occurred during get.
	<code>MsrMgrErrParam</code>	wrong setting of flexible field.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Requests card data information. This function is for the unbuffered mode only.	
Example	<code>error = MsrReadMSRUnbuffer(GmsrMgrLibRefNum, buff);</code>	
See Also	<code>MsrReadMSRBuffer</code>	
	<code>MsrReadMSRBuffer</code>	

MsrSetDecoderMode

Purpose	Set the decoder mode for the MSR 3000.	
Prototype	<code>Err MsrSetDecoderMode(UInt refNum, Byte mode)</code>	
Parameters	<code>refNum</code>	library reference number
	<code>mode</code>	decoder mode
Return	<code>MsrMgrNormal</code>	No error occurred during get.
	<code>MsrMgrErrParam</code>	wrong setting of flexible field.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.
Comments	Sets the decoder mode for the MSR 3000. Mode should be one of the following:	
	<code>MsrNormalDecoder</code>	Standard Decoder for ISO, DMV, AAMVA formats
	<code>MsrGenericDecoder</code>	Generic Decoder
	<code>MsrRawDataDecoder</code>	Raw Data Decoder
Example	<code>error = MsrSetDecoderMode(GmsrMgrLibRefNum, MsrNormalDecoder);</code>	
See Also	<code>MsrSetTrackFormat</code>	
	<code>MsrSetReservedChar</code>	



MsrSetTrackFormat

Purpose Set the parameters for the Generic Decoder, such as the Bit Format, Start and End Sentinel.

Prototype

```
Err MsrSetTrackFormat( Uint refNum, Byte track_ID,  
                      Byte format, Byte SS_Bits, Byte SS_ASCII,  
                      Byte ES_Bits, Byte ES_ASCII )
```

Parameters	refNum	library reference number
	track_ID	Track to be set
	format	Bit Format
	SS_Bits	The bit pattern of Start Sentinel for track 1
	SS-ASCII	The ASCII character of Start Sentinel for track 1
	ES-Bits	The bit pattern of End Sentinel for track 1
	ES-ASCII	The ASCII character of End Sentinel for track 1

Return	MsrMgrNormal	No error occurred during get.
	MsrMgrErrParam	wrong setting of flexible field.
	MsrMgrErrNAK	firmware NAK answer.
	MsrMgrErrRes	error response from MSR 3000.
	serErrTimeOut	handshake timeout.
	serErrBadPort	this port does not exist.

Comments Sets the parameters for the Generic Decoder, such as the Bit Format, Start and End Sentinel.
track_Id should be one of the following:

MsrTrack1Format	Track 1
MsrTrack2Format	Track 2
MsrTrack3Format	Track 3

The format should be one of the following:

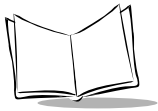
Msr5BitsFormat	5 bits with parity
Msr7BitsFormat	7 bits with parity

Example

```
error = MsrSetTrackFormat(GmsrMgrLibRefNum, MsrTrack1Format,  
                           Msr7BitsFormat, 0x51, '%', 0x7C, '?');
```

See Also

MsrSetDecoderMode
MsrSetReservedChar



MsrSetReservedChar

Purpose Set Special Reserved Characters for Generic Decoder only. Generic Decoder is based on ISO standard 5 or 7 bits encoded format. This command can be used to redefine the character in any position of ISO standard 5 or 7 bits coded character set.

Prototype `Err MsrSetReservedChar(Uint refNum, ReservedChar * setting)`

Parameters	<code>refNum</code>	library reference number
	<code>setting</code>	a set of reserved character to set

Return	<code>MsrMgrNormal</code>	No error occurred during setting.
	<code>MsrMgrErrParam</code>	wrong setting of flexible field.
	<code>MsrMgrErrNAK</code>	firmware NAK answer.
	<code>MsrMgrErrRes</code>	error response from MSR 3000.
	<code>serErrTimeOut</code>	handshake timeout.
	<code>serErrBadPort</code>	this port does not exist.

Comments Sets Special Reserved Characters.
Up to MAX_RES_CHAR_NUM characters can be set. Be sure to set format of last character to NULL.
The following is the structure definition for special reserved character.

```
Typedef struct ReservedChar {  
    Byte format;  
    char SR_Bits;  
    char SR_Chars;
```

```
} ReservedCharSetting;
```

format: bit format of a special reserved character

SR-Bits: The bit pattern of a special reserved character

SR-ASCII: The ASCII character of a special reserved character

Example

```
error = MsrSetReservedChar(GmsrMgrLibRefNum, setting);
```

See Also

MsrSetDecoderMode

MsrSetTrackFormat



Application Templates

The MSR Manager Shared Library is designed to be very easy to use. It also provides flexibility for developers who want to have maximum control. The following are some templates for different application needs.

If the developer needs an application to handle card data as soon as it arrives, then using Unbuffered mode is recommended.

Unbuffered Mode, Simple Application

The application typically follows this template:

1. MsrOpen()
2. Prompt user to swipe a card.
3. Call MsrReadMSRUnbuffer(), when a msrDataReadyKey keyDown event is received.
4. MsrClose()

If the developer needs an application to control the card swipe and card data read by itself, then using Buffered mode is recommended.

Buffered Mode, Simple Application

The application typically follows this template:

1. MsrOpen()
2. Prompt user to swipe a card.
3. MsrReadMSRBuffer(), if return code indicates No-Data, loop back to 3. or prompt user.
4. MsrClose()

Buffered Mode, Maximum Control

The application typically follows this template:

1. MsrOpen()
2. MsrArmtoRead()
3. Prompt user to swipe a card.
4. Perform other tasks until the application is ready to receive card data.
5. MsrGetDataBuffer(), if return code indicates No-Data, loop back to 5. or other process.
6. MsrClose().

MSR 3000 Configurator

Introduction

The MSR 3000 Configurator Program is a Windows application program. The Configurator provides an easy to use graphic user interface for selecting features and setup the MSR 3000. The Configurator generates a header file with all the settings. The MSR Manager Shared Library supports the application program with this header file included.

File menu commands

The File menu offers the following commands:

- | | |
|------|---------------------------------------|
| New | Creates a new configuration file. |
| Open | Opens an existing configuration file. |
| Exit | Exits MSR 3000 Configurator. |

View menu commands

The View menu offers the following commands:

- | | |
|------------|--------------------------------|
| Toolbar | Shows or hides the toolbar. |
| Status Bar | Shows or hides the status bar. |



Help Menu Commands

The Help menu offers the following commands, which provide assistance with this application:

Help Topics	Displays an index of help topics for selection.
About	Displays the version number of this application.

New Command (File Menu)

Use this command to create a new file in MSR 3000 Configurator.

Use the Open command to open an existing file.

Shortcuts

Toolbar:



Keys: CTRL+N




Open Command (File Menu)

Use this command to open an existing file in a new window.

Use the New command to create a new file.

Shortcuts

Toolbar: 

Keys: CTRL+O

File Open Dialog Box

The following options allow you to specify which file to open:

- File Name:** Type or select the filename you want to open. This box lists files with the extension you select in the List Files of Type box.
- List Files of Type:** Select the type of file you want to open:*.fig
- Drives:** Select the drive in which MSR 3000 Configurator stores the file that you want to open.
- Directories:** Select the directory in which MSR 3000 Configurator stores the file that you want to open.

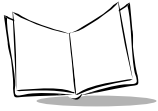
Exit Command (File Menu)

Use this command to end your MSR 3000 Configurator session. The MSR 3000 Configurator prompts you to save files with unsaved changes.

Shortcuts

Mouse: Double-click the MSR 3000 Configurator's Control menu button in the upper left-hand corner of the screen.

Keys: ALT+F4



Toolbar Command (View Menu)




Use this command to display and hide the Toolbar, which includes buttons for some of the most common commands in MSR 3000 Configurator, such as File Open. A check mark appears next to the toolbar menu item when the Toolbar is displayed.

See [Toolbar](#) on page 2-51 for help on using the toolbar.

Toolbar

The toolbar is displayed across the top of the MSR 3000 Configurator's window, below the menu bar. The toolbar provides quick mouse access to many tools used in MSR 3000 Configurator.

To hide or display the Toolbar, choose Toolbar from the View menu (ALT, V, T).

Click:	To:
	Create a new file
	Open an existing file.
	Open the Help Topics screen, which allows you to search for information about the Configurator.



Status Bar Command (View Menu)

Use this command to display and hide the Status Bar, which describes the action to be executed by the selected menu item or depressed toolbar button, and keyboard latch state. A check mark appears next to the menu item when the Status Bar is displayed.

See [Status Bar](#) on page 2-53 for help on using the status bar.

Status Bar

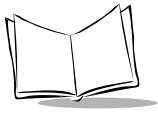


The status bar is displayed at the bottom of the MSR 3000 Configurator window. To display or hide the status bar, use the Status Bar command in the View menu.

The left area of the status bar describes actions of menu items as you use the arrow keys to navigate through menus. This area similarly shows messages that describe the actions of toolbar buttons as you press them, before releasing them. If, after viewing the description of the toolbar button command, you wish not to execute the command, then release the mouse button while the pointer is off the toolbar button.

The right areas of the status bar indicate which of the following keys are latched down:

Indicator	Description
CAP	The Caps Lock key is latched down.
NUM	The Num Lock key is latched down.
SCRL	The Scroll Lock key is latched down.



Help Topic Command (Help Menu)

Use this command to display the Help Index Contents screen. From the Index Contents screen, you can jump to step-by-step instructions for using MSR 3000 Configurator and various types of reference information.

Once you open Help, you can click the Contents button whenever you want to return to the Index screen. There are two other search options within the help application. You can search for information via an alphabetic index of topics by selecting the Index command, or you can enter search criteria using the Find command.

About Command (Help Menu)

Use this command to display the copyright notice and version number of your copy of MSR 3000 Configurator.



Context Help Command

Use the Context Help command to obtain help on the MSR 3000 Configurator. The Help topic for the item you selected displays.

Shortcut

Keys: SHIFT+F1

Title Bar

"MSR 3000 Configurator" displays in the title bar.

The title bar is located along the top of a window. It contains the name of the application and document.

To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

A title bar may contain the following elements:

- Application Control-menu button
- Document Control-menu button
- Maximize button
- Minimize button
- Name of the application
- Name of the document
- Restore button

Scroll Bars

Displayed at the right and bottom edges of the document window. The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document. You can use the mouse to scroll to other parts of the document.

Restore Command (Control menu)

Use this command to return the active window to its size and position before you chose the Maximize or Minimize command.



Move Command (Control Menu)

Use this command to display a four-headed arrow so you can move the active window or dialog box with the arrow keys.



Note: *This command is unavailable if you maximize the window.*

Shortcut

Keys: CTRL+F7

Size Command (Control Menu)

Use this command to display a four-headed arrow so you can size the active window with the arrow keys.



After the pointer changes to the four-headed arrow:

1. Press one of the DIRECTION keys (left, right, up, or down arrow key) to move the pointer to the border you want to move.
2. Press a DIRECTION key to move the border.
3. Press ENTER when the window is the size you want.

Note: *This command is unavailable if you maximize the window.*

Shortcut

Mouse: Drag the size bars at the corners or edges of the window.



Minimize Command (Control Menu)

Use this command to reduce the MSR 3000 Configurator window to an icon.

Shortcut

Mouse: Click the minimize icon  on the title bar.

Keys: ALT+F9

Maximize Command (Control Menu)

Use this command to enlarge the active window to fill the available space.

Shortcut

Mouse: Click the maximize icon  on the title bar; or double-click the title bar.

Keys: CTRL+F10 enlarges a document window.



Close Command (Control Menu)

Use this command to close the MSR 3000 Configurator.

Double-clicking the Control-menu box is the same as choosing the Close command.



Shortcuts

Keys: ALT+F4 closes the application window

Configurator Properties Buttons

There are four buttons which appear on each screen within the configurator.

Save Config Button

Use this button to save the active configuration file to its current name and directory. When you save a file for the first time, MSR 3000 Configurator displays the Save As dialog box so you can name your file.

File Save As dialog box

The following options allow you to specify the name and location of the file you're about to save:

File Name: Type a new filename to save a file with a different name. A filename can contain up to eight characters and an extension of up to three characters. MSR 3000 Configurator adds the extension you specify in the Save File As Type box.

Drives: Select the drive in which you want to store the document.

Directories: Select the directory in which you want to store the document.

Close Button

Use this button to close the MSR 3000 Configurator Properties dialog. If you didn't save, the current settings changed will lost.

Default All Button

Use this button to restore all configuration settings to their default values.

Help Button

Use this button to display detailed help information about the fields on the screen.

Using the Configurator to Set the MSR 3000

Introduction

The Configurator is a Windows GUI utility. It helps the application developer to do MSR 3000 configuration and be aware of MSR 3000 setting structure. The Configurator can



create a CodeWarrior include file for selected MSR 3000 setting, and the application developer can use the include file and call MsrSendSet() function to set up the MSR 3000.

MSR Setting Structure

```
typedef          struct      ReservedChar {

    Byte          format;

    char           SR_Bits;

    char           SR_Chars;

} ReservedChar;

Typedef    struct      MSR_Setting {

    Byte     Buffer_mode;

    Byte     Terminator;

    char     Preamble[MAX_PRE_POST_SIZE+1];

    char     Postamble[MAX_PRE_POST_SIZE+1];

    Byte     Track_selection;

    Byte     Track_separator;

    Byte     LRC_setting;

    Byte     Data_edit_setting;

    Byte     Decoder_mode;

    Byte     Track_format[MAX_TRACK_NUM][TRACK_FORMAT_LEN];

    ReservedChar     Reserved_chars[MAX_RES_CHAR_NUM];

    char     Added_field[MAX_AFLD_NUM][MAX_AFLD_LEN+1];

    char     Send_cmd[MAX_SCMD_NUM][MAX_SCMD_LEN];

    char     Flexible_field[MAX_FFLD_NUM][MAX_FFLD_LEN]

} MSR_Setting;
```

MSR Constant

#define	LF	0x0A
#define	CR	0x0D
#define	DC1	0X11
#define	DC3	0X13
#define	MsrUnbufferedMode	'0'
#define	MsrBufferedMode	'1'
#define	MsrArmtToReadMode	'0'
#define	MsrClearBufferMode	'1'
#define	MsrLEDOff	'0'
#define	MsrLEDOn	'1'
#define	MsrLEDBlink	'2'
#define	MsrTerminatorCRLF	'0'
#define	MsrTerminatorCR	'1'
#define	MsrTerminatorLF	'2'
#define	MsrTerminatorNone	'3'
#define	MsrAnyTrack	0
#define	MsrTrack1Only	1
#define	MsrTrack2Only	2
#define	MsrTrack1Track2	3
#define	MsrTrack3Only	4
#define	MsrTrack1Track3	5
#define	MsrTrack2Track3	6
#define	MsrAllThreeTracks	7



```
#define      MsrNoLRC           '0'

#define      MsrSendLRC         '1'

#define      MsrDisableDataEdit '0'

#define      MsrDataEditMatch   '1'

#define      MsrDataEditUnmatch '3'

#define      MsrGetAllTracks     '0'

#define      MsrGetTrack1       '1'

#define      MsrGetTrack2       '2'

#define      MsrGetTrack3       '3'
```

Include File Format Created by Configurator

```
MSRSetting user_MsrSetting = {

    MsrBufferedMode,

    MsrTerminatorCRLF,

    "Preamble\n",

    "Postamble\n",

    MsrAnyTrack,

    CR,

    MsrSendLRC,

    MsrDisableDataEdit,

    {"addf1","addf2","addf3", "w", "w", ""},

    {{0x08, 0xE0, ...},{0x10, 0xE1, ...},{0X06, 0XE2, },{NULL}},

    {{0XF0, ...},{0XF1, ...}...}

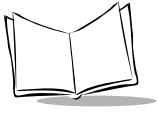
}
```

Generating an Include File by Configurator

1. In the File Menu, Click New to display the MSR Configurator Properties screen.
2. On the MSR Configurator Properties screen, select all fields, as described in [Appendix C, Data Editing Overview for Magnetic Stripe Reader](#).
3. Click the “Save Config” button to display a “Save As” dialog box.
4. Change directory to where the file should be saved.
5. Type the file name in the “File Name” edit box. The default extension of the file is “.h”.
6. Press the “Save” button to save the file or the “Cancel” button exit without saving.

Setting the MSR 3000 with the Configurator

1. Create a include file by Configurator to a specified setting, actually it define a static variable user_MsrSetting.
2. Include this include file in the application program.
3. Call SysLibFind(MsrMgrLibName, &GMrMgrLibRefNum) and/or SysLibLoad(MsrMgrLibTypeID, MsrMgrLibCreatorID, &GMrMgrLibRefNum) to load library and get library reference number GmsrMgrLibRefNum.
4. Call MsrOpen(GMrMgrLibRefNum, &msrVer, &libVer) to open MSR shared library.
5. Call MsrSendSetting(GMrMgrLibRefNum, user_MsrSetting) to set MSR 3000.
6. Call MsrClose(GmsrMgrLibRefNum) to close the library.
7. Call SysLibRemove(GMrMgrLibRefNum) to remove library from memory.



A Simple Application Program Sample

Include Files

The following `#include` statement provides MSR Manager interface definitions.

- `#include "MsrMgrLib.h"`

PilotMain Routine

The PilotMain function is a standard Palm organizer application. If the launch code is sysAppLaunchCmdNormalLaunch, the application program will do initialization in AppStart and an run event loop until it closes the application program. At this point, the application program will be terminated by AppStop.

```

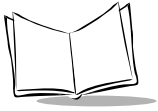
/*****
*
* FUNCTION: PilotMain
*
* DESCRIPTION: This is the main entry point for the application program.
*
* PARAMETERS: cmd - word value specifying the launch code.
*
*             cmdPB - pointer to a structure that is associated with the launch code.
*
*             launchFlags - word value providing extra information about the launch.
*
* RETURNED: Result of launch
*
* REVISION HISTORY:
*
*
*****/

static DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
    Err      error;

    FormPtr  frmP;

    switch (cmd)

```



```
{  
  
    case sysAppLaunchCmdNormalLaunch:  
  
        error = AppStart();  
  
        if (error) {  
  
            AppStop();  
  
            return error;  
  
        }  
  
  
  
        frmP = FrmInitForm(MSRInfoForm);  
  
        FrmDoDialog(frmP);  
  
        // Delete the info form.  
  
        FrmDeleteForm(frmP);  
  
  
  
        FrmGotoForm(MainForm);  
  
  
  
        AppEventLoop();  
  
        AppStop();  
  
  
  
    default:  
  
        break;  
  
  
    }  
  
    return 0;  
  
}
```

AppStart Function

The AppStart function shows what you need to do to initialize the MSR. This function:

- Loads the MSR Manager shared library.
- Powers on the MSR.
- Initiates communication between the application program and MSR 3000.
- Allocates a global handle for card data to display.

```

/*****
 *
 * FUNCTION:      AppStart
 *
 * DESCRIPTION:   Initialize Application program.
 *
 * PARAMETERS:   nothing
 *
 * RETURNED:      Err value 0 if nothing went wrong
 *
 * REVISION HISTORY:
 *
 *
 *****/

static Err AppStart(void)
{
    Err                error;

    VoidHand           gH;

    MSRLibGlobals_Ptr  gP;

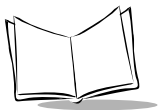
    SysLibTblEntryPtr  libEntryP;

    unsigned long      versionNo;

    // Load the MSR library

    error = SysLibFind(MsrMgrLibName, &GMrMgrLibRefNum);

```



```
if ( error) {

    // to load MSR library

    error = SysLibLoad(MsrMgrLibTypeID, MsrMgrLibCreatorID, &GMrMgrLibRefNum);

}

if ( error == MsrMgrNormal) {

    error = MsrOpen(GMrMgrLibRefNum, &versionNo);

    if (error != MsrMgrNormal) {

        FormPtrfrmP;

        frmP = FrmInitForm(NomSRForm);

        FrmDoDialog(frmP);

        // Delete the info form.

        FrmDeleteForm(frmP);

        //MsrClose(GMrMgrLibRefNum);

        return (error);

    }

};

// Allocate a new memory chunk that will contain received string.

newHandle = MemHandleNew(MAX_CARD_DATA);

return (MsrMgrNormal);

}
```

MainFormHandleEvent Function

PilotMain calls EventLoop after calling AppStart. The MainFormHandleEvent function is an event handler for main form. It deals with following events.

- FrmLoadEvent: Initialize and load Main Form
- FrmOpenEvent: Draw the main Form
- FrmCloseEvent: Close the main Form
- MenuEvent: Open the About Form
- KeyDownEvent with MsrDataReadyEve: call RS232_Receive to read card data and display it.

```

/*****
*
* FUNCTION:  MainFormHandleEvent
*
* DESCRIPTION: This routine is the event handler for the
*              "MainForm" of this application program.
*
* PARAMETERS: eventP - a pointer to an EventType structure
*
* RETURNED:   true if the event has handle and should not be passed
*              to a higher level handler.
*
* REVISION HISTORY:
*
*****/

```

```
static Boolean MainFormHandleEvent(EventPtr eventP)
```

```

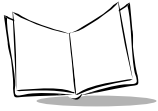
{
    Boolean    handled = false;

    FormPtr    frmP;

    Err        error;

    FieldPtr    fld;

```



```
switch (eventP->eType)
{
    case menuEvent:
        switch (eventP->data.menu.itemID)
        {
            case MainOptionsAboutMSRSample:
                // Load the info form, then display it.
                MenuEraseStatus (CurrentMenu);
                // Clear the Main form.
                frmP = FrmInitForm(MSRInfoForm);
                FrmDoDialog(frmP);
                // Delete the info form.
                FrmDeleteForm(frmP);
                handled = true;
                break;
        }

        break;

    case frmLoadEvent:
        FrmInitForm(MainForm);
        handled = true;
        break;
```

```
case frmOpenEvent:
```

```
    frmP = FrmGetActiveForm();
```

```
    FrmDrawForm(frmP);
```

```
    handled = true;
```

```
    break;
```

```
case frmCloseEvent:
```

```
    frmP = FrmGetActiveForm();
```

```
    fld = GetObjectPtr(MainCardInfoField);
```

```
    FldSetTextHandle(fld, NULL);
```

```
    frmP = FrmGetActiveForm();
```

```
    FrmEraseForm(frmP);
```

```
    FrmDeleteForm(frmP);
```

```
    handled = true;
```

```
    break;
```

```
case ctlSelectEvent: // A control button was pressed and released.
```

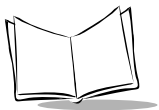
```
    break;
```

```
case keyDownEvent:
```

```
    if ((eventP->data.keyDown.chr==msrDataReadyKey)) {
```

```
        handled=true;
```

```
        error = MSR_Receive();
```



```
    }  
  
    break;  
  
default:  
  
    break;  
  
}  
  
return handled;  
  
}
```



```

/*****
*
* FUNCTION:  MSR_Receive
*
* DESCRIPTION: This routine is check RS232 Port
*              and display received data on DataInfo field
*
* PARAMETERS: frmP - a pointer to Main Form
*
* RETURNED:  true if no error occurred.
*
* REVISION HISTORY:
*
*****/

```

```
static UInt      MSR_Receive()
```

```
{
```

```

    FieldPtr          fld;

    Err               error;

    unsigned short     numReceived;

    CharPtr           newText;

```

```
// get Field pointer
```

```
    fld = GetObjectPtr(MainCardInfoField);
```

```
// RS232 receive
```

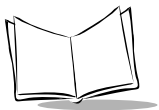
```
// get a card data time out is 500 ms
```

```
    error = MsrReadMSRUnbuffer( GMsrMgrLibRefNum, buff);
```

```
    if (error)
```

```
        return (error);
```

```
    numReceived = StrLen(buff);
```



```
if ( numReceived ){  
  
    // Lock down the handle and get a pointer to the memory chunk.  
  
    newText = MemHandleLock(newHandle);  
  
    // clear screen  
  
    *newText = NULL;  
  
    FldSetTextHandle(fld, newHandle);  
  
    FldDrawField(fld);  
  
    // Copy the data from the RS232 to the new memory chunk.  
  
    if (numReceived && (numReceived<MAX_CARD_DATA)) {  
  
        StrCopy(newText, buff);  
  
    }  
  
    // Unlock the new memory chunk.  
  
    MemHandleUnlock(newHandle);  
  
    // Set the field's text to the data in the new memory chunk.  
  
    FldSetTextHandle(fld, newHandle);  
  
    FldDrawField(fld);  
  
}  
  
return (MsrMgrNormal);  
  
}
```

AppStop Function

The AppStop function is called at the end of PilotMain. It

- Closes the MSR Manager shared library
- Powers off the MSR
- Frees the global handle for card data to display

```

/*****
*
* FUNCTION: AppStop
*
* DESCRIPTION: Save the current state of the application program.
*
* PARAMETERS: nothing
*
* RETURNED: nothing
*
* REVISION HISTORY:
*
*
*****/

static void AppStop(void)
{
    Err    error;

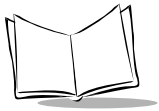
    // close MSR manager library

    error = MsrClose(GMsrMgrLibRefNum);

    // Uninstall the MSR Manager library

    if ( !GMsrMgrLibWasPreLoaded && GMsrMgrLibRefNum != sysInvalidRefNum )
    {
        error = SysLibRemove(GMsrMgrLibRefNum);
    }
}

```



SPT Terminal Series System Software Manual

```
ErrFatalDisplayIf(error, "error uninstalling MSR Manager library.");

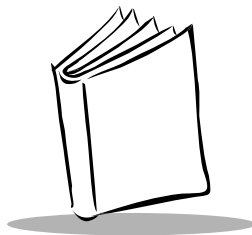
GMSrMgrLibRefNum = sysInvalidRefNum;

}


// Free a memory chunk
if (newHandle)

    MemHandleFree(newHandle);


}
```



Chapter 3

Printers for Palm Computing Platform

Introduction

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

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

Commercial Printers	Portable Label Printers
PCL	Comtec
PostScript	Monarch
	O'Neil
	Symbol
	Zebra

See [Appendix E, Supported Printers](#) for a list of supported printer names and models.



Application Programming Interface (API)

The Symbol printer API has four main features:

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

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

API Architectural Overview

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

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

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

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

<http://devzone.symbol.com>

Figure 3-1 on page 3-3 shows an architectural overview of the printer API and its associated printcap database.

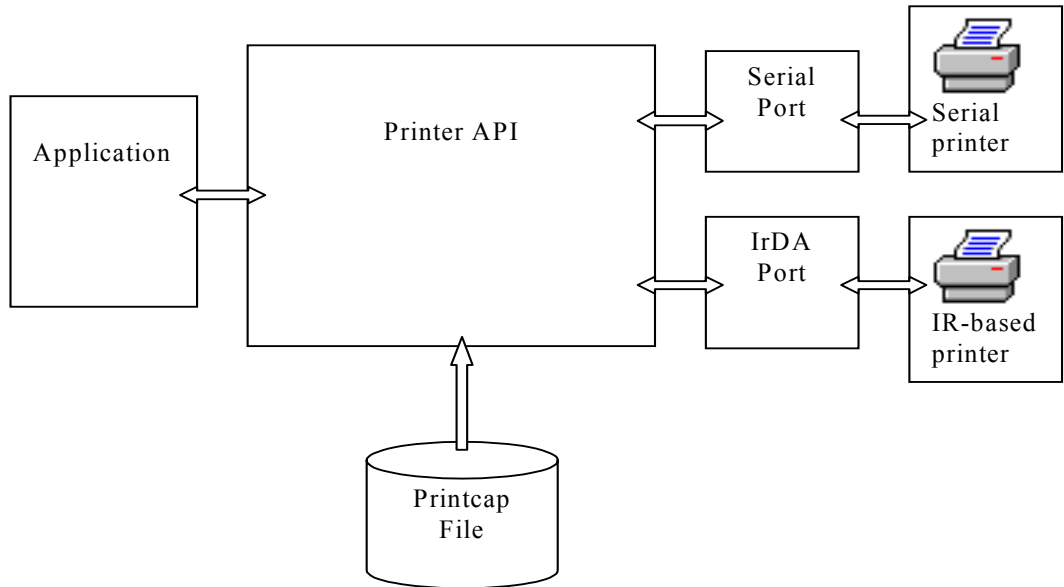


Figure 3-1. Printer API Interfaces

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

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

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



Section Descriptions

API Function Calls—Description of each function call in the API. These calls are divided into the following sections:

- [General Purpose Interface Functions](#)
- [High-level API Calls](#)
- [Lower-level API Calls](#)

Also included are definitions of the external data structures and enumerated types used by the Symbol Printer API.

Sample Application—An example application that uses some of the function calls available in the API.

System Requirements

Hardware requirements: Palm III or SPT

Software requirements: Palm OS 3.0 or greater
 HotSync 2.0 or greater

Conventions Used in this Manual

This guide uses the following typographical conventions.

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

API Function Calls

Introduction

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

Returned Status Definitions

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

STATUS CODE	DEFINITION
PTStatusAlreadyOpen	The printer is already open.
PTStatusAlreadyConnect	A connection has already been made to the printer.
PTStatusBadParameter	One of the parameters of the function call was incorrect.
PTStatusErrLine	When using the serial transport, a failure occurred during the SerReceive application call.
PTStatusFail	The function call failed.
PTStatusIrBindFailed	When using the IR transport, a failure occurred during the IrBind application call.
PTStatusIrConnectFailed	When using the IR transport, a failure occurred during the IrConnectReq application call.
PTStatusIrConnectLapFailed	When using the IR transport, a failure occurred during the IrConnectLap application call.
PTStatusIrDiscoverFail	When using the IR transport, a failure occurred during the IrDiscoveryReq application call.
PTStatusIrNoDeviceFound	When using the IR transport, no device was found during the IrDiscoveryReq application call.
PTStatusIrQueryFailed	When using the IR transport, a failure occurred during the IrIASQuery application call.
PTStatusNoMemory	No dynamic heap space is available to hold library data.
PTStatusNotOpen	Tried to use or close a port that was not open.



STATUS CODE	DEFINITION
PTStatusOK	The function call was successfully executed.
PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.
PTStatusPrinterNotFound	The target printer is not defined in the printcap database.
PTStatusRomIncompatible	The Palm OS version must be 3.0 or greater.
PTStatusTimeOut	A timeout occurred while waiting for data from the printer.
PTStatusTransportNotAvail	The requested transport (serial port, IrDa) is not available.

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

Print Commands

The calls in the API provide three types of functions:

FUNCTION TYPE	PURPOSE	PAGE
General Purpose Interface Functions	Control the printer's operation.	3-7
High-level API Calls	Print text and simple graphics.	3-18
Lower-level API Calls	Retrieve printcap entries and send user-formatted data to the printer.	3-26

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

Data Structures	3-29
------------------------	------

General Purpose Interface Functions

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

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

The calls included in this section are:

FUNCTION	PAGE
<i>ptClosePrinter</i>	3-8
<i>ptConnectPrinter</i>	3-9
<i>ptDisconnectPrinter</i>	3-11
<i>ptInitPrinter</i>	3-12
<i>ptOpenPrinter</i>	3-13
<i>ptPrintApiVersion</i>	3-15
<i>ptQueryPrinter</i>	3-16
<i>ptResetPrinter</i>	3-17



ptClosePrinter

Purpose Cleans up and closes the shared library.

Prototype `PTStatus ptClosePrinter ();`

Returned Status `PTStatusOK` The function call was successfully executed.

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

Comments Call `ptClosePrinter` after the printing is completed.

See Also [*ptOpenPrinter*](#)

ptConnectPrinter

Purpose	Establishes a connection to the printer using the serial port or an IR connection.	
Prototype	<code>PTStatus ptConnectPrinter (CharPtr printerName);</code>	
Parameters	<code>-> printerName</code>	Name of printer; used for future transports other than serial or IR.
Returned Status	<code>PTStatusOK</code>	The function call was successfully executed.
	<code>PTStatusNotOpen</code>	Tried to use or close a port that was not open.
	<code>PTStatusBadParameter</code>	Serial only. One of the parameters of the function call was incorrect.
	<code>PTStatusFail</code>	IrDa only. The function call failed.
	<code>PTStatusTransportNotAvail</code>	The requested transport (serial port, IrDa) is not available.
	<code>PTStatusAlreadyConnect</code>	A connection has already been made to the printer.
	<code>PTStatusNoMemory</code>	No dynamic heap space is available to hold library data.
	<code>PTStatusIrConnectFailed</code>	When using the IR transport, a failure occurred during the IrConnectReq application call.
	<code>PTStatusTimeOut</code>	IrDa only. A timeout occurred while waiting for data from the printer.



<code>PTStatusIrNoDeviceFound</code>	IrDa only. When using the IR transport, no device was found during the <code>IrDiscoveryReq</code> application call.
<code>PTStatusIrDiscoverFail</code>	IrDa only. When using the IR transport, a failure occurred during the <code>IrDiscoveryReq</code> application call.
<code>PTStatusIrConnectFailed</code>	IrDa only. When using the IR transport, a failure occurred during the <code>IrConnectReq</code> application call.
<code>PTStatusIrConnectLapFailed</code>	IrDa only. When using the IR transport, a failure occurred during the <code>IrConnectLap</code> application call.
<code>PTStatusIrQueryFailed</code>	IrDa only. When using the IR transport, a failure occurred during the <code>IrIASQuery</code> application call.

Comments

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

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

See Also

[*ptClosePrinter*](#)

ptDisconnectPrinter

Purpose	Disconnects the printer.	
Prototype	PTStatus ptDisconnectPrinter ();	
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusNotOpen	Tried to use or close a port that was not open.
Comments	<p>This function call closes the port for the serial transport. If the printer is connected through IR, the IR connection is disconnected.</p> <p>To save battery power on the handheld device, use this function call to disconnect the printer after your application has successfully printed the data. This is especially important if your application is using the serial port. However, you do not need to close the printer until the application is finished.</p>	
See Also	<i>ptConnectPrinter</i> <i>ptClosePrinter</i>	



ptInitPrinter

Purpose Initializes the printer.

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

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

`-> length` Length of initialization string.

Returned Status `PTStatusOK` The function call was successfully executed.

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

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

See Also [*ptResetPrinter*](#)

ptOpenPrinter

Purpose	Opens and initializes the shared library.	
Prototype	<pre>PTStatus ptOpenPrinter(CharPtr printerModel, PTTransports transport, PTConnectSettingsPtr connectSettings);</pre>	
Parameters	-> printerModel	Model of the printer to be opened. References printcap information.
	-> transport	The transport type.
	-> connectSettings	Contains information for baud rate, parity settings, stop bits, and handshaking. This information can overwrite connection settings in the printcap database.
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusAlreadyOpen	The printer is already open.
	PTStatusRomIncompatible	The Palm OS version must be 3.0 or greater.
	PTStatusNoMemory	No dynamic heap space is available to hold library data.
	PTStatusPrinterNotFound	The target printer is not defined in the printcap database.



<code>PTStatusIrBindFailed</code>	IrDa only. When using the IR transport, a failure occurred during the IrBind application call.
-----------------------------------	--

<code>PTStatusTimeOut</code>	IrDa only. A timeout occurred while waiting for data from the printer.
------------------------------	--

Comments	To use the printcap database default connection settings, set connectSettings to NULL. If the printer is connected using IR, only the baud rate applies. The ptConnectSettings structure is documented in the Data Structures section at the end of this chapter. It is also included in the ptPrint.h file.
-----------------	--

See Also	ptClosePrinter
-----------------	--------------------------------

ptPrintApiVersion

Purpose	Returns the version number of the Printer Library.	
Prototype	<code>PTStatus ptPrintApiVersion (CharPtr ptr, Int len);</code>	
Parameters	<code>-> ptr</code>	Pointer to the character buffer in which the version number will be placed.
	<code>-> len</code>	Size of the character buffer.
Returned Status	<code>PTStatusOK</code>	The function call was successfully executed.
	<code>PTStatusNotOpen</code>	The application must first call ptOpenPrinter before using this function.
	<code>PTStatusFail</code>	The ptr parameter is invalid.
Comments	<p>In version 1.0, the buffer must be 12 characters long, as the return string is "ptPrint 1.0."</p> <p>You must open the printer library before making this function call. You can close the library after the call.</p>	

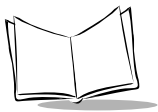


ptQueryPrinter

Purpose	Queries the printer condition.	
Prototype	<pre>PTStatus ptQueryPrinter (VoidPtr queryPtr, ULong length, VoidPtr queryResPtr, uLong queryResLen);</pre>	
Parameters	-> queryPtr	Pointer to the memory used to send the query to the printer. If it is NULL, the “qs” (query string) value from the printcap database entry for this printer is used.
	-> length	Length of query data to be sent to the printer.
	-> queryResPtr	Pointer to the memory used to hold any response from the query. This buffer is provided and released by the application.
	-> queryResLen	Length of the query response buffer pointed to by queryResPtr.
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusErrLine	Serial only. When using the serial transport, a failure occurred during the SerReceive application call.
	PTStatusNotOpen	Tried to use or close a port that was not open.
	PTStatusTimeOut	IrDa only. A timeout occurred while waiting for data from the printer.
Comments	<p>The data referenced by queryPtr is a set of hexadecimal bytes that represents the printer-specific query command. The data is sent to the printer to query its status. The response is then placed into the memory allocated by the application, pointed to by queryResPtr.</p> <p>An error is returned to the calling application if a printcap entry for the destination printer does not exist and the queryPtr argument is NULL.</p>	

ptResetPrinter

Purpose	Resets the printer.	
Prototype	PTStatus ptResetPrinter (VoidPtr resetPtr, ULong length);	
Parameters	-> resetPtr	Pointer to data that is sent to the printer; this data is used to reset the printer. If it is NULL, the "rs" (reset string) value from the printcap database entry for this printer is used.
	-> length	Length of reset string.
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusNotOpen	Tried to use or close a port that was not open.
Comments	The reset string is a set of hexadecimal bytes sent to the printer to reset it. An error is returned if a printcap entry does not exist for the destination printer and the resetPtr argument is NULL.	
See Also	<i>ptInitPrinter</i>	



High-level API Calls

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

- Text
- Simple graphics

The calls included in this section are:

FUNCTION	PAGE
<i>ptLineToBuffer</i>	3-19
<i>ptPrintPrintBuffer</i>	3-20
<i>ptRectToBuffer</i>	3-21
<i>ptResetPrintBuffer</i>	3-22
<i>ptSetFont</i>	3-23
<i>ptStartPrintBuffer</i>	3-24
<i>ptTextToBuffer</i>	3-25

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

ptLineToBuffer

Purpose	Adds the print line command to the buffer.	
Prototype	<pre>PTStatus ptLineToBuffer (UInt xStart, UInt yStart, UInt xEnd, UInt yEnd, UInt lineThickness);</pre>	
Parameters	-> xStart	X-coordinate of line start point.
	-> yStart	Y-coordinate of line start point.
	-> xEnd	X-coordinate of line end point.
	-> yEnd	Y-coordinate of line end point.
	-> lineThickness	Thickness of the line to be drawn (dots).
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusFail	The function call failed.
	PTStatusNoMemory	No dynamic heap space is available to hold library data.
	PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.
Comments	<p>ptLineToBuffer determines which command sequence from the printcap database is needed to print a simple line on a specific printer. It adds the command to the allocated buffer and includes logic to expand the buffer size if it is not large enough to hold the command and the line.</p> <p>Refer to the printer documentation for a definition of the start and end points.</p>	
See Also	<i>ptTextToBuffer</i> <i>ptRectToBuffer</i>	



ptPrintPrintBuffer

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

Prototype `PTStatus ptPrintPrintBuffer (CharPtr printerName);`

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

Returned Status

<code>PTStatusOK</code>	The function call was successfully executed.
<code>PTStatusFail</code>	The function call failed.
<code>PTStatusNoMemory</code>	No dynamic heap space is available to hold library data.
<code>PTStatusPrintCapFailed</code>	The function call tried to access a printcap entry that doesn't exist.

Comments

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

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

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

ptRectToBuffer

Purpose	Adds the print rectangle command to the buffer.	
Prototype	<pre>PTStatus ptRectToBuffer (UInt xTopLeft, UInt yTopLeft, UInt xBottomRight, UInt yBottomRight, UInt lineThickness);</pre>	
Parameters	-> xTopLeft	X-coordinate of rectangle top left point.
	-> yTopLeft	Y-coordinate of rectangle top left point.
	-> xBottomRight	X-coordinate of rectangle bottom right point.
	-> yBottomRight	Y-coordinate of rectangle bottom right point.
	-> lineThickness	Thickness of the lines of the rectangle (dots).
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusFail	The function call failed.
	PTStatusNoMemory	No dynamic heap space is available to hold library data.
	PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.
Comments	ptRectToBuffer determines which command sequence from the printcap database is needed to print a rectangle on a specific printer. It adds the command to the allocated buffer and includes logic to expand the buffer size if it is not large enough to hold the command and the rectangle.	
See Also	ptLineToBuffer ptTextToBuffer	



ptResetPrintBuffer

Purpose	Frees the print buffer memory.	
Prototype	<code>PTStatus ptResetPrintBuffer ()</code>	
Returned Status	<code>PTStatusOK</code>	The function call was successfully executed.
	<code>PTStatusFail</code>	The function call failed.
	<code>PTStatusNoMemory</code>	No dynamic heap space is available to hold library data.
	<code>PTStatusPrintCapFailed</code>	The function call tried to access a printcap entry that doesn't exist.
Comments	This function call performs the same function as ptPrintPrintBuffer , except it doesn't print the data in the print buffer.	

ptSetFont

Purpose	Sets the printer's typeface and type size.	
Prototype	<code>PTStatus ptSetFont (CharPtr fontBuffPtr);</code>	
Parameters	<code>-> fontBuffPtr</code>	Contains information about the font the user would like to set.
Returned Status	<code>PTStatusOK</code>	The function call was successfully executed.
	<code>PTStatusFail</code>	The function call failed.
	<code>PTStatusNoMemory</code>	No dynamic heap space is available to hold library data.
	<code>PTStatusPrintCapFailed</code>	The function call tried to access a printcap entry that doesn't exist.
Comments	<p>The format of fontBuffPtr is:;;</p> <p>The printer API parses the fontBuffPtr string to get the font type, width, and length. When text is printed, the API substitutes the variable data for the font. If a printer doesn't require the font width, you can leave that variable blank by providing only ;</p> <p>You must include both semicolons between and so the printer API recognizes that the variable is missing.</p>	



ptStartPrintBuffer

Purpose Sets the printer library's initial buffer size.

Prototype `PTStatus ptStartPrintBuffer (ULong size);`

Parameters `-> size` Length of the initial buffer.

Returned Status `PTStatusOK` The function call was successfully executed.

`PTStatusFail` The function call failed.

`PTStatusNoMemory` No dynamic heap space is available to hold library data.

`PTStatusPrintCapFailed` The function call tried to access a printcap entry that doesn't exist.

See Also [*ptTextToBuffer*](#)
 [*ptLineToBuffer*](#)
 [*ptRectToBuffer*](#)
 [*ptPrintPrintBuffer*](#)

ptTextToBuffer

Purpose	Adds the print text command to the buffer.	
Prototype	<pre>PTStatus ptTextToBuffer (UInt xStart, UInt y Start, CharPtr pText);</pre>	
Parameters	-> xStart	Beginning X-coordinate of print data.
	-> yStart	Beginning Y-coordinate of print data.
	-> pText	Pointer to text that will be printed.
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusFail	The function call failed.
	PTStatusNoMemory	No dynamic heap space is available to hold library data.
	PTStatusPrintCapFailed	The function call tried to access a printcap entry that doesn't exist.
Comments	<p>ptTextToBuffer determines which command sequence from the printcap database is needed to print text on a specific printer. It adds the command and the text to the allocated buffer. ptTextToBuffer includes logic to expand the size of the buffer if it is not large enough to hold the command and the text.</p>	
See Also	<i>ptLineToBuffer</i> <i>ptRectToBuffer</i>	



Lower-level API Calls

The lower-level API calls do the following:

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

The calls included in this section are:

FUNCTION	PAGE
<i>ptQueryPrintCap</i>	3-27
<i>ptWritePrinter</i>	3-28

ptQueryPrintCap

Purpose	Queries the printcap database for the given token.	
Prototype	<pre>PTStatus ptQueryPrintCap (CharPtr query, VoidPtr queryResPtr, ULong queryResLen);</pre>	
Parameters	-> query	Pointer to the memory used to find the value in the printcap entry for the open printer.
	-> queryResPtr	Pointer to the memory used to hold any response from the query.
	-> queryResLen	Maximum length of the query response.
Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusNotOpen	Tried to use or close a port that was not open.
Comments	<p>After querying the printcap database, the queryResPtr buffer is filled in with the query results. Some special characters will be represented by hexadecimal bytes. For example, ESC will be converted to the hexadecimal value 0x1B. Other escape character conversions are also supported, such as ALT, CTRL, Newline, and Carriage Return.</p>	



ptWritePrinter

Purpose Writes data to the printer.

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

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

Returned Status	PTStatusOK	The function call was successfully executed.
	PTStatusNotOpen	Tried to use or close a port that was not open.
	PTStatusTimeOut	A timeout occurred while waiting for data from the printer.

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

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

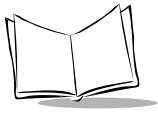
The `ptWritePrinter` function call can be used to print barcodes, text, and simple graphics on legacy printers, which rely on forms.

Contact the printer manufacturer for proper formatting for printer commands.

Data Structures

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

EXTERNAL DATA STRUCTURE	PAGE
<i>Printcap Database</i>	3-30
<i>PTConnectSettings</i>	3-30
<i>PTStatus</i>	3-31
<i>PTTransports</i>	3-31



Printcap Database

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

<http://devzone.symbol.com>

PTConnectSettings

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

```
typedef struct tagPTConnectSettings {  
  
    ULONG    baudRate;           // baud rate  
    ULONG    timeOut;           // time out in System Ticks  
    ULONG    flags;             // transport flags; see SerialMgr.h  
    UINT     recvBufSize;       // receive buffer size  
  
} PTConnectSettings;  
typedef PTConnectSettings * PTConnectSettingsPtr;
```

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

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

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

PTStatus

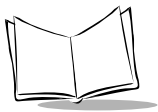
PTStatus represents the possible status values from the printer API to the application.

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

PTTransports

PTTransports represents the possible transports.

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



Sample Application

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

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

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

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

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

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

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

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

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

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

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

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

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

Code Samples

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

Opening the Print Library

The first step in using the printer API is to open and initialize the library; this is done with the *ptOpenPrinter* function call. In this call, you must specify the printer model name and the transport mechanism. Below is an example of the code needed for the Comtec RP3 printer.



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

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

```
Err error;
Boolean retval = FALSE;
PTConnectSettings cSettings;
// specify connection settings
cSettings.baudRate = PTDefaultSerBaudRate;
cSettings.timeOut = PTDefaultSerTimeout;
cSettings.flags = PTDefaultSerFlags;
cSettings.recvBufSize = PTDefaultSerRecvBuf;
// connect to serial printer
error = ptOpenPrinter( "Unknown", PTSerial, &cSettings );
if (error != PTStatusOK)
{
    ErrNonFatalDisplayIf(error,
        "Cannot initialize print library");
    return retval;
}
```

Similarly, if you are printing with a PostScript printer, specify it as your printer model name in the [ptOpenPrinter](#) function call.

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

```

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

```

Connecting to the Printer

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

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

```

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

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

```

Querying the Printer

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

```

#define StatusBuffLen 256

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

```



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

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

```
#define StatusBuffLen 256  
  
Err error;  
Boolean retval = FALSE;  
CharPtr queryString = "QUERY";                                // made-up query string  
                                                         // for Unknown printer  
  
Byte statusBuffer[StatusBuffLen ];  
  
error = ptQueryPrinter( queryString,  
                        (ULong) ( StrLen(queryString) ),  
                        &statusBuffer,  
                        (ULong) StatusBuffLen );  
  
if (error != PTStatusOK)  
{  
    ErrNonFatalDisplayIf(error, "Could not query the printer");  
    ptClosePrinter();  
    return retval;  
}
```

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

- [*ptQueryPrinter*](#)
- [*ptInitPrinter*](#)
- [*ptResetPrinter*](#)

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

Writing Data to the Printer

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

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

Using the Low-Level API

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

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

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



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

Using the High-Level API

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

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

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

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

```

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

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

```

Disconnecting the Printer and Closing the Library

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

```

Err error;

// Disconnect the printer
error = ptDisconnectPrinter( );

// Close the printer
error = ptClosePrinter();

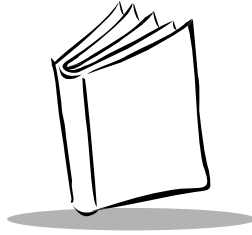
```

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

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



SPT Terminal Series System Software Manual



Chapter 4

Spectrum24

Introduction

This chapter provides an overview of Spectrum24® wireless operation, as well as information about the Spectrum24 programming interface.

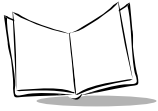
Caution

The API detailed in this chapter is supported by Palm OS v4.1 Symbol Terminals only.

ESSID and BSSID

ESSID and BSSID are names used by Spectrum24 terminal (mobile Spectrum24® units) radios to connect with 802.11 protocol enabled access points using the same names (the Spectrum24 terminal is 802.11 enabled). ESSID is the local radio network's name and is used by both access points and Spectrum24 terminals to identify members of the network. An ESSID name is mandatory for 802.11 operation. Only units that have the proper ESSID are allowed on the network. ESSID is valid throughout the RF network, so Spectrum24 terminal's that roam can stay connected to the network throughout an installation.

Note: *ESSID is a character string of up to 32 bytes in length. Do not programmatically set a string to non-printable characters, because it will be hard to decode that string later if it is forgotten.*



BSSID is the 6-byte MAC address of a particular access point. BSSID can limit network accessibility to specific access points in the attempt to load balance a LAN, or to keep specific operations in one area. Note that each Spectrum24 terminal also has its own BSSID.

Wired and Wireless Network Connections

At the highest level of network communications, applications exchange data messages. These messages are often sent from one computer to another, along with system control messages. The transfer medium can be a physical connection (wire, fiber), open-air (radio, IR), or a combination of the two.

Every device in a totally wired network can always communicate with all other devices in the network, as long as the devices are turned on and the wiring connecting them is intact. Most standard communication protocol stacks assume that the messages always get through because they are written to depend on the intact wired connection. When a message is sent, the sending device starts a time-out counter and waits for a reply from the receiving device. If the time-out expires, the original message can be retransmitted and the time-out restarted. This retransmit process could continue indefinitely, but it is usually performed fewer than five times. If the last retransmit times out, an error condition is returned to the application indicating that the connection was lost. At that point, an error message is usually displayed and the application is aborted because the wire is assumed to be broken.

In a wireless network, where devices can be moved around, the device may not be able to maintain the connection to the network. This could be a normal occurrence, as when a device is taken from one building to another and is temporarily out of the network's range. Ideally, the connection between the application in the Spectrum24 terminal and the application in the network's host computer remains intact. When the terminal gets back in range, the operation should resume. However, if the protocol stack (written for a wired network) times out because it does not get an ACK message from the receiving device, the connection is usually assumed to be broken; an error code is returned to the application, and the application is aborted. In a wireless environment, this should not happen. Software in a wireless environment should at least detect an out-of-range condition and not push messages onto the transport layer until the Spectrum24 terminal is back in range. If detection is not possible, applications should be designed to either recover from the out-of-range condition (rather than abort) or use transport layers that are tolerant of a protracted inability to communicate. If tolerant transport layers are not available, set the time-out or retransmit parameters to their maximum so that the occasional out-of-range condition does not abort the application as soon as this condition starts.

Figure 1-1 shows how an application interfaces with the Spectrum24 terminal's PalmOS NetLib API functions.

Note: *NetLib is a wrapper around the IP stack and is used by Spectrum24 terminal applications to send and receive data. For a detailed description of NetLib, see the standard Palm Programming documentation.*

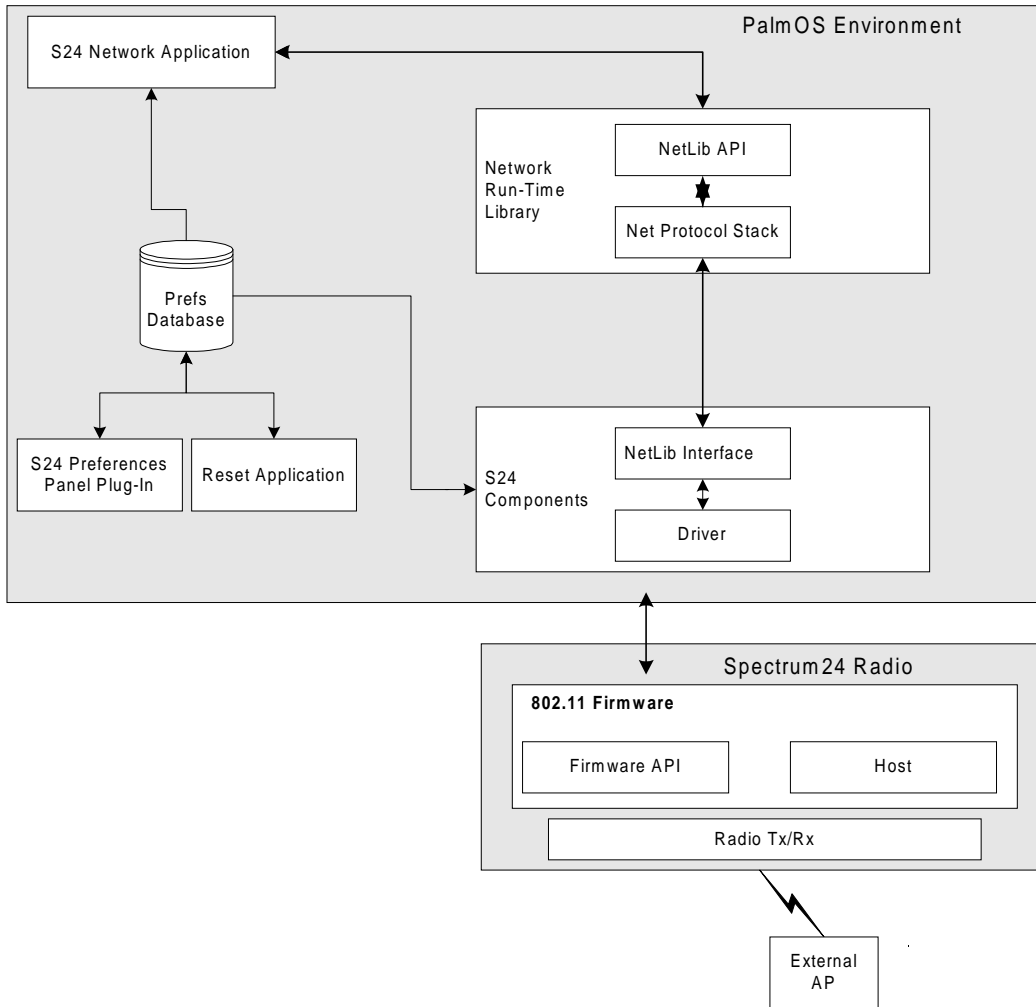
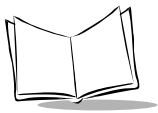


Figure 4-1. Spectrum24 Interfaces

Spectrum24/NetLib Design and Implementation Considerations

Feature Limitations for this Model

The following list describes features not supported:

- Spectrum24 terminals support only the IEEE 802.11 protocol.
- 802.1 header message formats are not supported.
- Mobile IP is not supported.
- Multicast messages are not supported.

General Application Design Considerations

Applications running on a local radio network (LRN) must be able to handle the following adverse conditions:

- Long delays caused by congestion (other Spectrum24 communications) of the LRN and by the Spectrum24 terminal roaming in and out of the LRN's range
- Long delays in the wired network responses
- High TCP retry timeouts, which might cause the socket to close and lose the network connection
- Slow connection or re-connection to the host application server
- RF transmissions being locked out by aggressive user scanning operations.

Turning the Spectrum24 Radio Interface On and Off

When an application calls the NetLibOpen command (see the standard Palm Programming documentation), NetLib interfaces to the Spectrum24 Radio Driver which powers up the radio, and initializes and configures the radio for RF transmissions. The Spectrum24 driver powers off the radio when an application calls the NetLibClose command.

NetLib Interface UI

When initializing, the network interface on the Spectrum24 terminal displays three windows that contain the following status information about the RF network connection:

- Associating with ESSID <ssid>
- Binding (getting an IP address from the DHCP server)
- Connected



One key status indicator is association with an access point. Your application can determine through a Spectrum24 API ([netIFSettingS24AssociationStatus](#) on page 4-20) whether or not the Spectrum24 terminal is associated to an access point. Ideally, your application should display a dialog or other indication when the Spectrum24 terminal radio loses association with an AP for a period of time.

Note: *Situations where the UI does not go beyond the associating stage usually imply a failure to connect to an access point. Stopping at the binding stage indicates a problem negotiating an IP address through the DHCP process.*

Roaming/Unassociation Condition

A Spectrum24 terminal unassociates with an access point when the Spectrum24 terminal radio is out of range or is in the process of associating with another access point. If an application tries to transfer data while the radio is unassociated with an access point, the data and perhaps the network connection may be lost. The application should not attempt to transmit data until the association is reacquired.

Applications can use a Spectrum24 API ([netIFSettingS24AssociationStatus](#) on page 4-20) to get the current AP association status before sending data through NetLib calls.

Note: *A user interface should display if the Spectrum24 terminal is unassociated for more than a few seconds to notify the operator to return to the network and display the reason the application is on hold. Do not display the out-of-range UI on the first unassociation; give the terminal a chance to re-associate.*

Power Management

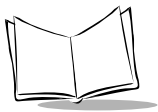
Power management functionality is integrated into the radio driver and network interface. The power management implementation integrates features provided in the Spectrum24 terminal Palm OS (auto-off timer, power key on/off, dead battery level, and low battery level), Spectrum24 radio firmware (sleep, host power-down), and the Spectrum24 hardware interface controller application-specific integrated circuit (ASIC) (VCC on/off).

To maximize battery life on the Spectrum24 terminal, the Spectrum24 driver automatically tries to keep the radio operating in the most energy efficient manner.

Note: *It is the application's responsibility to re-establish the connections to the host. Socket connections must be re-established through NetLib using the NetLibConnectionRefresh() function call.*

NetLibConnectionRefresh()

Power-off conditions cause the MAC-layer (radio) to go down while NetLib stays open. Calls to NetLibConnectionRefresh() ensure that the MAC-layer is up. If an application has not been closed and NetLib is still open, the application should call NetLibConnectionRefresh() before each socket operation and Spectrum24 API call. This causes the Spectrum24 driver to either restart or resume the Spectrum24 terminal radio. It also causes reassociation to an AP.



Spectrum24 Radio API

Caution

This API is supported by Palm OS v4.1 Symbol Terminals only.

The Programmer's Interface

This section describes the Spectrum24 Radio API that developers can use for Spectrum24 radio terminals. Using this API, an application can control some of the Spectrum24 terminal's radio behavior and gather information about its radio settings. The API lets the application configure such items as the radio's ESSID string, and read items such as the current radio association status.

Spectrum24 is a network interface that abstracts the low-level networking protocols, much like SLIP or PPP. In Palm OS, the network library configuration is structured so that network interface specific settings, called IF settings, can be specified for each network interface independently. `NetLibIFSettingGet()` and `NetLibIFSettingSet()` calls set and retrieve these IF settings.

The Spectrum24 Radio API is a set of Spectrum24 network interface specific settings, such as the radio's ESSID string ([netLibSettingS24EssID](#) on page 4-16) or the current radio association status with an access point ([netLibSettingS24AssociationStatus](#) on page 4-20).

To use the Spectrum24 Radio API in your application:

1. Include the `S24API.h` header file in your application.
2. Obtain the PalmOS net library's reference number.
3. Call `NetLibIFSettingGet()` or `NetLibIFSettingSet()` with the following parameters:
 - **libRefNum** Reference number of the net library obtained in step 2
 - **ifCreator** Creator of the Spectrum24 network interface; `netIFCreatorS24`
 - **ifInstance** Instance number of the Spectrum24 network interface; `netIFInstanceS24`
 - **setting** Setting to retrieve or set; one of the `s24NetIFSettings` constants
 - **valueP** Buffer pointer for the retrieved or set value of the setting
 - **valueLenP** Buffer size for the retrieved or set value of the setting

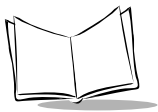
Note: *Applications are not required to configure any of the radio settings using this API. By default, the Spectrum24 terminal radio initializes and connects to the Spectrum24 network based on the preferences set in the Spectrum24 Network Preference Panel on your terminal.*

The rest of this chapter covers:

- [Spectrum24 Data Structures](#) on page 4-10
- [Spectrum24 Constants](#) on page 4-13
- [Spectrum24 Network Interface Settings](#) on page 4-14

For more information on the PalmOS Net Library, see the *Network Communication* chapter in the *Palm OS Programmer's Companion, vol. II, Communications*.

For more information on the NetLibIFSettingGet() and NetLibIFSettingSet() PalmOS API functions, see the *Net Library Functions* section of the *Net Library* chapter in the *Palm OS Programmer's API Reference, Part III: Communications*.



Spectrum24 Data Structures

S24UserNetworkPrefs

These preferences can be get or set by calling `NetLibIFSettingGet()` and `NetLibIFSettingSet()` functions with [netLibSettingS24Preferences](#) on page 4-26. They are also changeable in the Spectrum24 Network Preference Panel in your terminal.

```
struct
{
    Char          szESSID[S24_ESS_ID_LENGTH + 1];
    Boolean       bShowESSID;
    Boolean       bUseDHCP;
    Boolean       bEnabledDNS;
    NetIPAddr     addrStaticIP;
    NetIPAddr     addrSubnetMask;
    NetIPAddr     addrGateway;
    NetIPAddr     addrPrimaryDNS;
    NetIPAddr     addrSecondaryDNS;
    Char          szHostName[S24_HOST_NAME_LEN];
    Char          szDomainName[S24_DOMAIN_NAME_LEN];
} S24UserNetworkPrefs;
```

Field Descriptions

<code>szESSID</code>	ESS ID to use when associating.
<code>bShowESSID</code>	Show/Hide ESS ID string on association screen.
<code>bUseDHCP</code>	Enable or disable DHCP.
<code>bEnabledDNS</code>	Enable or disable DNS.
<code>addrStaticIP</code>	Static IP of the Spectrum24 device. Only used if DHCP is disabled.
<code>addrSubnetMask</code>	Subnet mask for the Spectrum24 device. Only used if DHCP is disabled.
<code>addrGateway</code>	Gateway used by the Spectrum24 device. Only used if DHCP is disabled.

<code>addrPrimaryDNS</code>	Primary DNS used by the Spectrum24 device. Only used if DHCP is disabled.
<code>addrSecondaryDNS</code>	Secondary DNS used by the Spectrum24 device. Only used if DHCP is disabled.
<code>szHostName</code>	Host name of the Spectrum24 device.
<code>szDomainName</code>	Domain name of the Spectrum24 device.



S24MKKCallsign

MKK call sign data structure (for Japanese radio configuration only).

The MKK call sign buffer is structured as follows: The first byte is a flag byte. The following 15 bytes are the preamble, the frame delimiter, and the encoded MKK serial number.

```
struct
{
    Boolean      valid;
    UInt8        callsign[S24_MKK_CALLSIGN_LEN];
} S24MKKCallsign;
```

Field Descriptions

valid	Valid flag
callsign	Call sign data

S24IFSetting

I/F setting structure, used to pass complex data structures to NetLibIFSettingGet() and NetLibIFSettingSet() functions.

```
struct
{
    UInt16       option;
    MemPtr       buffer;
} S24IFSetting;
```

Field Descriptions

option	I/F setting, i.e., specific Spectrum24 preference
buffer	Pointer to data being get or set

Spectrum24 Constants

- `netIFCreatorS24` Creator ID of the Spectrum24 network interface passed to `NetLibIFSettingGet()` and `NetLibIFSettingSet()` functions.
- `netIFInstanceS24` Instance of the Spectrum24 network interface passed to `NetLibIFSettingGet()` and `NetLibIFSettingSet()` functions.

S24PreferenceType

Following is a list of Spectrum24 network preferences.

Table 4-1. Spectrum24 Network Preferences

Preference	Data Type	Description
<code>s24PrefESSID</code>	<code>Char[S24_ESS_ID_LENGTH + 1]</code>	ESS ID to use when associating
<code>s24PrefShowESSID</code>	Boolean	Show ESS ID string on association screen
<code>s24PrefUseDHCP</code>	Boolean	Enable DHCP
<code>s24PrefEnableDNS</code>	Boolean	Enable DNS
<code>s24PrefStaticIP</code>	<code>NetIPAddr</code>	Static IP of the Spectrum24 device
<code>s24PrefSubnetMask</code>	<code>NetIPAddr</code>	Subnet mask for the Spectrum24 device
<code>s24PrefGateway</code>	<code>NetIPAddr</code>	Gateway used by the Spectrum24 device
<code>s24PrefPrimaryDNS</code>	<code>NetIPAddr</code>	Primary DNS used by the Spectrum24 device
<code>s24PrefSecondaryDNS</code>	<code>NetIPAddr</code>	Secondary device used by the Spectrum24 device
<code>s24PrefHostName</code>	<code>Char[S24_HOST_NAME_LEN]</code>	Host name of the Spectrum24 device
<code>s24PrefDomainName</code>	<code>Char[S24_DOMAIN_NAME_LEN]</code>	Domain name of the Spectrum24 device



Spectrum24 Network Interface Settings

Table 4-2. Spectrum24 Network Interface Settings

Setting Name	Page Number
netIFSettingsS24Device	4-15
netIFSettingsS24EssID	4-16
netIFSettingsS24AccessPointBSSID	4-18
netIFSettingsS24DriverVersion	4-19
netIFSettingsS24AssociationStatus	4-20
netIFSettingsS24MKKCallsign	4-22
netIFSettingsS24CountryText	4-23
netIFSettingsS24FirmwareVersion	4-24
netIFSettingsS24FirmwareDate	4-25
netIFSettingsS24Preferences	4-26

netIFSettingS24Device

Description	Allows application to ensure that it is running on a Spectrum24 radio terminal.
Data Type	Boolean
Comments	This is a read only setting.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

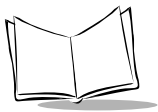
...

UInt16          netLibRef;
UInt16          valueLen;
Boolean          bS24Device;
Err              err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Is this an S24 device? */
valueLen = sizeof(bS24Device);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24Device, &bS24Device, &valueLen);
```



netIFSettingS24EssID

Description	Allows the application to get or set the Spectrum24 radio's ESS_ID string.
Data Type	Char [S24_ESS_ID_LENGTH + 1]
Comments	<p>The ESS_ID identifies the network that the Spectrum24 radio is a member of.</p> <p>Changes made with this setting occur immediately and only affect the current connection. This setting does not change the corresponding Spectrum24 preference setting.</p>

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16 netLibRef;
UInt16 valueLen;
Char    szESSID[S24_ESS_ID_LENGTH + 1];
Char    szNewESSID[S24_ESS_ID_LENGTH + 1] = "NEW_ESSID";
Err     err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get current ESSID */
valueLen = sizeof(szESSID);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24EssID, &szESSID, &valueLen);

...
```

```
/* Set new ESSID */  
valueLen = sizeof(szNewESSID);  
err = NetLibIFSettingSet(netLibRef, netIFCreatorS24, netIFInstancesS24,  
                        netIFSettingS24EssID, &szNewESSID, valueLen);
```



netIFSettingS24AccessPointBSSID

Description	Retrieves the BSS_ID (MAC address) of the access point to which the Spectrum24 radio is associated.
Data Type	UInt8 [S24_BSS_ID_LENGTH]
Comments	This is a read only setting.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16 netLibRef;
UInt16 valueLen;
UInt8  byApBSSID[S24_BSS_ID_LENGTH];
Err     err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get BSSID of our Access Point */
valueLen = sizeof(byApBSSID);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstancesS24,
                        netIFSettingS24AccessPointBSSID, &byApBSSID,
                        &valueLen);
```

netIFSettingS24DriverVersion

Description Retrieves the version number of the Spectrum24 driver.

Data Type Char [S24_VER_STRING_LEN]

Comments This is a read only setting.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

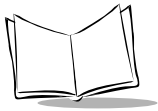
...

UInt16 netLibRef;
UInt16 valueLen;
Char    szDriverVer[S24_VER_STRING_LEN];
Err     err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get driver version */
valueLen = sizeof(szDriverVer);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24DriverVersion, &szDriverVer, &valueLen);
```



netIFSettingS24AssociationStatus

Description	Retrieves the Spectrum24 radio's current association status.
Data Type	Boolean
Comments	<p>This is a read only setting.</p> <p>This setting determines whether the Spectrum24 radio is able to communicate with an access point (AP). Reasons why a Spectrum24 radio cannot associate with an AP include:</p> <ul style="list-style-type: none">• The Spectrum24 radio is out of range of any APs in the local radio network (LRN) defined by the ESSID.• The LRN is down.• The Spectrum24 radio is in the process of switching to another AP (roaming) and will reassociate quickly. <p>If a Spectrum24 radio becomes associated with an AP, do not assume the Spectrum24 radio will continue communicating with the LRN because one of the above conditions may occur. Therefore, if the Spectrum24 radio becomes unassociated with an AP, the application should stop sending data messages on the stack (NetLib) while the radio waits for reassociation. This helps prevent errors due to a transport process timing out a message.</p> <p>The application may also put up a user interface message when unassociated for a period of time. Do not put up this message the first time the unit is unassociated; allow some time to reassociate.</p>

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16      netLibRef;
UInt16      valueLen;
Boolean      bAssociated;
Err          err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get association status */
valueLen = sizeof(bAssociated);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24AssociationStatus, &bAssociated,
                        &valueLen);
```



netIFSettingS24MKKCallsign

Description	Retrieves the MKK call sign bytes for Spectrum24 radios programmed for Japanese configuration.
Data Type	S24MKKCallsign (see S24MKKCallsign on page 4-12 for definition)
Comments	This is a read only setting. This setting call returns meaningful information only for Spectrum24 radios programmed for Japanese configuration. Other country configurations return zeros.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16          netLibRef;
UInt16          valueLen;
S24MKKCallsign  callsign;
Err             err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get MKK callsign */
valueLen = sizeof(callsign);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24MKKCallsign, &callsign, &valueLen);
```

netLibSettingS24CountryText

Description	Retrieves the Spectrum24 radio's country code string.
Data Type	Char [S24_COUNTRY_TEXT_LEN]
Comments	This is a read only setting. Country code is a factory setting, and it is used by the Spectrum24 radio's firmware to program the radio hardware to operate within a particular country's regulations.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16 netLibRef;
UInt16 valueLen;
Char    szCountryText[S24_COUNTRY_TEXT_LEN];
Err      err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get country text */
valueLen = sizeof(szCountryText);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24CountryText, &szCountryText, &valueLen);
```



netIFSettingS24FirmwareVersion

Description	Retrieves the version number of the Spectrum24's RF firmware.
Data Type	Char [S24_VER_STRING_LEN]
Comments	This is a read only setting.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16 netLibRef;
UInt16 valueLen;
Char    szFWVer[S24_VER_STRING_LEN];
Err     err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get adapter firmware version */
valueLen = sizeof(szFWVer);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24FirmwareVersion, &szFWVer, &valueLen);
```

netIFSettingS24FirmwareDate

Description Retrieves the date of the Spectrum24 RF firmware.

Data Type Char [S24_VER_STRING_LEN]

Comments This is a read only setting.

Code Sample

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16 netLibRef;
UInt16 valueLen;
Char    szFWDate[S24_VER_STRING_LEN];
Err     err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

...

/* Get adapter firmware date */
valueLen = sizeof(szFWDate);
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstanceS24,
                        netIFSettingS24FirmwareDate, &szFWDate, &valueLen);
```



netIFSettingS24Preferences

Description	Allows applications to get or set the Spectrum24 radio's network preferences.
Data Type	See Table 4-1 on page 4-13 for data types of individual preferences
Comments	<p>The S24UserNetworkPrefs preferences structure can be used as a template when using the <i>netIFSettingS24Preferences</i> setting.</p> <p>Changes made using this setting do not take place immediately, but occur the next time NetLibOpen() is called. These changes are saved in Spectrum24 preferences database on the terminal until a hard reset is performed.</p>

Code Sample - s24PrefESSID preference

```
#include <PalmOS.h>
#include <NetMgr.h>
#include "S24API.h"

...

UInt16          netLibRef;
UInt16          valueLen;
S24IFSetting    ifSetting;
S24UserNetworkPrefs prefs;
Char            szNewESSID[S24_ESS_ID_LENGTH + 1] = "NEW_ESSID";
Err             err;

/* Obtain network library reference number */
err = SysLibFind("Net.lib", &netLibRef);

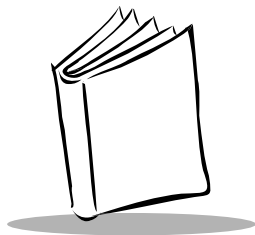
...

/* Get current ESSID from S24 preferences */
ifSetting.option = s24PrefESSID;
ifSetting.buffer = prefs.szESSID;
valueLen = sizeof(prefs.szESSID);
```

```
err = NetLibIFSettingGet(netLibRef, netIFCreatorS24, netIFInstancesS24,  
                        netIFSettingS24Preferences, &ifSetting, &valueLen);  
...  
  
/* Set new ESSID in S24 preferences */  
ifSetting.option = s24PrefESSID;  
ifSetting.buffer = szNewESSID;  
valueLen = sizeof(szNewESSID);  
err = NetLibIFSettingSet(netLibRef, netIFCreatorS24, netIFInstancesS24,  
                        netIFSettingS24Preferences, &ifSetting, valueLen);
```



SPT Terminal Series System Software Manual



Appendix A

ASCII Equivalents

Table A-1 contains the ASCII equivalents for adding prefix and suffix values to scanned data. See [ScanGetPrefixSuffixValues](#) and [ScanSetPrefixSuffixValues](#).

Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1000	00h	%U	CTRL+2
1001	01h	\$A	CTRL+A
1002	02h	\$B	CTRL+B
1003	03h	\$C	CTRL+C
1004	04h	\$D	CTRL+D
1005	05h	\$E	CTRL+E
1006	06h	\$F	CTRL+F
1007	07h	\$G	CTRL+G
1008	08h	\$H	CTRL+H
1009	09h	\$I	CTRL+I
1010	0Ah	\$J	CTRL+J
1011	0Bh	\$K	CTRL+K
1012	0Ch	\$L	CTRL+L

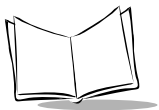


Table A-1. ASCII Equivalents (Continued)

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1013	0Dh	\$M	CTRL+M
1014	0Eh	\$N	CTRL+N
1015	0Fh	\$O	CTRL+O
1016	10h	\$P	CTRL+P
1017	11h	\$Q	CTRL+Q
1018	12h	\$R	CTRL+R
1019	13h	\$S	CTRL+S
1020	14h	\$T	CTRL+T
1021	15h	\$U	CTRL+U
1022	16h	\$V	CTRL+V
1023	17h	\$W	CTRL+W
1024	18h	\$X	CTRL+X
1025	19h	\$Y	CTRL+Y
1026	1Ah	\$Z	CTRL+Z
1027	1Bh	%A	CTRL+[
1028	1Ch	%B	CTRL+\
1029	1Dh	%C	CTRL+]
1030	1Eh	%D	CTRL+6
1031	1Fh	%E	CTRL+-
1032	20h	Space	Space
1033	21h	/A	!
1034	22h	/B	'
1035	23h	/C	#
1036	24h	/D	\$
1037	25h	/E	%

Table A-1. ASCII Equivalents (Continued)

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1038	26h	/F	&
1039	27h	/G	'
1040	28h	/H	(
1041	29h	/I)
1042	2Ah	/J	*
1043	2Bh	/K	+
1044	2Ch	/L	,
1045	2Dh	"	"
1046	2Eh	.	.
1047	2Fh	/	/
1048	30h	0	0
1049	31h	1	1
1050	32h	2	2
1051	33h	3	3
1052	34h	4	4
1053	35h	5	5
1054	36h	6	6
1055	37h	7	7
1056	38h	8	8
1057	39h	9	9
1058	3Ah	/Z	:
1059	3Bh	%F	;
1060	3Ch	%G	<
1061	3Dh	%H	=
1062	3Eh	%I	>

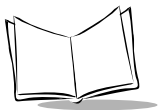


Table A-1. ASCII Equivalents (Continued)

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1063	3Fh	%J	?
1064	40h	%V	@
1065	41h	A	A
1066	42h	B	B
1067	43h	C	C
1068	44h	D	D
1069	45h	E	E
1070	46h	F	F
1071	47h	G	G
1072	48h	H	H
1073	49h	I	I
1074	4Ah	J	J
1075	4Bh	K	K
1076	4Ch	L	L
1077	4Dh	M	M
1078	4Eh	N	N
1079	4Fh	O	O
1080	50h	P	P
1081	51h	Q	Q
1082	52h	R	R
1083	53h	S	S
1084	54h	T	T
1085	55h	U	U
1086	56h	V	V
1087	57h	W	W

Table A-1. ASCII Equivalents (Continued)

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1088	58h	X	X
1089	59h	Y	Y
1090	5Ah	Z	Z
1091	5Bh	%K	[
1092	5Ch	%L	\
1093	5Dh	%M]
1094	5Eh	%N	^
1095	5Fh	%O	_
1096	60h	%W	'
1097	61h	+A	a
1098	62h	+B	b
1099	63h	+C	c
1100	64h	+D	d
1101	65h	+E	e
1102	66h	+F	f
1103	67h	+G	g
1104	68h	+H	h
1105	69h	+I	i
1106	6Ah	+J	j
1107	6Bh	+K	k
1108	6Ch	+L	l
1109	6Dh	+M	m
1110	6Eh	+N	n
1111	6Fh	+O	o
1112	70h	+P	p

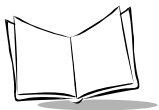
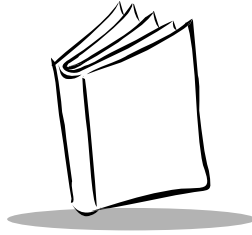


Table A-1. ASCII Equivalents (Continued)

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
1113	71h	+Q	q
1114	72h	+R	r
1115	73h	+S	s
1116	74h	+T	t
1117	75h	+U	u
1118	76h	+V	v
1119	77h	+W	w
1120	78h	+X	x
1121	79h	+Y	y
1122	7Ah	+Z	z
1123	7Bh	%P	{
1124	7Ch	%Q	
1125	7Dh	%R	}
1126	7Eh	%S	~
1127	7Fh	Undefined	



Appendix B

Scan Manager Parameter Definitions

[Table B-1](#) lists the parameters available in the Scan Manager shared library. The information in this table includes parameter name, the terminal default setting, and the acceptable values.

Table B-1. Scan Manager Parameter Definitions

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
ParamDefaults		All defaults	
BeepFrequency	Decode	3000 Hz	0 - 15,000 Hz
	Low	1500 Hz	
	Medium	3000 Hz	
	High	7500 Hz	
BeepDuration	Decode	90 ms	0 - 10,000 ms
	Short	70 ms	
	Medium	90 ms	
	Long	240 ms	
LaserOnTime		3.0 seconds	0 - 10
AimDuration		0.0 seconds	
TriggeringModes		Level	Level, Pulse, Host
BeepAfterGoodDecode		Enable	Enable, Disable



Table B-1. Scan Manager Parameter Definitions (continued)

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
LinearCodeTypeSecurityLevel		Security_Level1	Level1 - Level4
BidirectionalRedundancy		Disable	
BarcodeEnabled	UPC-A	Enable	
	UPC-E	Enable	
	UPC-E1	Disable	
	EAN-8	Enable	
	EAN-13	Enable	
	Bookland EAN	Disable	
	Code 128	Enable	
	UCC/EAN-128	Enable	
	ISBT 128	Enable	
	Code 39	Enable	
	Trioptic Code 39	Disable	
	Code 93	Disable	
	I2of5	Enable	
	D2of5	Disable	
	Codabar	Disable	
	MSI Plessey	Disable	
DecodeUpcEanSupplementals		Ignore	
DecodeUpcEanRedundancy		7	2-20
TransmitCheckDigit	UPC-A	Enable	
	UPC-E	Enable	
	UPC-E1	Enable	
	Code 39	Disable	
	I2of5	Disable	
	MSI Plessey	Disable	

Table B-1. Scan Manager Parameter Definitions (continued)

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
UpcPreamble	UPC-A	System character	
	UPC-E	System character	
	UPC-E1	System character	
Convert	UPC-E to A	Disable	
	UPC-E1 to A	Disable	
	EAN-8 to EAN-13	Type is EAN-13	
	Code 39 to Code 32	Disable	
	I2of5 to EAN-13	Disable	
EanZeroExtend		Disable	
UpcEanSecurityLevel		0	Level 1 - Level 4
Code32Prefix		Disable	
BarcodeLengths	Code 39	2-32	
	Code 93	4-55	
	I2of5	14	
	D2of5	12	
	Codabar	5-55	
BarcodeLengths (cont'd)	MSI Plessey	6-55	
Code39CheckDigitVerification		Disable	
Code39FullAscii		Disable	
I2of5CheckDigitVerification		Disable	
CIsiEditing		Disable	
NotisEditing		Disable	
MsiPlesseyCheckDigits		One	One, Two
MsiPlesseyCheckDigitAlgorithm		Mod 10/Mod 10	



Table B-1. Scan Manager Parameter Definitions (continued)

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
TransmitCodeIdCharacter		None	
PrefixSuffixValues	Prefix	Null	
	Suffix 1	LF	
	Suffix 2	CR	
ScanDataTransmissionFormat		Data as is	
ScanAngle		Wide	Wide, Narrow
DecodeLedOnTime		3 seconds	0 - 99



Appendix C

Data Editing Overview for Magnetic Stripe Reader

Introduction

The data editing feature allows you to edit the data that has been read from a magnetically encoded card before data is transmitted to the application software. The application software translates the information into the exact format expected by the application software.

Functions

The following functions can be performed on the data input record:

Rearrange the Data

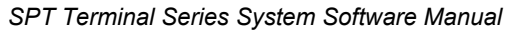
The fields, within a track, created by established standards, can be transmitted to the application software in any order desired, regardless of the order in which they occurred in the card track.

Insert Character Strings into the Output Data Record

Character Strings can be defined and inserted at any place in the data output record.

Duplicate Fields

Fields, within a track, can be transmitted to the application software as many times as desired and in any order.



Fields, within a track, can be selected for output or not selected for output.

[Matched or Unmatched] If matched, MSR 3000 does not send data that does not match the data edit formula. Otherwise, all data that does not match the data edit formula is sent.

The data editing concept is based upon fields. The standard field location, length, and content are determined by Standards. The field standards for ISO Credit Cards, California driver's licenses, and AAMVA driver's licenses are listed in Appendix A. By separating the input data record into small blocks(fields), each block can then be treated separately. Additional fields can also be added to the record in any position, allowing specific functions, such as carriage returns. The fields are identified by field number starting with the character 'a' up to 'z', in the order they were created, starting with the predetermined fields in the standard and adding any newly created fields, allowing as many as 26 fields per track to be defined. These fields are then sent to the application software in the order in which the user specifies.

For example, if the input data record is in the Credit Card Format for Track2:

Field id	a	b	c	d	e	f	g
----------	---	---	---	---	---	---	---

and your application software is expecting the data to be in the following format:

1234567890123456<CR>

then we must break the input data record into fields, select only the desired fields, reverse the order they are sent to the application software, create a new field<CR> and insert it after each field.

We do this by using the defined fields and adding a new field:

Field d = 9912

Added Field a = <CR>

And sending {Field d}{Added Field a}{Field b}{Added Field a}

Formulas

A set of instructions to edit data is referred to as a data editing “formula”. The MSR 3000 supports four types of formulas:

- Credit Card
- California driver’s license
- AAMVA driver’s license
- Customized Format.

The user can either define four types of formula or only one formula each time. At the same time, the MSR 3000 can only keep one Credit Card, California driver’s licenses, AAMVA driver’s licenses and customized format.

If the input data matches the format (Credit Card, Driver’s License, etc.) of the formula, then it will apply the data editing functions and output the reformatted data to the application software. If the input data does not match the criteria spelled out in the first formula, then the match criteria of the second formula is applied. This process continues for each of the successive formulas until a match is found. If no match is found to any of the formulas programmed into the MSR 3000, then either nothing or the unedited data record is transmitted to the application software, according to the Data Edit Mode setting.

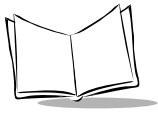
Added Field

An output field is created containing the character string. Up to 6 fields can be defined. The maximum characters of each field is 6.

Search Method

When working with a customer-defined format which is not Credit Card, DMV or AAMVA format, the MSR 3000 supports the following five methods:

- | | |
|----------------|--|
| Length Match: | The card data on the specified track has to meet the minimum and maximum length requirements. |
| String Match: | The card data on the specified track has to include the specified string in the specified position. |
| Search Before: | Generate a new field, which contains the whole message data before matching the specified string for specified times on the specified track. |



- Search Between:** Generate a new field, which contains the whole message data between the matches of the two specified strings for specified times on the specified track.
- Search After:** Generate a new field, which contains the message data with the specified length at a specified offset after matching the specified string for a specified times on the specified track.

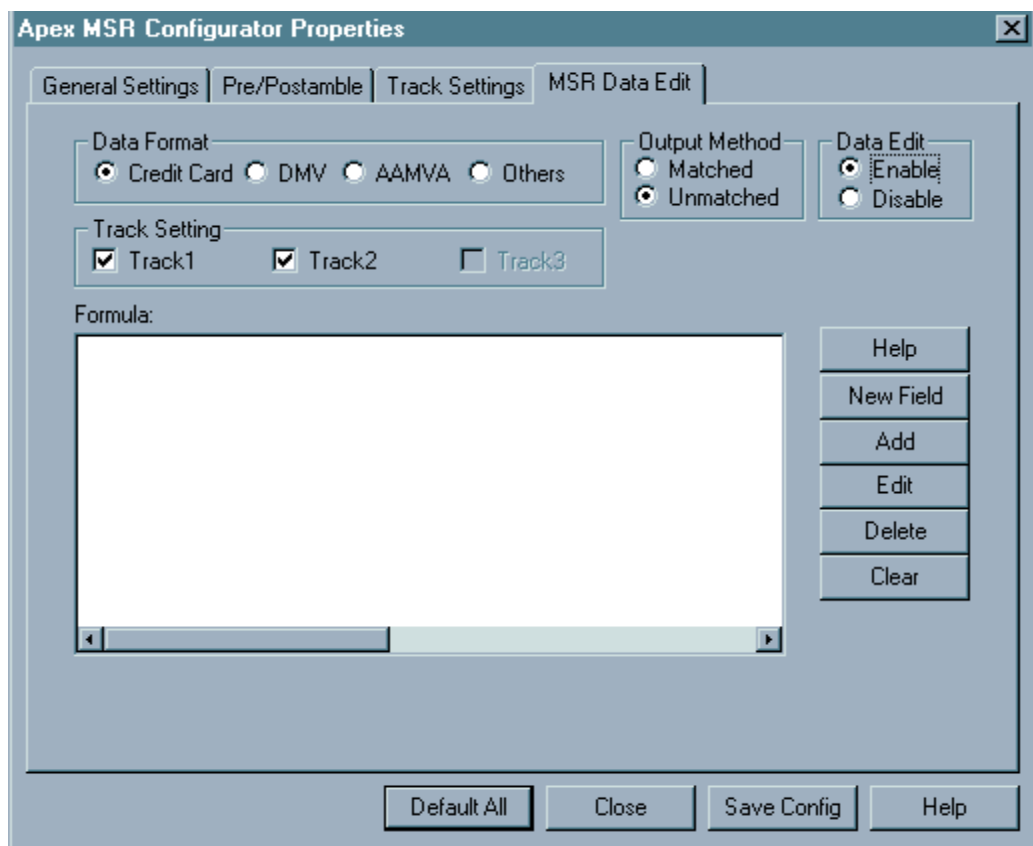
Operation

The data editing of the magnetic stripe is based on fields divided for a format. By separating the input data into fields, each field can be transmitted separately. Additional fields can also be added, allowing specific functions, such as carriage returns to be inserted at any place in data transmitted to the application.

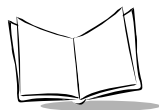
The following includes the all operations performed on the formula:

Add Formula - To add a formula:

1. Set the data edit to 'enable'.
2. Select one of the data formats, such as 'Credit Card'.
3. Select the tracks that you want to transmit the data to the application.
Default is track1, track2 and track3 selected.



4. Press the Add button to display the dialogue box with the selected data format fields. For example, the Credit Card Format.



Credit Card Fields for Track1 and Track2 :

Track1 Fields	Track2 Fields
<input checked="" type="checkbox"/> (a)Start Sentinel	<input checked="" type="checkbox"/> (a)Start Sentinel
<input type="checkbox"/> (b)Format Code	<input checked="" type="checkbox"/> (b)Account Code
<input type="checkbox"/> (c)Account Code	<input type="checkbox"/> (c)Separator
<input type="checkbox"/> (d)Separator	<input type="checkbox"/> (d)Expiration Date
<input checked="" type="checkbox"/> (e)Name	<input type="checkbox"/> (e)Options
<input type="checkbox"/> (f)Separator	<input checked="" type="checkbox"/> (f)End Sentinel
<input checked="" type="checkbox"/> (g)Expiration Date	<input type="checkbox"/> (g)LRC
<input type="checkbox"/> (h)Options	
<input type="checkbox"/> (i)End Sentinel	
<input type="checkbox"/> (j)LRC	

Track 1 Selected Fields:

Track 2 Selected Fields:

Added Fields List:

+

New Field

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a)

1a0a1g0a1e2a2b2f

OK

Cancel

5. In this dialog box , check the fields you want transmitted to the application in track1, track2 and track3. The fields selected display in the Track Selected Fields for each track.
6. If you click the new field button, you can input characters in new filed dialog box to get the new field, you can put this field in any place of output data.
7. In the Track–Selected-Field pull-down list, which lists the all fields selected in this track, select the fields to generate fields sequence which are then displayed in the Fields Sent Sequence edit box. The fields will be sent to the application according to this sequence.
8. Press OK to save the formula, or Cancel button to exit without saving.

Apex MSR Configurator Properties

General Settings | Pre/Postamble | Track Settings | **MSR Data Edit**

Data Format:
☒ Credit Card ☐ DMV ☐ AAMVA ☐ Others

Output Method:
☐ Matched ☒ Unmatched

Data Edit:
☒ Enable ☐ Disable

Track Setting:
☒ Track1 ☒ Track2 ☐ Track3

Formula:
 CreditCard: 1a0a1q0a1e2a2b2f

Help
 New Field
 Add
 Edit
 Delete
 Clear

Default All Close Save Config Help

Example

Data Edit : Enable; Data Format: Credit Card;

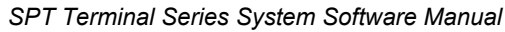
Track Selection: track1 and track2;

Input Data :

%B4000111222333^SMITH/JOHN R.DR^930458799999?C

Field No.	Field Name	Field Value
-----------	------------	-------------

(a)	Start Sentinel	%
-----	----------------	---



- ```
;4000111222333=930458799999000000000000?C
```

- If you want to send the start sentinel, cardholder name and expiration date in track1; and the start sentinel, account code and end sentinel in track2, you should select fields (a), (e), (g) in track1 and field (a), (b), (f) in track2. If you need a new field with value '+' in track1, press the new field button and key in '+' in edit box. Now you have field (a),(e),(g) in track1 and (a),(b),(f) in track2. The added field is '+'.

Field (g) = 9304

Track2:

Field (a) = ;

Field (b) = 4000111222333

Field (f) = ?

Added Field: '+'

Track1 fields selected are (a)(e)(g);

Track2 fields selected are (a)(b)(f);

If you want to send the expiration date before cardholder name and add a new field to use as a separator, go to the combo box and generate the sequence (1a)(0a)(1g)(0a)(1e), so you get the formula in the list box:

Credit Card: (1a)(0a)(1g)(0a)(1e)(2a)(2b)(2f)

The output data should be:

Track1:

%+9304+Smith/John R. DR

Track2:

;4000111222333?

Track1 and track2 output data will be transmitted to application.

**Edit Formula** - To edit the formula selected in the formula list box:

1. Enable data editing.
2. Press the Edit button, which shows the dialogue box with data format of the selected formula in the formula list box.
3. In this dialog box , select the fields to be transmitted to the application in track1, track2 and track3. The selected fields are displayed in the Track Selected Fields for each track.
4. If you click the new field button, you can input characters in new field dialog box to get the new field, you can put this field in any place of output data.
5. In the Track–Selected-Field pull-down list, which lists the all fields selected in this track, select the fields to generate fields sequence which are then displayed in the Fields Sent Sequence edit box. The fields will be sent to the application according to this sequence.



Credit Card Fields for Track1 and Track2 :

| Track1 Fields                                          | Track2 Fields                                         |
|--------------------------------------------------------|-------------------------------------------------------|
| <input checked="" type="checkbox"/> (a)Start Sentinel  | <input checked="" type="checkbox"/> (a)Start Sentinel |
| <input type="checkbox"/> (b)Format Code                | <input checked="" type="checkbox"/> (b)Account Code   |
| <input type="checkbox"/> (c)Account Code               | <input type="checkbox"/> (c)Separator                 |
| <input type="checkbox"/> (d)Separator                  | <input type="checkbox"/> (d)Expiration Date           |
| <input checked="" type="checkbox"/> (e)Name            | <input type="checkbox"/> (e)Options                   |
| <input type="checkbox"/> (f)Separator                  | <input checked="" type="checkbox"/> (f)End Sentinel   |
| <input checked="" type="checkbox"/> (g)Expiration Date | <input type="checkbox"/> (g)LRC                       |
| <input type="checkbox"/> (h)Options                    |                                                       |
| <input type="checkbox"/> (i)End Sentinel               |                                                       |
| <input type="checkbox"/> (j)LRC                        |                                                       |

Track 1 Selected Fields:

Track 2 Selected Fields:

Added Fields List:

+

New Field

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a)

1a0a1g0a1e2a2b2f

OK

Cancel

6. Press OK to save the formula, or Cancel button to exit without saving.

## Example

Enable data editing. If the formula "Credit Card : (1a)(0a)(1g)(0a)(1e)(2a)(2b)(2f)" is selected, Press the Edit button the Credit Card format dialog box will show. (a)(e)(g) in track1 and (a)(b)(f) in track2 were selected.

Track1 fields selected are (a)(e)(g);

Track2 fields selected are (a)(b)(f);

If you want send cardholder name before expiration date ,select the (a)(e)(g) sequentially in the track1 sequence sent combo box.

Track1 sequence sent is changed to (a)(e)(g);

Press the OK button you will get the new formula in the list box of the MSR Edit

dialog-box.

Credit Card : (1a)(1e)(1g)(2a)(2b)(2f)

The output is

Track1:

%Smith/John R. DR9304

Track2:

;4000111222333?

**Delete Formula** - To Delete a formula:

1. Enable data editing.
2. Select a formula you want to delete, then press the Delete button. This formula is deleted from the formula list box.

**Clear Formulas** - To Delete all formulas:

1. Enable data editing.
2. Press the Clear button. All formulas are deleted from the formula list box.

**New Field** - To add new fields:

1. Click the New Field button.
2. Type in the string in the Input Added Field, or, if you want to input a non-printable character, you can use the non-printable character pull-down list to select the non-printable character.
3. Press Add button to append the new field to the added fields list. Press Delete button to delete the selected field from the added fields list.

Press OK to accept, or press Cancel to exit without saving changes.



**New Field**

Added Fields List:

+

Input Added Field:

Non-Printable Char

Add

OK Cancel Delete

---

**Note:** *The maximum number of new fields is 6, the maximum number of characters for each field is 6.*

---

## Customized Format

To create a customized format:

1. On the MSR Configurator Properties screen, select the Others option in the Data Format Group..

The screenshot shows the 'Apex MSR Configurator Properties' dialog box with the 'MSR Data Edit' tab selected. The dialog has four tabs: 'General Settings', 'Pre/Postamble', 'Track Settings', and 'MSR Data Edit'. In the 'MSR Data Edit' tab, there are three main sections: 'Data Format', 'Output Method', and 'Data Edit'. The 'Data Format' section has four radio buttons: 'Credit Card', 'DMV', 'AAMVA', and 'Others' (which is selected). The 'Output Method' section has two radio buttons: 'Matched' and 'Unmatched' (which is selected). The 'Data Edit' section has two radio buttons: 'Enable' (which is selected) and 'Disable'. Below these sections is a 'Track Setting' section with three checkboxes: 'Track1', 'Track2', and 'Track3', all of which are checked. At the bottom left of the dialog is a 'Formula:' label and a large text area for entering a formula. To the right of the text area is a vertical stack of buttons: 'Help', 'New Field', 'Add', 'Edit', 'Delete', and 'Clear'. At the bottom of the dialog are four buttons: 'Default All', 'Close', 'Save Config', and 'Help'.

Apex MSR Configurator Properties

General Settings | Pre/Postamble | Track Settings | MSR Data Edit

Data Format:  
☐ Credit Card ☐ DMV ☐ AAMVA ☒ Others

Output Method:  
☐ Matched ☒ Unmatched

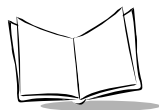
Data Edit:  
☒ Enable ☐ Disable

Track Setting:  
☒ Track1 ☒ Track2 ☒ Track3

Formula:

Help  
New Field  
Add  
Edit  
Delete  
Clear

Default All Close Save Config Help



- Click the Add Button to display the customized field defined dialog box.

Dialog box for defining customized fields:

☒ Length Limit    Track No.     Minimum:     Maximum:

☒ Match String    Track No.     Offset:     String:

Three Method Generating Customized Fields:

Flexible Field No.     Track No.

☐ Before String    Times     String

☒ Among String    Times1     String1     Times2     String2

☐ After String    Times     String     Offset     Length

Customized Fields:

|    |
|----|
| 1a |
| 2b |

Added Fields:

|   |
|---|
| + |
|---|

New Field

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a)

OK

Cancel

- Define the length limit and match string for different tracks, if desired.
- Select a value from the Flexible Field No pull-down menu.
- Select a value from the Track No. pull-down menu.
- Select one of the three search methods.
- Select from customized fields list and added fields list to generate the fields sent sequence.
- Press OK to accept the new formula for the customized format(others), Press Cancel to exit without saving changes.



The MSR Configurator Properties screen redisplay, with the new custom formula listed in the formula list box.

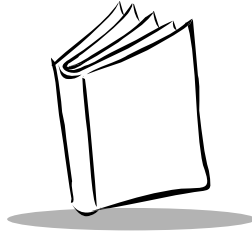
The screenshot shows the 'Apex MSR Configurator Properties' dialog box with the 'MSR Data Edit' tab selected. The dialog has four tabs: 'General Settings', 'Pre/Postamble', 'Track Settings', and 'MSR Data Edit'. The 'MSR Data Edit' tab contains the following controls:

- Data Format:** Radio buttons for 'Credit Card', 'DMV', 'AAMVA', and 'Others'. 'Others' is selected.
- Output Method:** Radio buttons for 'Matched' and 'Unmatched'. 'Unmatched' is selected.
- Data Edit:** Radio buttons for 'Enable' and 'Disable'. 'Enable' is selected.
- Track Setting:** Checkboxes for 'Track1', 'Track2', and 'Track3'. All three are checked.
- Formula:** A text area labeled 'Formula:' containing the text 'Customized: 2b0a1a0a'.
- Buttons:** A vertical stack of buttons on the right: 'Help', 'New Field', 'Add' (highlighted with a dashed border), 'Edit', 'Delete', and 'Clear'.

At the bottom of the dialog are four buttons: 'Default All', 'Close', 'Save Config', and 'Help'.



*SPT Terminal Series System Software Manual*



## *Appendix D*

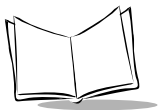
### *Common Magnetic Card Encoding Formats for MSR*

#### **Credit Card Format**

---

##### ***Track1 Format:***

| Field No. | Element/Definition                  | Size     |
|-----------|-------------------------------------|----------|
| A         | Start Sentinel<br>Always "%"        | 1        |
| B         | Format Code<br>Always "B"           | 1        |
| C         | Account Code<br>13 or 16 characters | 13 or 16 |
| D         | Separator<br>Always "^"             | 1        |
| E         | Cardholder Name                     | variable |



|   |                                          |          |
|---|------------------------------------------|----------|
| F | Separator<br>Always "^"                  | 1        |
| G | Expiration Date<br>4 Digits, YYMM Format | 4        |
| H | Optional Discretionary Data              | variable |
| I | End Sentinel<br>Always "?"               | 1        |
| J | LRC                                      | 1        |

### ***Track2 Format:***

| Field No. | Element/Definition                       | Size     |
|-----------|------------------------------------------|----------|
| A         | Start Sentinel<br>Always ";              | 1        |
| B         | Account Code<br>13 or 16 characters      | 13or16   |
| C         | Separator<br>Always "="                  | 1        |
| D         | Expiration Date<br>4 Digits, YYMM Format | 4        |
| E         | Optional Discretionary Data              | variable |
| F         | End Sentinel<br>Always "?"               | 1        |
| G         | LRC                                      | 1        |

## California Driver's License Format (DMV)

---

### ***Track1 Format:***

| Field No. | Element/Definition                                                                                                   | Size |
|-----------|----------------------------------------------------------------------------------------------------------------------|------|
| A         | Start Sentinel 05                                                                                                    | 1    |
| B         | Format Type<br><br>C=Commercial<br><br>S=Salesperson<br><br>D=Driver<br><br>I=Identification<br><br>R=Senior Citizen | 1    |
| C         | Name Line1                                                                                                           | 29   |
| D         | Name Line2                                                                                                           | 29   |
| E         | Address Line                                                                                                         | 29   |
| F         | City                                                                                                                 | 13   |
| G         | End Sentinel(1fh)                                                                                                    | 1    |
| H         | LRC                                                                                                                  | 1    |



## **Track2 Format:**

| Field No. | Element/Definition                                                                                                                                                            | Size             |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| A         | Start Sentinel 05                                                                                                                                                             | 1                |
| B         | IDENTIFICATION NUMBER <ul style="list-style-type: none"><li>• ANSI User ID</li><li>• DL/ID Alpha translated</li><li>• 7 position DL/ID number</li><li>• Check digit</li></ul> | 6<br>2<br>7<br>1 |
| C         | Field Separator                                                                                                                                                               | 1                |
| D         | Expiration Date                                                                                                                                                               | 4                |
| E         | Field Separator                                                                                                                                                               | 1                |
| F         | Discretionary Data Field contains eight position Birth Date                                                                                                                   | 8                |
| G         | End Sentinel (1 fh)                                                                                                                                                           | 1                |
| H         | LRC                                                                                                                                                                           | 1                |

**Track3 Format:**

| Field No. | Element/Definition | Size |
|-----------|--------------------|------|
| A         | Start Sentinel 05  | 1    |
| B         | Class              | 4    |
| C         | Endorsements       | 4    |
| D         | State Code         | 2    |
| E         | Zip Code           | 9    |
| F         | Sex                | 1    |
| G         | Hair               | 3    |
| H         | Eyes               | 3    |
| I         | Height             | 3    |
| J         | Weight             | 3    |
| K         | Restrictions       | 10   |
| L         | Issue Date         | 8    |
| M         | Office             | 3    |
| N         | Employee ID        | 2    |
| O         | LRE ID             | 2    |
| P         | Fee Due Year       | 4    |
| Q         | Address Line2      | 28   |
| R         | Reserved Space     | 11   |
| S         | End Sentinel       | 1    |
| T         | LRC                | 1    |



## Driver's License Format Recommended by AAMVA

---

### ***Track1 Format:***

| Field No. | Element/Definition                                                                                                                                                                                                                    | Size |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| A         | Start Sentinel<br><br>this character must be used at the beginning of the track.                                                                                                                                                      | 1    |
| B         | State or Province(addressee)<br><br>(Mailing or Residential code)<br><br>this field will use the ANSI D-20 standard                                                                                                                   | 2    |
| C         | City<br><br>this field should be truncated with a field separator^ if less than 13 characters long. If a field separator ^ is used, the "NAME" field follows immediately.<br><br>EXAMPLE: Bear^                                       | 13   |
| D         | Name<br><br>this field should be truncated with a field separator^ if less than 35 characters long. If a field separator is used, the "ADDRESS" field follows immediately. The \$ symbol is to be used as a delimiter, between names. | 35   |



|   |                                                                                                                                                                                                                                                                                                                                                      |    |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
|   | EXAMPLE: Roe\$Cheryal\$A^                                                                                                                                                                                                                                                                                                                            |    |
|   | <p>EXAMPLE using "city and Name"</p> <p>Bear^Roe\$Cheryal\$A^</p> <p>at this point a total of 19 bytes have been used, allowing the remainder to be used for the address.</p>                                                                                                                                                                        |    |
| E | <p>Address</p> <p>this field has a minimum length of 29</p> <p>which can be exceeded when utilizing the space from either the city and/or name field.</p> <p>The \$ symbol can be used as a delimiter of multiple address lines.</p> <p>EXAMPLE using the City, Name, Address</p> <p>Combination: Bear^Roe\$Cheryal</p> <p>\$A^123 Something St^</p> | 29 |
| F | <p>End Sentinel</p> <p>this character must be the next to the last character of the track.</p>                                                                                                                                                                                                                                                       | 1  |
| G | <p>LRC</p> <p>this character must be used at the end of the track.</p>                                                                                                                                                                                                                                                                               | 1  |



## Track2 Format:

| Field No. | Element/Definition                                                                                                                                                                                                                                                                                                                                                                                                   | Size |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| A         | Start Sentinel<br><br>this character must be used at the beginning of the track.                                                                                                                                                                                                                                                                                                                                     | 1    |
| B         | ANSI User Code<br><br>this field is assigned by ANSI for the utilization of Track2                                                                                                                                                                                                                                                                                                                                   | 1    |
| C         | ANSI User ID<br><br>this field is the assigned identification number from ISO(International Standards Organization).                                                                                                                                                                                                                                                                                                 | 5    |
| D         | Jurisdiction ID/DL Number<br><br>this field is used to represent the ID/DL number assigned by each jurisdiction. If less than 13 bytes are used the field is truncated by a field separator character (binary bit string 1101). If 13 bytes are used the field separator character MUST appear in the 14th position. Overflow can be accommodated in the field number 7. It is essential that all users refer to the | 14   |

*Common Magnetic Card Encoding Formats for MSR*

|   |                                                                                                                                                        |   |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------|---|
|   | Guidelines which follow.                                                                                                                               |   |
| E | Expiration Date<br><br>this field will be represented in the following<br>format YYMM. This meets the ISO standard                                     | 4 |
| F | Birthdate<br><br>this field will be represented in the ANSI<br>D-20 Standard YYYYMMDD.                                                                 | 8 |
| G | Remainder of Jurisdictional ID/DL #<br><br>this field is used to handle the overflow<br>from the jurisdiction ID/DL field. Refer to the<br>Guidelines. | 5 |
| H | End Sentinel<br><br>this character must be the next to the last<br>character of the track.                                                             | 1 |
| I | LRC<br><br>this character must be used at the end of<br>the track.                                                                                     | 1 |



## Track3 Format:

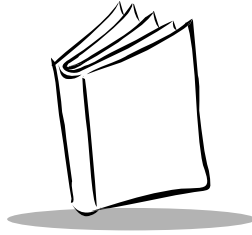
| Field No. | Element/Definition                                                                                                                                                                                                                        | Size |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| A         | Start Sentinel<br><br>this character must be used at the beginning of the track.                                                                                                                                                          | 1    |
| B         | Template Version #<br><br>1 byte table, value 01-63, this field will be used to store the magnetic stripe version being used. It will be necessary to register the use with AAMVA. Refer to the Guidelines.                               | 1    |
| C         | Security Version #.<br><br>1 byte table, value 00-63, this field will be used to store the magnetic security version used. 00 represents security is not used. Refer to the Guidelines.                                                   | 1    |
| D         | Postal Code<br><br>this field will be used to store an 11 position Zip Code or the Canadian postal code. 11 Alphanumeric digits will soon be required to meet postal standards growth. Left justified with spaces filled. Use no hyphens. | 11   |
| E         | Class                                                                                                                                                                                                                                     | 2    |

*Common Magnetic Card Encoding Formats for MSR*

|   |                                                                                                                    |    |
|---|--------------------------------------------------------------------------------------------------------------------|----|
|   | this field will be alphanumeric and will represent the type of license. Use ANSI D-20 codes as modified for CDLIS. |    |
| F | Restrictions<br><br>this field will be alphanumeric and will use the ANSI D20 standard.                            | 10 |
| G | Endorsements<br><br>this field will be alphanumeric and will use the ANSI D-20 standard.                           | 4  |
| H | Sex<br><br>represent as alphanumeric                                                                               | 1  |
| I | Height<br><br>represented as numeric. See ANSI D-20                                                                | 3  |
| J | Weight<br><br>represented as numeric. See ANSI D-20                                                                | 3  |
| K | Hair Color<br><br>represented as alphanumeric. See ANSI D-20                                                       | 3  |
| L | Eye Color<br><br>represented as alphanumeric. See ANSI D-20                                                        | 3  |
| M | ID#<br>this field can be utilized by each jurisdiction as needed, but if used it will be necessary                 | 10 |
|   | to register the use with AAMVA                                                                                     |    |



|   |                                                                                                                                                                 |    |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| N | Reserved Space<br><br>this field can be utilized by each jurisdiction<br>as needed, but if used it will be necessary<br>to register the use with AAMVA          | 16 |
| O | Error Correction<br><br>this field can be utilized by jurisdiction,<br>but is not a mandatory field. Refer to the<br>Guidelines, Appendix "D" for specific use. | 6  |
| P | Security<br><br>this field for use of each jurisdiction. Refer to<br>the Guidelines.                                                                            | 5  |
| Q | End Sentinel<br><br>this character must be the next to the last<br>character for the track.                                                                     | 1  |
| R | LRC<br><br>this character must be used at the end of<br>the track.                                                                                              | 1  |



## *Appendix E*

### *Supported Printers*

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

<http://devzone.symbol.com>

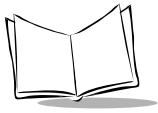
The first piece of information you need is the printer model name. For instance, if you are working with the ONeil printer, you need to provide the string "ONeil" as the printer model in the *ptOpenPrinter* function call. If you are working with the Paxar Monarch 9490 printer, you need to provide either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

To use the Symbol PGP1000 or the Zebra EPLII, use the string "Eltron" as the printer model in the *ptOpenPrinter* function call.

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

The abbreviations used in the tables that follow are:

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



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

## Variables in Printcap Strings

---

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

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



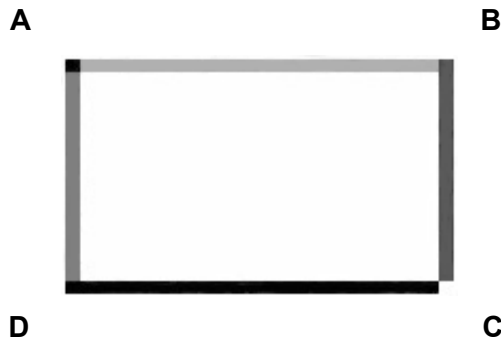
## Printing Rectangles

---

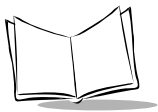
**Note:** *The information in this section applies ONLY to printers that do not print rectangles. These printers create a rectangle by drawing four separate lines.*

---

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



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



## Portable Label Printers

---

*Printer model name: Comtec*

Comtec printcap tokens and associated values:

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

Comtec has no initialization string ("is").

All supported Comtec printers understand the same printer language.

**Printer model name: Eltron**

Eltron printcap tokens and associated values:

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

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

All supported Eltron printers understand the same printer language.



**Printer model name: Paxar Monarch 9490**  
**Monarch 9490**

Monarch 9490 printcap tokens and associated values:

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

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

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

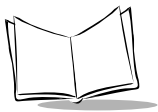
**Printer model name: ONeil**

ONeil printcap tokens and associated values:

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

ONeil has no initialization string ("is").

All supported ONeil printers understand the same printer language.



## Printer model name: Symbol (PGP 1000)

Symbol printcap tokens and associated values:

| Token     | Token description                                               | String Value                                                         |
|-----------|-----------------------------------------------------------------|----------------------------------------------------------------------|
| <b>rs</b> | Reset string                                                    | <CTRL>@                                                              |
| <b>br</b> | Baud rate                                                       | 9600                                                                 |
| <b>sb</b> | Stop bits                                                       | 1                                                                    |
| <b>pr</b> | Parity                                                          | n                                                                    |
| <b>fD</b> | Default font; Symbol PGP 1000 font #2, 16.9 cpi, 7 pt.          | 2;1;1                                                                |
| <b>f1</b> | Other available font; Symbol PGP 1000 font #3, 14.5 cpi, 10 pt. | 3;1;1                                                                |
| <b>bi</b> | High-level API init                                             | \r\nN\r\n                                                            |
| <b>bt</b> | Text command                                                    | A%x,%y,0,%f,%w,%h,N,"%s"\r\n                                         |
| <b>bl</b> | Line command                                                    | LO%x,%y,%l,%t\r\n                                                    |
| <b>bb</b> | Rectangle (box)                                                 | LO%x,%y,%t,%H\r\nLO%X,%y,%t,%H\r\nLO%x,%y,%l,%t\r\nLO%x,%Y,%B,%t\r\n |
| <b>be</b> | High-level API end                                              | P1\r\n                                                               |

Symbol PGP 1000 has no initialization string ("is") or query string ("qs").

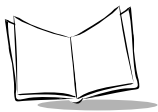
**Printer model name: Zebra (EPL II)**

Zebra printcap tokens and associated values:

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

Zebra EPL II has no initialization string ("is") or query string ("qs").

All supported Zebra EPL II printers understand the same printer language (EPL II).



## Commercial Printers

*Printer model name: PCL*

PCL printcap tokens and associated values:

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

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

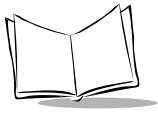


*Printer model name: Postscript*

Postscript printcap tokens and associated values:

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

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



## Using Forms with Legacy Printers

---

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

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

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

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

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

## ***Writing the form to the printer***

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

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



## *SPT Terminal Series System Software Manual*



# Glossary

|                                                |                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Access Point</b>                            | A device that provides transparent access between Ethernet wired networks and IEEE 802.11 interoperable radio-equipped mobile units (MUs) like Symbol's hand-held computers. The mobile unit may roam among the APs in the same subnet while maintaining a continuous, seamless connection to the wired network. Refer to <b>Subnet</b> .                                                                  |
| <b>AP</b>                                      | See <b>Access Point</b> .                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Application Programming Interface (API)</b> | An interface by means of which one software component communicates with or controls another. Usually used to refer to services provided by one software component to another, usually via software interrupts or function calls                                                                                                                                                                            |
| <b>Autodiscrimination</b>                      | The ability of an interface controller to determine the code type of a scanned bar code. After this determination is made, the information content is decoded.                                                                                                                                                                                                                                             |
| <b>Bar</b>                                     | The dark element in a printed bar code symbol.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Bar Code</b>                                | A pattern of variable-width bars and spaces which represents numeric or alphanumeric data in machine-readable form. The general format of a bar code symbol consists of a leading margin, start character, data or message character, check character (if any), stop character, and trailing margin. Within this framework, each recognizable symbology uses its own unique format. See <b>Symbology</b> . |
| <b>Bar Code Density</b>                        | The number of characters represented per unit of measurement (e.g., characters per inch).                                                                                                                                                                                                                                                                                                                  |
| <b>Bar Height</b>                              | The dimension of a bar measured perpendicular to the bar width.                                                                                                                                                                                                                                                                                                                                            |



|                              |                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Bar Width</b>             | Thickness of a bar measured from the edge closest to the symbol start character to the trailing edge of the same bar.                                                                                                                                                                                                                                    |
| <b>Baud Rate</b>             | A measure of the data flow or number of signaling events occurring per second. When one bit is the standard "event," this is a measure of bits per second (bps). For example, a baud rate of 50 means transmission of 50 bits of data per second.                                                                                                        |
| <b>Bit</b>                   | Binary digit. One bit is the basic unit of binary information. Generally, eight consecutive bits compose one byte of data. The pattern of 0 and 1 values within the byte determines its meaning.                                                                                                                                                         |
| <b>Bits per Second (bps)</b> | Bits transmitted or received.                                                                                                                                                                                                                                                                                                                            |
| <b>BSS_ID</b>                | Basic Service Set ID (IEEE MAC address). See <a href="#">MAC Address</a> .                                                                                                                                                                                                                                                                               |
| <b>Byte</b>                  | On an addressable boundary, eight adjacent binary digits (0 and 1) combined in a pattern to represent a specific character or numeric value. Bits are numbered from the right, 0 through 7, with bit 0 the low-order bit. One byte in memory is used to store one ASCII character.                                                                       |
| <b>Character</b>             | A pattern of bars and spaces which either directly represents data or indicates a control function, such as a number, letter, punctuation mark, or communications control contained in a message.                                                                                                                                                        |
| <b>Character Set</b>         | Those characters available for encoding in a particular bar code symbology.                                                                                                                                                                                                                                                                              |
| <b>Check Digit</b>           | A digit used to verify a correct symbol decode. The scanner inserts the decoded data into an arithmetic formula and checks that the resulting number matches the encoded check digit. Check digits are required for UPC but are optional for other symbologies. Using check digits decreases the chance of substitution errors when a symbol is decoded. |
| <b>Codabar</b>               | A discrete self-checking code with a character set consisting of digits 0 to 9 and six additional characters: (- \$ : / , +).                                                                                                                                                                                                                            |
| <b>Code 128</b>              | A high density symbology which allows the controller to encode all 128 ASCII characters without adding extra symbol elements.                                                                                                                                                                                                                            |

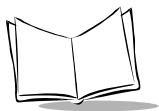
|                                            |                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Code 3 of 9 (Code 39)</b>               | A versatile and widely used alphanumeric bar code symbology with a set of 43 character types, including all uppercase letters, numerals from 0 to 9, and 7 special characters (- . / + % \$ and space). The code name is derived from the fact that 3 of 9 elements representing a character are wide, while the remaining 6 are narrow. |
| <b>Code 93</b>                             | An industrial symbology compatible with Code 39 but offering a full character ASCII set and a higher coding density than Code 39.                                                                                                                                                                                                        |
| <b>Code Length</b>                         | Number of data characters in a bar code between the start and stop characters, not including those characters.                                                                                                                                                                                                                           |
| <b>Continuous Code</b>                     | A bar code or symbol in which all spaces within the symbol are parts of characters. There are no intercharacter gaps in a continuous code. The absence of gaps allows for greater information density.                                                                                                                                   |
| <b>Cradle</b>                              | A cradle is used for charging the terminal battery and for communicating with a host computer, and provides a storage place for the terminal when not in use.                                                                                                                                                                            |
| <b>Data Communications Equipment (DCE)</b> | A device (such as a modem) which is designed to attach directly to a DTE (Data Terminal Equipment) device.                                                                                                                                                                                                                               |
| <b>Data Terminal Equipment (DTE)</b>       | A device (such as a terminal or printer) which is designed to attach directly to a DCE (Data Communications Equipment) device.                                                                                                                                                                                                           |
| <b>DCE</b>                                 | Refer to <b>Data Communications Equipment</b> .                                                                                                                                                                                                                                                                                          |
| <b>Dead Zone</b>                           | An area within a scanner's field of view, in which specular reflection may prevent a successful decode.                                                                                                                                                                                                                                  |
| <b>Decode</b>                              | To recognize a bar code symbology (e.g., Codabar, Code 128, Code 3 of 9, UPC/EAN, etc.) and analyze the content of the bar code scanned.                                                                                                                                                                                                 |
| <b>Decode Algorithm</b>                    | A decoding scheme that converts pulse widths into data representation of the letters or numbers encoded within a bar code symbol.                                                                                                                                                                                                        |
| <b>Depth of Field</b>                      | The range between minimum and maximum distances at which a scanner can read a symbol with a certain minimum element width.                                                                                                                                                                                                               |
| <b>Development Kits</b>                    | A set of software tools provided to customers to help them create applications for their terminals.                                                                                                                                                                                                                                      |



|                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Direct Sequence</b>              | Direct sequence spread spectrum is one of two approaches to spread spectrum modulation for digital signal transmission over airwaves. In direct sequence, data to be transmitted is divided into small pieces allocated to a frequency channel across the spectrum. A data signal at the point of transmission is combined with a higher data-rate bit sequence (also known as a <i>chipping code</i> ) that divides the data according to a spreading ratio. The redundant chipping code helps the signal resist interference and also enables the original data to be recovered if data bits are damaged during transmission. |
| <b>Discrete 2 of 5</b>              | A binary bar code symbology representing each character by a group of five bars, two of which are wide. The location of wide bars in the group determines which character is encoded; spaces are insignificant. Only numeric characters (0 to 9) and START/STOP characters may be encoded.                                                                                                                                                                                                                                                                                                                                      |
| <b>Discrete Code</b>                | A bar code or symbol in which the spaces between characters (intercharacter gaps) are not part of the code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DTE</b>                          | Refer to <b>Data Terminal Equipment</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>EAN</b>                          | European Article Number. This European/International version of the UPC provides its own coding format and symbology standards. Element dimensions are specified metrically. EAN is used primarily in retail.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Element</b>                      | Generic term for a bar or space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Encoded Area</b>                 | Total linear dimension occupied by all characters of a code pattern, including start/stop characters and data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>ESS_ID</b>                       | Extended Service Set ID. 802.11 network name. Can be up to 32 ASCII characters in length.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>File Transfer Protocol (FTP)</b> | A TCP/IP application protocol governing file transfer via network or telephone lines. Refer to <b>TCP/IP</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Form</b>                         | A set of printer-specific commands that format a receipt or label. Refer to the printer manufacturer's documentation for more detailed information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Frequency Hopping</b>            | The use of a random sequence of frequency channels to achieve spread spectrum compliance. Stations that use frequency hopping change their communications frequency at regular intervals. A <b>hopping sequence</b> determines the pattern at which frequencies are changed. Messages take place within a hop. Refer to <b>Hopping Sequence</b> and <b>Spread Spectrum</b> .                                                                                                                                                                                                                                                    |



|                                                    |                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FTP</b>                                         | See <b>File Transfer Protocol</b> .                                                                                                                                                                                                                                                                                                                                                                |
| <b>General purpose printer interface functions</b> | A set of commands that control the basic operation of the printer, such as opening the printer, initializing the printer, and closing the printer.                                                                                                                                                                                                                                                 |
| <b>High-level printer API calls</b>                | A set of commands to the printer to print text and simple graphics.                                                                                                                                                                                                                                                                                                                                |
| <b>Hopping Sequence</b>                            | A set of random frequencies designed to minimize interference with other sets of random frequencies. A hopping sequence determines the pattern with which a station that uses frequency hopping changes its communications frequency. Refer to Frequency Hopping.                                                                                                                                  |
| <b>Host</b>                                        | A computer that serves other terminals in a network, providing services such as network control, data base access, special programs, supervisory programs, or programming languages.                                                                                                                                                                                                               |
| <b>Host Computer</b>                               | A computer that serves other terminals in a network, providing such services as computation, database access, supervisory programs, and network control.                                                                                                                                                                                                                                           |
| <b>IEC</b>                                         | International Electrotechnical Commission. This international agency regulates laser safety by specifying various laser operation classes based on power output during operation.                                                                                                                                                                                                                  |
| <b>IEC (825) Class 1</b>                           | This is the lowest power IEC laser classification. Conformity is ensured through a software restriction of 120 seconds of laser operation within any 1000 second window and an automatic laser shutdown if the scanner's oscillating mirror fails.                                                                                                                                                 |
| <b>Intercharacter Gap</b>                          | The space between two adjacent bar code characters in a discrete code.                                                                                                                                                                                                                                                                                                                             |
| <b>Interleaved 2 of 5</b>                          | A binary bar code symbology representing character pairs in groups of five bars and five interleaved spaces. Interleaving provides for greater information density. The location of wide elements (bar/spaces) within each group determines which characters are encoded. This continuous code type uses no intercharacter spaces. Only numeric (0 to 9) and START/STOP characters may be encoded. |
| <b>Interleaved Bar Code</b>                        | A bar code in which characters are paired together, using bars to represent the first character and the intervening spaces to represent the second.                                                                                                                                                                                                                                                |



|                                     |                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IP</b>                           | Internet Protocol.                                                                                                                                                                                                                                                                                                         |
| <b>LAN</b>                          | Local Area Network.                                                                                                                                                                                                                                                                                                        |
| <b>LASER</b>                        | Light Amplification by Stimulated Emission of Radiation. The laser is an intense light source. Light from a laser is all the same frequency, unlike the output of an incandescent bulb. Laser light is typically coherent and has a high energy density.                                                                   |
| <b>LCD</b>                          | Refer to <b>Liquid Crystal Display</b> .                                                                                                                                                                                                                                                                                   |
| <b>LED</b>                          | Refer to <b>Light Emitting Diode</b> .                                                                                                                                                                                                                                                                                     |
| <b>LED Indicator</b>                | A semiconductor diode (LED - Light Emitting Diode) used as an indicator, often in digital displays. The semiconductor uses applied voltage to produce light of a certain frequency determined by the semiconductor's particular chemical composition.                                                                      |
| <b>Legacy printer</b>               | A printer that uses printer-specific or proprietary control commands. Compare with industry standard commands such as PCL or Postscript.                                                                                                                                                                                   |
| <b>Light Emitting Diode (LED)</b>   | A low power electronic light source commonly used as an indicator light. Uses less power than incandescent light bulb but more than a Liquid Crystal Display (LCD).                                                                                                                                                        |
| <b>Liquid Crystal Display (LCD)</b> | A display that uses liquid crystal sealed between two glass plates. The crystals are excited by precise electrical charges, causing them to reflect light outside according to their bias. They use little electricity and react relatively quickly. They require external light to reflect their information to the user. |
| <b>MAC Address</b>                  | MAC (Media Access Control) address is the Spectrum24 radio's unique hardware number. On an Ethernet LAN, it's the same as the Ethernet address.                                                                                                                                                                            |
| <b>Misread (Misdecode)</b>          | A condition which occurs when the data output of a reader or interface controller does not agree with the data encoded within a bar code symbol.                                                                                                                                                                           |
| <b>Nominal</b>                      | The exact (or ideal) intended value for a specified parameter. Tolerances are specified as positive and negative deviations from this value.                                                                                                                                                                               |
| <b>Nominal Size</b>                 | Standard size for a bar code symbol. Most UPC/EAN codes are used over a range of magnifications (e.g., from 0.80 to 2.00 of nominal).                                                                                                                                                                                      |

|                                      |                                                                                                                                                                                                                                                                                                 |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Null Modem</b>                    | A special cable that allows direct connection of two DTE (Data Terminal Equipment) devices by making each perceive the other as a DCE (Data Communications Equipment) device.                                                                                                                   |
| <b>PCL</b>                           | Printer Control Language. A language that enables application programs to control various Hewlett-Packard printers.                                                                                                                                                                             |
| <b>Percent Decode</b>                | The average probability that a single scan of a bar code would result in a successful decode. In a well-designed bar code scanning system, that probability should approach near 100%.                                                                                                          |
| <b>Postscript</b>                    | A programming language that describes the appearance of a printed page.                                                                                                                                                                                                                         |
| <b>Print Contrast Signal (PCS)</b>   | Measurement of the contrast (brightness difference) between the bars and spaces of a symbol. A minimum PCS value is needed for a bar code symbol to be scannable. $PCS = (RL - RD) / RL$ , where RL is the reflectance factor of the background and RD the reflectance factor of the dark bars. |
| <b>Printcap database</b>             | A database of printer capability and initialization information, such as baud rates, stop bits, and reset commands.                                                                                                                                                                             |
| <b>Printer initialization string</b> | A printer-specific command that initializes or sets defaults for a printer. Refer to the printer manufacturer's documentation for more detailed information.                                                                                                                                    |
| <b>Printer library</b>               | A static body of compiled code, containing an API to interface with the serial and IrDa ports of a Palm PDA. The library is linked with the application.                                                                                                                                        |
| <b>Printer lower level API calls</b> | API calls that write directly to the printer.                                                                                                                                                                                                                                                   |
| <b>Printer model</b>                 | The name of the printer in the printcap entry. For example, Comtec RP3.                                                                                                                                                                                                                         |
| <b>Printer name</b>                  | Not currently used. Will be used to identify specific printers.                                                                                                                                                                                                                                 |
| <b>Programming Mode</b>              | The state in which a scanner is configured for parameter values. See SCANNING MODE.                                                                                                                                                                                                             |
| <b>Quiet Zone</b>                    | A clear space, containing no dark marks, which precedes the start character of a bar code symbol and follows the stop character.                                                                                                                                                                |



|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Resolution</b>          | The narrowest element dimension which is distinguished by a particular reading device or printed with a particular device or method.                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Roaming</b>             | A condition in which a Spectrum24 radio associates from one AP to another.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Router</b>              | A device that connects networks and supports the required protocols for packet filtering. Routers are typically used to extend the range of cabling and to organize the topology of a network into subnets. Refer to <b>Subnet</b> .                                                                                                                                                                                                                                                                                         |
| <b>Scan Area</b>           | Area intended to contain a symbol.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Scanner</b>             | <p>An electronic device used to scan bar code symbols and produce a digitized pattern that corresponds to the bars and spaces of the symbol. Its three main components are:</p> <ol style="list-style-type: none"><li>1. Light source (laser or photoelectric cell) - illuminates a bar code.</li><li>2. Photodetector - registers the difference in reflected light (more light reflected from spaces).</li><li>3. Signal conditioning circuit - transforms optical detector output into a digitized bar pattern.</li></ol> |
| <b>Scanning Mode</b>       | The scanner is energized, programmed, and ready to read a bar code.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Scanning Sequence</b>   | A method of programming or configuring parameters for a bar code reading system by scanning bar code menus.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Self-Checking Code</b>  | A symbology that uses a checking algorithm to detect encoding errors within the characters of a bar code symbol.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Shared library</b>      | A shared body of compiled code that resides on the Palm PDA. At run time, this library is dynamically linked with the application.                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Space</b>               | The lighter element of a bar code formed by the background between bars.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Spectrum24</b>          | Symbol's spread spectrum cellular network.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Specular Reflection</b> | The mirror-like direct reflection of light from a surface, which can cause difficulty decoding a bar code.                                                                                                                                                                                                                                                                                                                                                                                                                   |

|                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Spread Spectrum</b>      | <p>A technique for uniformly distributing the information content of a radio signal over a frequency range larger than normally required for robust transmission of data. Spreading the signal without adding additional information adds significant redundancy, which allows the data to be recovered in the presence of strong interfering signals such as noise and jamming signals.</p> <p>The primary advantage of spread spectrum technology is its ability to provide robust communications in the presence of interfering signals.</p> |
| <b>Start/Stop Character</b> | <p>A pattern of bars and spaces that provides the scanner with start and stop reading instructions and scanning direction. The start and stop characters are normally to the left and right margins of a horizontal code.</p>                                                                                                                                                                                                                                                                                                                   |
| <b>STEP</b>                 | <p>Symbol Terminal Enabler Program.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Subnet</b>               | <p>A subset of nodes on a network that are serviced by the same router. Refer to <b>Router</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Symbol</b>               | <p>A scannable unit that encodes data within the conventions of a certain symbology, usually including start/stop characters, quiet zones, data characters, and check characters.</p>                                                                                                                                                                                                                                                                                                                                                           |
| <b>Symbol Aspect Ratio</b>  | <p>The ratio of symbol height to symbol width.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Symbol Height</b>        | <p>The distance between the outside edges of the quiet zones of the first row and the last row.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Symbol Length</b>        | <p>Length of symbol measured from the beginning of the quiet zone (margin) adjacent to the start character to the end of the quiet zone (margin) adjacent to a stop character.</p>                                                                                                                                                                                                                                                                                                                                                              |
| <b>Symbology</b>            | <p>The set of structural rules and conventions used to represent data within a particular bar code (e.g., UPC/EAN, Code 39, PDF417, etc.).</p>                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>TCP/IP</b>               | <p>Refer to <b>Transmission Control Protocol/Internet Protocol</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Terminal</b>             | <p>A Symbol portable computer product.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Tolerance</b>            | <p>Allowable deviation from the nominal bar or space width.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

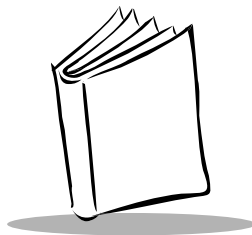


**Transmission  
Control Protocol/  
Internet Protocol  
(TCP/IP)**

A suite of the standard network protocols that were originally used in UNIX environments but are now used in many others. The TCP governs sequenced data; the IP governs packet forwarding. TCP/IP is the primary protocol that defines the Internet.

**UPC**

Universal Product Code. A relatively complex numeric symbology. Each character consists of two bars and two spaces, each of which can be any of four widths. The standard symbology for retail food packages in the United States.



# Index

## A

|                                            |          |
|--------------------------------------------|----------|
| API Architectural Overview .....           | 3-2      |
| API Function Calls .....                   | 3-5      |
| API Introduction .....                     | 3-1, 4-1 |
| Application Programming Interface (API) .. | 3-2      |
| ASCII Equivalents .....                    | A-1      |

## B

|                                   |      |
|-----------------------------------|------|
| Barcode Parameter Functions ..... | 1-37 |
| Codabar .....                     | 1-39 |
| Code 32 .....                     | 1-44 |
| Code 39 .....                     | 1-47 |
| I 2 of 5 .....                    | 1-63 |
| MSI Plessey .....                 | 1-66 |
| UPC/EAN .....                     | 1-71 |
| BSSID .....                       | 4-1  |

## C

|                                       |      |
|---------------------------------------|------|
| Code Samples .....                    | 3-33 |
| Commercial Printers .....             | E-10 |
| PCL .....                             | E-10 |
| Postscript .....                      | E-11 |
| Connecting to the Printer .....       | 3-35 |
| Conventions Used in this Manual ..... | 3-4  |

## D

|                                                            |      |
|------------------------------------------------------------|------|
| Data Editing Overview .....                                | C-1  |
| Data Structures .....                                      | 3-29 |
| Design Considerations .....                                | 4-5  |
| Disconnecting the Printer and Closing the<br>Library ..... | 3-39 |

## E

|             |     |
|-------------|-----|
| ESSID ..... | 4-1 |
|-------------|-----|

## F

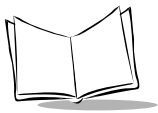
|                           |     |
|---------------------------|-----|
| Feature Limitations ..... | 4-5 |
|---------------------------|-----|

## G

|                                           |      |
|-------------------------------------------|------|
| General Purpose Interface Functions ..... | 3-7  |
| ptClosePrinter .....                      | 3-8  |
| ptConnectPrinter .....                    | 3-9  |
| ptDisconnectPrinter .....                 | 3-11 |
| ptInitPrinter .....                       | 3-12 |
| ptOpenPrinter .....                       | 3-13 |
| ptPrintApiVersion .....                   | 3-15 |
| ptQueryPrinter .....                      | 3-16 |
| ptResetPrinter .....                      | 3-17 |

## H

|                                              |      |
|----------------------------------------------|------|
| Hardware Parameter Functions .....           | 1-84 |
| ScanGetAngle .....                           | 1-95 |
| ScanGetBeepAfterGoodDecode .....             | 1-87 |
| ScanGetBeepDuration .....                    | 1-88 |
| ScanGetBeepFrequency .....                   | 1-89 |
| ScanGetBidirectionalRedundancy ..            | 1-90 |
| ScanGetDecodeLedOnTime .....                 | 1-91 |
| ScanGetLaserOnTime .....                     | 1-92 |
| ScanGetLinearCodeTypeSecurity<br>Level ..... | 1-93 |
| ScanGetPrefixSuffixValues .....              | 1-94 |
| ScanGetScanDataTransmission<br>Format .....  | 1-96 |
| ScanGetTransmitCodeIdCharacter ..            | 1-97 |



|                                   |       |                                 |      |
|-----------------------------------|-------|---------------------------------|------|
| ScanGetTriggeringModes .....      | 1-98  | MsrGetDataBuffer .....          | 2-34 |
| ScanIsPalmSymbolUnit .....        | 1-99  | MsrGetSetting .....             | 2-14 |
| ScanSetAimDuration .....          | 1-100 | MsrGetStatus .....              | 2-17 |
| ScanSetAngle .....                | 1-101 | MsrGetVersion .....             | 2-16 |
| ScanSetBeepAfterGoodDecode ...    | 1-102 | MsrOpen .....                   | 2-10 |
| ScanSetBeepDuration .....         | 1-103 | MsrReadMSRBuffer .....          | 2-36 |
| ScanSetBeepFrequency .....        | 1-104 | MsrReadMSRUnbuffer .....        | 2-38 |
| ScanSetBidirectionalRedundancy .. | 1-105 | MsrSelfDiagnose .....           | 2-19 |
| ScanSetDecodeLedOnTime .....      | 1-106 | MsrSendSetting .....            | 2-15 |
| ScanSetLaserOnTime .....          | 1-107 | MsrSetAddedField .....          | 2-29 |
| ScanSetLinearCodeTypeSecurity     |       | MsrSetBufferMode .....          | 2-20 |
| Level .....                       | 1-108 | MsrSetDataEdit .....            | 2-28 |
| ScanSetPrefixSuffixValues .....   | 1-110 | MsrSetDataEditSend .....        | 2-30 |
| ScanSetScanDataTransmission       |       | MsrSetDecoderMode .....         | 2-39 |
| Format .....                      | 1-111 | MsrSetDefault .....             | 2-13 |
| ScanSetTransmitCodeldCharacter .. | 1-112 | MsrSetFlexibleField .....       | 2-32 |
| ScanSetTriggeringModes .....      | 1-117 | MsrSetLRC .....                 | 2-27 |
| High-level API Calls .....        | 3-18  | MsrSetPreamble .....            | 2-23 |
| ptLineToBuffer .....              | 3-19  | MsrSetReservedChar .....        | 2-42 |
| ptPrintPrintBuffer .....          | 3-20  | MsrSetTerminator .....          | 2-22 |
| ptRectToBuffer .....              | 3-21  | MsrSetTrackFormat .....         | 2-40 |
| ptResetPrintBuffer .....          | 3-22  | MsrSetTrackSelection .....      | 2-25 |
| ptSetFont .....                   | 3-23  | MsrSetTrackSeparator .....      | 2-26 |
| ptStartPrintBuffer .....          | 3-24  | MstSetPostamble .....           | 2-24 |
| ptTextToBuffer .....              | 3-25  |                                 |      |
| <b>I</b>                          |       | <b>N</b>                        |      |
| Information, Service .....        | viii  | NetLib .....                    | 4-5  |
| <b>L</b>                          |       | Network Connections .....       | 4-2  |
| Library Interfaces                |       | Notational Conventions .....    | viii |
| Spectrum24 .....                  | 4-4   |                                 |      |
| Lower-level API Calls .....       | 3-26  | <b>O</b>                        |      |
| ptQueryPrintCap .....             | 3-27  | Opening the Print Library ..... | 3-33 |
| ptWritePrinter .....              | 3-28  |                                 |      |
| <b>M</b>                          |       | <b>P</b>                        |      |
| Magnetic Card Encoding .....      | D-1   | Parameter Definitions .....     | B-1  |
| MSR Commands .....                | 2-8   | Portable Label Printers .....   | E-4  |
| MSR Event .....                   | 2-9   | Comtec .....                    | E-4  |
| MsrArmtoRead .....                | 2-21  | Eltron .....                    | E-5  |
| MsrClose .....                    | 2-12  | ONeil .....                     | E-7  |
|                                   |       | Paxar Monarch 9490 .....        | E-6  |
|                                   |       | Symbol .....                    | E-8  |
|                                   |       | Zebra .....                     | E-9  |



|                           |     |
|---------------------------|-----|
| Power Management .....    | 4-7 |
| Print Commands .....      | 3-6 |
| Printing Rectangles ..... | E-3 |

## Q

|                            |      |
|----------------------------|------|
| Querying the Printer ..... | 3-35 |
|----------------------------|------|

## R

|                                   |     |
|-----------------------------------|-----|
| Returned Status Definitions ..... | 3-5 |
| Roaming .....                     | 4-6 |

## S

|                            |      |
|----------------------------|------|
| Scanner Commands .....     | 1-8  |
| ScanCmdAimOff .....        | 1-10 |
| ScanCloseDecoder .....     | 1-9  |
| ScanCmdAimOn .....         | 1-11 |
| ScanCmdBeep .....          | 1-12 |
| ScanCmdGetAllParams .....  | 1-14 |
| ScanCmdLedOff .....        | 1-15 |
| ScanCmdLedOn .....         | 1-16 |
| ScanCmdParamDefaults ..... | 1-17 |
| ScanCmdScanDisable .....   | 1-18 |
| ScanCmdScanEnable .....    | 1-19 |
| ScanCmdSendParams .....    | 1-20 |
| ScanCmdStartDecode .....   | 1-21 |
| ScanCmdStopDecode .....    | 1-22 |

|                                    |      |
|------------------------------------|------|
| ScanCmdTrigSledOff .....           | 1-23 |
| ScanCmdTrigSledOn .....            | 1-24 |
| ScanGetAimMode .....               | 1-25 |
| ScanGetDecodedData .....           | 1-26 |
| ScanGetDecoderVersion .....        | 1-30 |
| ScanGetExtendedDecodedData .....   | 1-29 |
| ScanGetLedState .....              | 1-31 |
| ScanGetScanEnabled .....           | 1-32 |
| ScanGetScanManagerVersion .....    | 1-33 |
| ScanGetScanPortDriverVersion ..... | 1-34 |
| ScanGetTrigSledMode .....          | 1-35 |
| ScanOpenDecoder .....              | 1-36 |

|                           |      |
|---------------------------|------|
| Service Information ..... | viii |
|---------------------------|------|

## Spectrum24

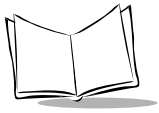
|                                  |     |
|----------------------------------|-----|
| Interfaces .....                 | 4-4 |
| Spectrum24 Radio Interface ..... | 4-5 |
| Supported Printers .....         | E-1 |
| Symbol Support Center .....      | ix  |
| System Requirements .....        | 3-4 |

## U

|                                        |      |
|----------------------------------------|------|
| Using Forms with Legacy Printers ..... | E-12 |
|----------------------------------------|------|

## W

|                                       |      |
|---------------------------------------|------|
| Writing Data to the Printer .....     | 3-37 |
| Writing the form to the printer ..... | E-13 |



## *SPT Terminal Series System Software Manual*

# ***Tell Us What You Think...***

We'd like to know what you think about this Manual. Please take a moment to fill out this questionnaire and fax this form to: (631) 738-3318, or mail to:

Symbol Technologies, Inc.  
One Symbol Plaza M/S B-4  
Holtsville, NY 11742-1300  
Attention: Technical Publications Manager

**IMPORTANT:** If you need product support, please call the appropriate customer support number provided. Unfortunately, we cannot provide customer support at the fax number above.

User's Manual Title: \_\_\_\_\_  
(please include revision level)

How familiar were you with this product before using this manual?

☐ Very familiar    ☐ Slightly familiar    ☐ Not at all familiar

Did this manual meet your needs? If not, please explain.

---

---

What topics need to be added to the index, if applicable?

---

---

What topics do you feel need to be better discussed? Please be specific.

---

---

What can we do to further improve our manuals?

---

---

Thank you for your input—We value your comments.







**72E-56803-01**  
**Revision A — May 2002**