

***Symbol Palm Terminal Scanner System
Software Manual***

70-35914-02

Revision A

May, 1999



© 1998-99 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.

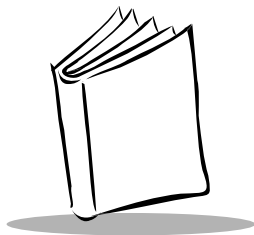
Symbol reserves the right to make changes to any software or product to improve reliability, function, or design.

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.

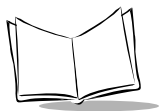
No license is granted, either expressly or by implication, estoppel, or otherwise under any Symbol Technologies, Inc., intellectual property rights. An implied license only exists for equipment, circuits, and subsystems contained in Symbol products.

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>



Chapter 1	
Using the Scan Manager Shared Library	1-1
Using the API	1-1
Using the Scan Demo Application	1-6
 Chapter 2	
Scanner Commands	2-1
Introduction	2-1
Returned Status Definitions	2-2
Scanner Commands	2-3
 Chapter 3	
Barcode Parameter Functions	3-1
Introduction	3-1
Returned Status Definitions	3-2
Barcode Types	3-3
Codabar Barcode Parameter Functions	3-4
Code 32 Barcode Parameter Functions	3-9
Code 39 Barcode Parameter Functions	3-12
General Barcode Parameter Functions	3-17
I 2 of 5 Barcode Parameter Functions	3-30
MSI Plessey Barcode Parameter Functions	3-33
UPC/EAN Barcode Parameter Functions	3-38
 Chapter 4	
Hardware Parameter Functions	4-1
Introduction	4-1
Returned Status Definitions	4-2
Hardware Parameter Functions	4-3
 Chapter 5	
Power Considerations	5-1



scanBatteryErrorEvent	5-1
Sudden Loss of Power	5-1
Backlighting	5-2
Other Power Notes	5-2

Chapter 6

Sample Scanning Application 6-1

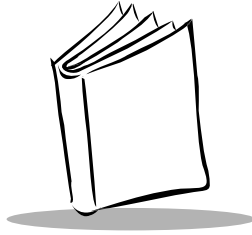
Writing the Code	6-1
------------------------	-----

Appendix A

ASCII Equivalents A-1

Appendix B

Parameter Definitions B-1



About This Guide

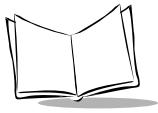
The *Symbol Palm Terminal Scanner System Software Manual* is part of the Scan Manager software development kit (SDK). You can use the SDK to create scan-aware applications for the Symbol Palm Terminal that scan and decode various types of barcodes.

This chapter provides an overview of the *Symbol Palm Terminal Scanner System Software Manual* and provides a list of the appropriate reference documents and conventions. This guide is for developers who want to create scan-aware applications for the terminal. The guide assumes that you are familiar with the CodeWarrior development environment.

Scan Manager Library API SDK Documentation

The *Symbol Palm Terminal Scanner System Software Manual* provides you with:

- ♦ A description of how to use the Scan Manager library
- ♦ Explanations of the Application Program Interface (API) function calls
- ♦ A description of a sample Scan Manager application



What This Guide Contains

This section provides a description of the chapters in this guide.

- Chapter 1** [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.
- Chapter 2** [Scanner Commands](#)—A list of the commands that operate the scanner.
- Chapter 3** [Barcode Parameter Functions](#)—A list of the parameter functions that set the scan parameters associated with specific types of barcodes.
- Chapter 4** [Hardware Parameter Functions](#)—A list of the parameter functions that set the parameters associated with the scanning hardware.
- Chapter 5** [Power Considerations](#)—A description of how Scan Manager functions affect the levels of power available to the scanner hardware.
- Chapter 6** [Sample Scanning Application](#)—A demo application included with the Scan Manager SDK that exercises nearly all of the API.
- Appendix A** [ASCII Equivalents](#)—A list of the scan value, hex value, full ASCII code, and keystrokes for each barcode.
- Appendix B** [Parameter Definitions](#)—A list of the parameters available to developers, and the parameter default values.

Conventions Used in this Guide

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	Hotlinks
<i>Italics</i>	Emphasis (for other elements)



Chapter 1

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.

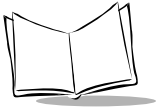
A typical application uses the Scan Manager shared library to do the following, in the order listed below:

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 Chapter 6 for a detailed walk-through of Simple Scan, a sample scanner-enabled application.

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

```
#include "Pilot.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 "SimpleScanRsc.h"  // application resource defines
#include "Utils.h"         // miscellaneous utility functions

DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
    // Check for a normal launch.
    if (cmd == sysAppLaunchCmdNormalLaunch)
    {
        Err error = STATUS_OK;

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

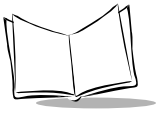
        // Start up the event loop.
        EventLoop();

        // 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:

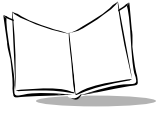
Call the **ScanSetTriggeringModes ()** function to identify the type of trigger that will initiate scans. The typical application passes this function to the **LEVEL** parameter.

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.

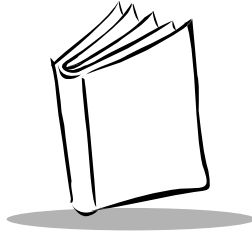


Using the Scan Demo Application

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.



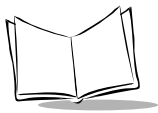
Chapter 2

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 2-1.

Table 2-1. Returned Status Codes

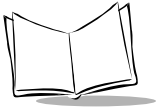
STATUS CODE	DEFINITION
Any non-negative value (0 to 32767)	Parameter value.
STATUS_OK	The function's parameters were verified. If a function must wait for an ACK from the scanner, STATUS_OK indicates that the ACK was received.
NOT_SUPPORTED	The last packet received from the scanner generated either a 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 2-2 lists the scanner commands described in this chapter.

Table 2-2. Scanner Commands

FUNCTION	PAGE
ScanCloseDecoder	2-4
ScanCmdAimOff	2-5
ScanCmdAimOn	2-6
ScanCmdBeep	2-7
ScanCmdGetAllParams	2-9
ScanCmdLedOff	2-10
ScanCmdLedOn	2-11
ScanCmdParamDefaults	2-12
ScanCmdScanDisable	2-13
ScanCmdScanEnable	2-14
ScanCmdSendParams	2-15
ScanCmdStartDecode	2-16
ScanCmdStopDecode	2-17
ScanGetAimMode	2-18
ScanGetDecodedData	2-19
ScanGetDecoderVersion	2-21
ScanGetLedState	2-22
ScanGetScanEnabled	2-23
ScanGetScanManagerVersion	2-24
ScanGetScanPortDriverVersion	2-25
ScanOpenDecoder	2-26



ScanCloseDecoder

Purpose Closes the Scan Manager shared library and frees up system resources.

Prototype `int ScanCloseDecoder (
 void);`

Returned Status Zero=No errors closing shared library
Non-zero=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 `int ScanCmdAimOff (
void);`

Returned Status [STATUS_OK](#)

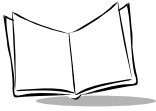
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanCmdAimOn](#)



ScanCmdAimOn

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

Prototype `int ScanCmdAimOn (
void);`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanCmdBeep (BeepType beep);`

Parameters `-> beep` Must be one of the following values:

ONE_SHORT_HIGH
TWO_SHORT_HIGH
THREE_SHORT_HIGH
FOUR_SHORT_HIGH
FIVE_SHORT_HIGH

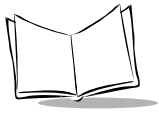
ONE_SHORT_LOW
TWO_SHORT_LOW
THREE_SHORT_LOW
FOUR_SHORT_LOW
FIVE_SHORT_LOW

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

If an error occurs, the returned status is the following:

BAD_PARAM

COMMUNICATIONS_ERROR

NOT_SUPPORTED

ScanCmdGetAllParams

Purpose Retrieves the current parameters from the scanner.

Prototype `int ScanCmdGetAllParams(
 unsigned char *ptr,
 int maxlength);`

Parameters

<p>-> <code>ptr[]</code></p> <p>-> <code>maxlength</code></p>	<p>Array where the scanner's parameter information is deposited.</p> <p>Maximum size (in bytes) of the parameter values stored in the <code>ptr[]</code> array.</p>
--	--

Returned Status Number of bytes copied into `ptr[]`.

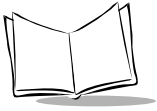
If an error occurs, the returned status is one of the following:

[COMMUNICATION_ERROR](#)

[NOT_SUPPORTED](#)

Comments The location of the array where the parameters are stored begins with `ptr[0]`. The parameters are returned as data pairs consisting of (`parameter_number` and `parameter_value`). You must parse through the data pairs and associate each `parameter_number` with a specific scanner capability. If the number of bytes you specify in `maxlength` is less than the number of scanner parameters retrieved, the remaining parameters are lost. To make sure you retrieve all of the parameters, set `Ptr` to at least 256 bytes.

As you use `ScanSet` commands to set the decoder's parameters, the new parameters will not be reflected in the `ptr[]` array. You must update your own parameter storage when you change parameters.



ScanCmdLedOff

Purpose Immediately turns off the scanner's green LED.

Prototype `int ScanCmdLedOff (
void);`

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanCmdLedOn](#)

ScanCmdLedOn

Purpose Immediately turns on the scanner's green LED.

Prototype `int ScanCmdLedOn (
void);`

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

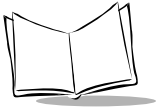
[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

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 `int ScanCmdParamDefaults (
void);`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

ScanCmdScanDisable

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

Prototype `int ScanCmdScanDisable (
void);`

Returned Status [STATUS_OK](#)

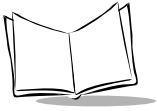
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanCmdScanEnable](#)



ScanCmdScanEnable

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

Prototype

```
int ScanCmdScanEnable (  
    void);
```

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

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 `int 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`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

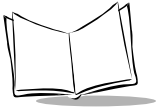
`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanCmdStartDecode (
void);`

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

Comments This command only initiates a scanning session if the trigger mode is set to Host (see [ScanSetTriggeringModes](#)). 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 [ScanSetLaserOnTime](#) x 10.

See Also [ScanCmdStopDecode](#)

ScanCmdStopDecode

Purpose Instructs the scanner to abort a decode attempt.

Prototype `int ScanCmdStopDecode (
void);`

Returned Status [STATUS_OK](#)

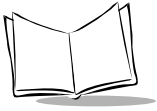
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanCmdStartDecode](#)



ScanGetAimMode

Purpose Identifies whether the scanner is in “normal mode” or “aim” mode (for use as a laser pointer).

Prototype `int ScanGetAimMode (
 void);`

Returned Status Zero=normal mode
 Non-zero=Aim mode

See Also [ScanCmdAimOn](#)
 [ScanCmdAimOff](#)

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 int ScanGetDecodedData (
                                MESSAGE *ptr);
```

Parameters	-> <code>ptr</code>	A pointer to the user-allocated DECODE_DATA_STRUCT where the decoded data is to be placed.
-------------------	---------------------	---

-> ptr->length	Number of characters in the decoded data string.
--------------------------	--

-> ptr->data	Contains the decoded data.
--------------	----------------------------

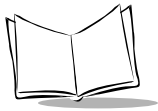
```
-> ptr->data
```

Start of the packet.

```
[ptr->length]      Checksum.
```

-> **ptr->type** 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

If an error occurs, the returned status is one of the following:

BAD_PARAM

COMMUNICATIONS_ERROR

NOT_SUPPORTED

Comments Typically, an application calls this function in response to an EVENT_DECODE_DATA type of event.

ScanGetDecoderVersion

Purpose Retrieves the ASCII revision string of the scanner's decode software. Also copies the string to a user-specified location.

Prototype `int ScanGetDecoderVersion (`
 `CharPtr ptr,`
 `Word max_length);`

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.

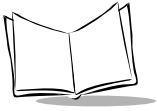
If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanGetLedState (
void);`

Returned Status Zero=**OFF**
Non-zero=**ON**

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

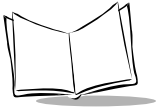
ScanGetScanEnabled

Purpose Indicates whether the scanner is currently enabled.

Prototype `int 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 `int ScanGetScanManagerVersion (
 CharPtr ptr,
 Word 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.

If an error occurs, the returned status is the following:

`NOT_SUPPORTED`

See Also [`ScanGetScanPortDriverVersion`](#)

ScanGetScanPortDriverVersion

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

Prototype `int ScanGetScanPortDriverVersion (
CharPtr ptr,
Word max_length);`

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

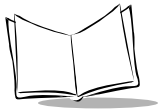
-> `max_length` Maximum number of characters to be copied to `ptr[]`.

Returned Status Length of the revision string.

If an error occurs, the returned status is the following:

[NOT_SUPPORTED](#)

See Also [ScanGetScanManagerVersion](#)



ScanOpenDecoder

Purpose Loads and initializes the Scan Manager shared library, and initializes the scanner.

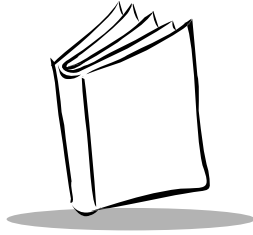
Prototype `int ScanOpenDecoder (
void);`

Returned Status `DECODER_ALREADY_OPEN`—The function was previously called without a corresponding call to the `ScanCloseDecoder` function.

[STATUS_OK](#)

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 `ScanCloseDecoder` function.

See Also [ScanCloseDecoder](#)



Chapter 3

Barcode Parameter Functions

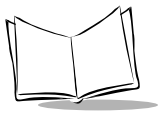
Introduction

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 3-1.

Table 3-1. 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.

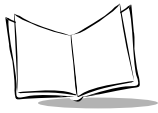
Barcode Types

Table 3-2 lists the barcode types that can be enabled by the parameter functions in this chapter.

Table 3-2. Barcode Types

BARCODE TYPE	PAGE
Codabar Barcode Parameter Functions	3-4
Code 32 Barcode Parameter Functions	3-9
Code 39 Barcode Parameter Functions	3-12
General Barcode Parameter Functions	3-17
I 2 of 5 Barcode Parameter Functions	3-30
MSI Plessey Barcode Parameter Functions	3-33
UPC/EAN Barcode Parameter Functions	3-38

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



Codabar Barcode Parameter Functions

Table 3-3 lists the Codabar barcode parameter functions described in this section.

Table 3-3. Codabar Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetClsiEditing	3-5
ScanGetNotisEditing	3-6
ScanSetClsiEditing	3-7
ScanSetNotisEditing	3-8

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**

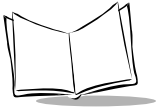
>zero=**ENABLE**

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetClsiEditing](#)



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**

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetNotisEditing](#)

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**

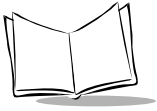
Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

See Also [ScanGetClsiEditing](#)



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`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_ERROR`

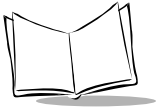
See Also `ScanGetNotisEditing`

Code 32 Barcode Parameter Functions

Table 3-4 lists the Code 32 barcode parameter functions described in this section.

Table 3-4. Code 32 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetCode32Prefix	3-10
ScanSetCode32Prefix	3-11



ScanGetCode32Prefix

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

Prototype `int ScanGetCode32Prefix (
void);`

Returned Status Zero=**DISABLE**

>zero=**ENABLE**

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

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 `int ScanSetCode32Prefix (
 Boolean bEnable);`

Parameters `-> bEnable` Must be one of the following values:

True=**ENABLE**
False=**DISABLE**

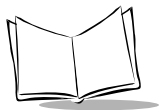
Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

See Also [ScanGetCode32Prefix](#)



Code 39 Barcode Parameter Functions

Table 3-5 lists the Code 39 barcode parameter functions described in this section.

Table 3-5. Code 39 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetCode39CheckDigitVerification	3-13
ScanGetCode39FullAscii	3-14
ScanSetCode39CheckDigitVerification	3-15
ScanSetCode39FullAscii	3-16

ScanGetCode39CheckDigitVerification

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

Prototype `int ScanGetCode39CheckDigitVerification (
void);`

Returned Status `ENABLE`

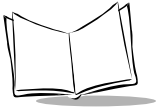
`DISABLE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanGetCode39FullAscii (
void);`

Returned Status Zero=**DISABLE**

>zero=**ENABLE**

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetCode39FullAscii](#)

ScanSetCode39CheckDigitVerification

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

Prototype `int ScanSetCode39CheckDigitVerification (
Word check_digit);`

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

ENABLE
DISABLE

Returned Status [STATUS_OK](#)

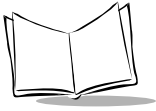
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 `int ScanSetCode39FullAscii (
 Boolean bEnable);`

Parameters -> `full_ascii` Must be one of the following values:

True=**ENABLE**
False=**DISABLE**

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 SH emulates the keystroke BACKSPACE. Scanning ABCSM outputs the keystroke equivalent of ABC ENTER.

Do not enable Code 39 Full ASCII and Trioptic Code 39 at the same time.

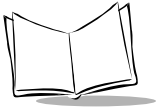
See Also [ScanGetCode39FullAscii](#)

General Barcode Parameter Functions

Table 3-6 lists the general barcode parameter functions described in this section.

Table 3-6. General Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetBarcodeEnabled	3-18
ScanGetBarcodeLengths	3-19
ScanGetConvert	3-21
ScanGetTransmitCheckDigit	3-22
ScanSetBarcodeEnabled	3-23
ScanSetBarcodeLengths	3-25
ScanSetConvert	3-27
ScanSetTransmitCheckDigit	3-29



ScanGetBarcodeEnabled

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

Prototype `int 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
```

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetBarcodeEnabled](#)

ScanGetBarcodeLengths

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

Prototype `int ScanGetBarcodeLengths (
 BarType barcodeType,
 WordPtr pLengthType,
 WordPtr pLength1,
 WordPtr pLength2);`

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

`barCODABAR`

`barCODE39`

`barCODE93`

`barD25`

`barI2of5`

`barMSI_PLESSEY`

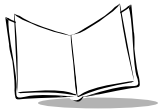
`pLengthType` will be filled with one of the following values:

`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

If an error occurs, the returned status is one of the following:

BAD_PARAM

BATCH_ERROR

COMMUNICATIONS_ERROR

NOT_SUPPORTED

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 `int ScanGetConvert (`
`ConvertType conversion);`

Parameters `-> conversion` Must be one of the following values:

`UPCEtoUPCA`
`UPCE1toUPCA`
`EAN8toEAN13`
`CODE39toCODE32`
`I2OF5toEAN13`

Returned Status `Zero=DISABLE`

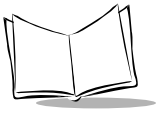
`>zero=ENABLE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [`ScanSetConvert`](#)



ScanGetTransmitCheckDigit

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

Prototype `int ScanGetTransmitCheckDigit (
 barType barcodeType);`

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

`barUPCA`
`barUPCE`
`barUPCE1`
`barCODE39`
`barI2OF5`
`barMSI_PLESSEY`

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

`TRANSMIT_CHECK_DIGIT`

`DO_NOT_TRANSMIT_CHECK_DIGIT`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also `ScanSetTransmitCheckDigit`

ScanSetBarcodeEnabled

Purpose Dictates whether the specified barcode type is to be enabled for decoding.

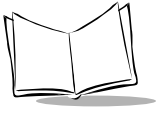
Prototype `int ScanSetBarcodeEnabled (
 BarType barcodeType,
 Boolean bEnable);`

Parameters -> `barcodeType` 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`

-> `bEnable` Must be one of the following values:

`True=ENABLE
False=DISABLE`



Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

See Also [ScanGetBarcodeEnabled](#)

ScanSetBarcodeLengths

Purpose Determines the number of human-readable symbols in the specified format that are to be decoded.

Prototype `int ScanSetBarcodeLengths (
 BarType barcodeType,
 Word lengthType,
 Word length1,
 Word length2);`

Parameters

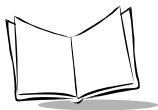
-> barcodeType	Must be one of the following values:
	<pre> barCODABAR barCODE39 barCODE93 barD25 barI2of5 barMSI_PLESSEY </pre>
-> lengthType	Must be one of the following values:
	<pre> ONE_DISCRETE_LENGTH TWO_DISCRETE_LENGTHS LENGTH_WITHIN_RANGE ANY_LENGTH </pre>
-> length1, length2	The discrete lengths 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](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 int ScanSetConvert (
                                ConvertType conversion,
                                Boolean bEnable);
```

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

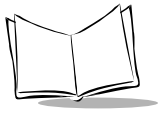
If an error occurs, the returned status is one of the following:

BAD_PARAM

BATCH ERROR

Comments **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).

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).



Converting EAN-8 to EAN-13—When EAN Zero Extend is disabled, this parameter has no effect on barcode data.

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.

See Also [ScanGetConvert](#)

ScanSetTransmitCheckDigit

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

Prototype `int ScanSetTransmitCheckDigit (
 BarType barcodeType,
 Word check_digit);`

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

`barUPCA`
`barUPCE`
`barUPCE1`
`barCODE39`
`barI2OF5`
`barMSI_PLESSEY`

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

`TRANSMIT_CHECK_`
`DIGIT`
`DO_NOT_TRANSMIT_`
`CHECK_DIGIT`

Returned Status [STATUS_OK](#)

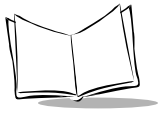
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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](#)



I 2 of 5 Barcode Parameter Functions

Table 3-7 lists the I 2 of 5 barcode parameter functions described in this section.

Table 3-7. I 2 of 5 Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetI2of5CheckDigitVerification	3-31
ScanSetI2of5CheckDigitVerification	3-32

ScanGetI2of5CheckDigitVerification

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

Prototype `int ScanGetI2of5CheckDigitVerification (
void);`

Returned Status `DISABLE`

`OPCC_CHECK_DIGIT`

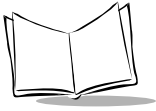
`USS_CHECK_DIGIT`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [ScanSetI2of5CheckDigitVerification](#)



ScanSetI2of5CheckDigitVerification

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

Prototype `int ScanSetI2of5CheckDigitVerification (
Word check_digit);`

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

`DISABLE`
`USS_CHECK_DIGIT`
`OPCC_CHECK_DIGIT`

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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)

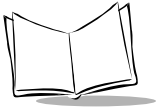
See Also [ScanGetI2of5CheckDigitVerification](#)

MSI Plessey Barcode Parameter Functions

Table 3-8 lists the MSI Plessey barcode parameter functions described in this section.

Table 3-8. MSI Plessey Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetMsiPlesseyCheckDigit Algorithm	3-34
ScanGetMsiPlesseyCheckDigits	3-35
ScanSetMsiPlesseyCheckDigit Algorithm	3-36
ScanSetMsiPlesseyCheckDigits	3-37



ScanGetMsiPlesseyCheckDigit Algorithm

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

Prototype `int ScanGetMsiPlessey
 CheckDigitAlgorithm (
 void);`

Returned Status `MOD10_MOD11`

`MOD10_MOD10`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [ScanSetMsiPlesseyCheckDigitAlgorithm](#)

ScanGetMsiPlesseyCheckDigits

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

Prototype `int ScanGetMsiPlesseyCheckDigits (
void);`

Return Status `ONE_CHECK_DIGIT`

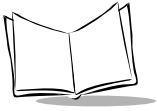
`TWO_CHECK_DIGITS`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [`ScanSetMsiPlesseyCheckDigits`](#)



ScanSetMsiPlesseyCheckDigit Algorithm

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

Prototype `int ScanSetMsiPlessey
 CheckDigitAlgorithm (
 Word algorithm);`

Parameters -> `algorithm` Must be one of the following values:

`MOD10_MOD11`
`MOD10_MOD10`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_ERROR`

See Also `ScanGetMsiPlesseyCheckDigitAlgorithm`

ScanSetMsiPlesseyCheckDigits

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

Prototype `int ScanSetMsiPlesseyCheckDigits (
 Word check_digits);`

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

`ONE_CHECK_DIGIT`
`TWO_CHECK_DIGITS`

Returned Status [STATUS_OK](#)

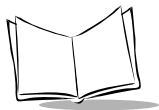
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 3-9 lists the UPC/EAN (European Article Numbering) barcode parameter functions described in this section:

Table 3-9. UPC/EAN Barcode Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetDecodeUpcEanRedundancy	3-39
ScanGetDecodeUpcEanSupplementals	3-40
ScanGetEanZeroExtend	3-41
ScanGetUpcEanSecurityLevel	3-42
ScanGetUpcPreamble	3-43
ScanSetDecodeUpcEanRedundancy	3-44
ScanSetDecodeUpcEanSupplementals	3-45
ScanSetEanZeroExtend	3-47
ScanSetUpcEanSecurityLevel	3-48
ScanSetUpcPreamble	3-50

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 `int ScanGetDecodeUpcEanRedundancy (void);`

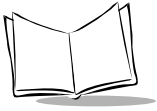
Returned Status Integer in the range [0...20].

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [`ScanSetDecodeUpcEanRedundancy`](#)



ScanGetDecodeUpcEanSupplementals

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

Prototype `int ScanGetDecodeUpcEanSupplementals (
void);`

Returned Status `DECODE_SUPPLEMENTALS`

`IGNORE_SUPPLEMENTALS`

`AUTODISCRIMINATE_SUPPLEMENTALS`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [`ScanSetDecodeUpcEanSupplementals`](#)

ScanGetEanZeroExtend

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

Prototype `int ScanGetEanZeroExtend (
void);`

Returned Status Zero=**DISABLE**

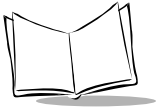
>zero=**ENABLE**

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetEanZeroExtend](#)



ScanGetUpcEanSecurityLevel

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

Prototype `int ScanGetUpcEanSecurityLevel (
void);`

Returned Status `SECURITY_LEVEL0`

`SECURITY_LEVEL1`

`SECURITY_LEVEL2`

`SECURITY_LEVEL3`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also `ScanSetUpcEanSecurityLevel`

ScanGetUpcPreamble

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

Prototype `int ScanGetUpcPreamble (
BarType barcodeType);`

Parameter `-> barcodeType` Must be one of the following values:

`barUPCA`
`barUPCE`
`barUPCE1`

Returned Status One of the following values:

`NO_PREAMBLE`

`SYSTEM_CHARACTER`

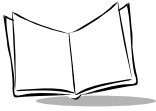
`SYSTEM_CHARACTER_COUNTRY_CODE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanSetDecodeUpcEanRedundancy (`
`Word supplemental_redundancy);`

Parameters -> `supplemental_redundancy` Must be an integer in the range [2...20].

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 int ScanSetDecodeUpcEanSupplementals (
                                     Word supplementals);
```

Parameters -> **supplementals** Must be one of the following values:

DECODE_SUPPLEMENTALS
IGNORE_SUPPLEMENTALS
AUTODISCRIMINATE_
SUPPLEMENTALS

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

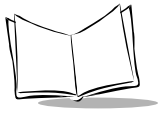
BAD_PARAM

BATCH_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 `int ScanSetEanZeroExtend (
 Boolean bEnable);`

Parameters `-> bEnable` Must be one of the following values:

True=**ENABLE**
False=**DISABLE**

Returned Status [STATUS_OK](#)

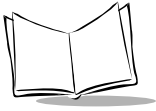
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

Comments This function call makes EAN-8 symbols compatible to EAN-13 symbols.

See Also [ScanGetEanZeroExtend](#)



ScanSetUpcEanSecurityLevel

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

Prototype `int ScanSetUpcEanSecurityLevel (
 Word security_level);`

Parameters -> `security_level` Must be one of the following values:

`SECURITY_LEVEL0`
`SECURITY_LEVEL1`
`SECURITY_LEVEL2`
`SECURITY_LEVEL3`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_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.

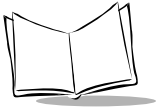
Security Level 0—The default setting. Allows the scanner to operate in its most aggressive state, while providing sufficient security for decoding in-spec barcodes.

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 `int ScanSetUpcPreamble (
 BarType barcodeType,
 int preamble);`

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

`barUPCA`
`barUPCE`
`barUPCE1`

-> `preamble` Must be one of the following values:

`NO_PREAMBLE`
`SYSTEM_CHARACTER`
`SYSTEM_CHARACTER_`

`COUNTRY_CODE`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

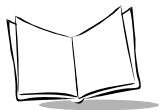
`BATCH_ERROR`

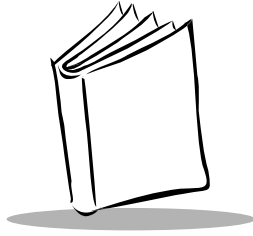
Comments Three options are given for transmitting lead-in characters (preamble) added to UPC-A symbols:

- ♦ 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](#)





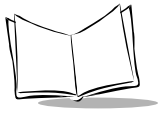
Chapter 4

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 4-1.

Table 4-1. 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.

Hardware Parameter Functions

Table 4-2 lists the hardware parameter functions described in this chapter.

Table 4-2. Hardware Parameter Functions

PARAMETER FUNCTION	PAGE
ScanGetAimDuration	4-5
ScanGetBeepAfterGoodDecode	4-6
ScanGetBeepDuration	4-7
ScanGetBeepFrequency	4-8
ScanGetBidirectionalRedundancy	4-9
ScanGetDecodeLedOnTime	4-10
ScanGetLaserOnTime	4-11
ScanGetLinearCodeTypeSecurityLevel	4-12
ScanGetPrefixSuffixValues	4-13
ScanGetAngle	4-14
ScanGetScanDataTransmissionFormat	4-15
ScanGetTransmitCodeIdCharacter	4-16
ScanGetTriggeringModes	4-17
ScanIsPalmSymbolUnit	4-18
ScanSetAimDuration	4-19
ScanSetAngle	4-20
ScanSetBeepAfterGoodDecode	4-21
ScanSetBeepDuration	4-22
ScanSetBeepFrequency	4-23
ScanSetBidirectionalRedundancy	4-24
ScanSetDecodeLedOnTime	4-25
ScanSetLaserOnTime	4-26

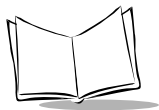


Table 4-2. Hardware Parameter Functions

PARAMETER FUNCTION	PAGE
ScanSetLinearCodeTypeSecurityLevel	4-27
ScanSetPrefixSuffixValues	4-29
ScanSetScanDataTransmissionFormat	4-30
ScanSetTransmitCodeIdCharacter	4-31
ScanSetTriggeringModes	4-38

ScanGetAimDuration

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

Prototype `int ScanGetAimDuration (
void);`

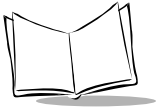
Returned Status Integer in the range [0...99], representing a time period of 0.0 to 9.9 seconds, in 0.1-second increments.

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetAimDuration](#)



ScanGetBeepAfterGoodDecode

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

Prototype `int ScanGetBeepAfterGoodDecode (
void);`

Returned Status `Zero=DISABLE`

`>zero=ENABLE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also `ScanSetBeepAfterGoodDecode`

ScanGetBeepDuration

Purpose Identifies the duration of the beep for the specified beep types.

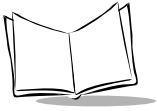
Prototype `int ScanGetBeepDuration (
DurationType type);`

Parameters -> `type` Must be one of the following values:

`DECODE`
`SHORT`
`MEDIUM`
`LONG`

Returned Status [STATUS_OK](#)

See Also [ScanSetBeepDuration](#)



ScanGetBeepFrequency

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

Prototype `int ScanGetBeepFrequency (`
`FrequencyType beep_type);`

Parameters -> `beep_type` Must be one of the following values:

`DECODE`
`LOW`
`MEDIUM`
`HIGH`

Returned Status `STATUS_OK`

See Also `ScanSetBeepFrequency`

ScanGetBidirectionalRedundancy

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

Prototype `int ScanGetBidirectionalRedundancy (
void);`

Returned Status `ENABLE`

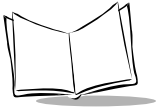
`DISABLE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [ScanSetBidirectionalRedundancy](#)



ScanGetDecodeLedOnTime

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

Prototype `int 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.

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetDecodeLedOnTime](#)

ScanGetLaserOnTime

Purpose Identifies the maximum scanner processing time allowed during a scan.

Prototype `int ScanGetLaserOnTime (
void);`

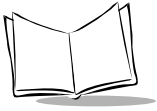
Returned Status Integer in the range [5...99], representing a time period of 0.5 to 9.9 seconds, in 0.1-second increments.

If an error occurs, the returned status is one of the following:

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

See Also [ScanSetLaserOnTime](#)



ScanGetLinearCodeTypeSecurityLevel

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

Prototype `int ScanGetLinearCodeTypeSecurityLevel (
void);`

Returned Status `SECURITY_LEVEL1`

`SECURITY_LEVEL2`

`SECURITY_LEVEL3`

`SECURITY_LEVEL4`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also Comment for `ScanSetLinearCodeTypeSecurityLevel`

ScanGetPrefixSuffixValues

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

Prototype `int ScanGetPrefixSuffixValues (
 CharPtr pPrefix,
 CharPtr pSuffix_1,
 CharPtr pSuffix_2);`

Returned Status `ptr[0]` returns `prefix`

`ptr[1]` returns `suffix_1`

`ptr[2]` returns `suffix_2`

[STATUS_OK](#)

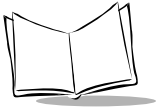
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[COMMUNICATIONS_ERROR](#)

[NOT_SUPPORTED](#)

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



ScanGetAngle

Purpose Identifies the scanner's field of view.

Prototype `int ScanGetAngle (
void);`

Returned Status `SCAN_ANGLE_WIDE`

`SCAN_ANGLE_NARROW`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also `ScanSetAngle`

ScanGetScanDataTransmissionFormat

Purpose Identifies the scan data transmission format.

Prototype `int ScanGetScanDataTransmissionFormat (
void);`

Returned Status `DATA_AS_IS`

 `DATA_SUFFIX1`

 `DATA_SUFFIX2`

 `DATA_SUFFIX1_SUFFIX2`

 `PREFIX_DATA`

 `PREFIX_DATA_SUFFIX1`

 `PREFIX_DATA_SUFFIX2`

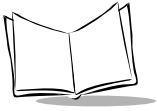
 `PREFIX_DATA_SUFFIX1_SUFFIX2`

If an error occurs, the returned status is one of the following:

[`COMMUNICATIONS_ERROR`](#)

[`NOT_SUPPORTED`](#)

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 `int ScanGetTransmitCodeIdCharacter (
void);`

Returned Status `AIM_CODE_ID_CHARACTER`

`DISABLE`

`SYMBOL_CODE_ID_CHARACTER`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [`ScanSetTransmitCodeIdCharacter`](#)

ScanGetTriggeringModes

Purpose Identifies the type of scan engine trigger.

Prototype `int ScanGetTriggeringModes (
void);`

Return Status `HOST`

`LEVEL`

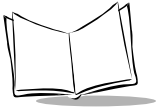
`PULSE`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

See Also [ScanSetTriggeringModes](#)



ScanIsPalmSymbolUnit

Purpose Identifies whether the application is running on an SPT device (Palm organizer with scanner hardware and software).

Prototype `int ScanIsPalmSymbolUnit (
void);`

Return Status `Zero=Unit is not an SPT device`

`Non-zero=Unit is an SPT device`

If an error occurs, the returned status is one of the following:

`COMMUNICATIONS_ERROR`

`NOT_SUPPORTED`

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 `int ScanSetAimDuration (
 Word 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](#)

If an error occurs, the returned status is one of the following:

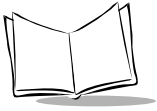
[BAD_PARAM](#)

[BATCH_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 `int ScanSetAngle (
 Word scanner_angle);`

Parameters -> `scanner_angle` Must be one of the following values:

`SCAN_ANGLE_WIDE`
`SCAN_ANGLE_NARROW`

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

Comments A `SCAN_ANGLE_WIDE` field of view allows the scanner to decode more barcode characters at the same time.

See Also [ScanGetAngle](#)

ScanSetBeepAfterGoodDecode

Purpose Determines whether the unit is to beep after a good decode.

Prototype `int ScanSetBeepAfterGoodDecode (
Boolean bEnableBeep);`

Parameters `-> bEnableBeep` Must be one of the following values:

True=**ENABLE**

False=**DISABLE**

Returned Status [STATUS_OK](#)

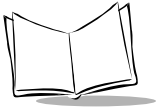
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

Comments When `bEnableBeep` is disabled, the beep still operates during parameter menu scanning, and indicates error conditions.

See Also [ScanGetBeepAfterGoodDecode](#)



ScanSetBeepDuration

Purpose Sets the duration of the beep for the specified beep types.

Prototype `int ScanSetBeepDuration (
DurationType type,
int beep_duration);`

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](#)

If an error occurs, the returned status is:

[BAD_PARAM](#)

Comments Default durations are:

<code>Decode</code>	90 ms
<code>Short</code>	70 ms
<code>Medium</code>	90 ms
<code>Long</code>	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 `int ScanSetBeepFrequency (
 FrequencyType type,
 int beep_freq);`

Parameters -> `type` Must be one of the following values:

`DECODE FREQUENCY`
`LOW FREQUENCY`
`MEDIUM FREQUENCY`
`HIGH FREQUENCY`

-> `beep_freq` A numeric beep frequency in hertz (Hz).

Returned Status [STATUS_OK](#)

If an error occurs, the returned status is:

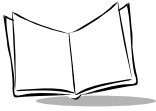
[BAD_PARAM](#)

Comments Default frequencies are:

<code>Decode frequency</code>	3000 Hz
<code>Low frequency</code>	1500 Hz
<code>Medium frequency</code>	3000 Hz
<code>High frequency</code>	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 `int ScanSetBidirectionalRedundancy (`
`Word redundancy);`

Parameters `-> redundancy` Must be one of the following values:

`ENABLE`
`DISABLE`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_ERROR`

Comments This parameter is only valid when the `ScanSetLinearCodeTypeSecurityLevel` 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 `int ScanSetDecodeLedOnTime (
 Word 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](#)

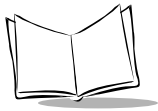
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_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 `int ScanSetLaserOnTime (
 Word laser_on_time);`

Parameters `-> laser_on_time` Must be an integer in the range [5...99], representing a time period of 0.5 to 9.9 seconds.

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_ERROR`

Comments 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.

The `laser_on_time` parameter is programmable in 0.1-second increments.

See Also `ScanGetLaserOnTime`

ScanSetLinearCodeTypeSecurityLevel

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

Prototype `int ScanSetLinearCodeTypeSecurityLevel (
Word security_level);`

Parameters `-> security_level` Must be one of the following values:

`SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3
SECURITY_LEVEL4`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

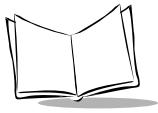
`BAD_PARAM`

`BATCH_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 `int ScanSetPrefixSuffixValues (
Char prefix, Char suffix_1,
Char suffix_2);`

Parameters -> `prefix,` The desired ASCII values.
`suffix_1`
and `suffix_2`

Returned Status [STATUS_OK](#)

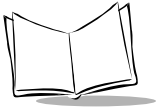
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

[BATCH_ERROR](#)

Comments Before setting the prefix/suffix values, set the Scan Data Transmission Format.

See Also [ScanGetPrefixSuffixValues](#)
[ScanSetScanDataTransmissionFormat](#)
[Appendix A](#) for prefix/suffix values



ScanSetScanDataTransmissionFormat

Purpose Changes the scan data transmission format.

Prototype `int ScanSetScanDataTransmissionFormat (
Word 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`

If an error occurs, the returned status is one of the following:

`BAD_PARAM`

`BATCH_ERROR`

See Also `ScanGetScanDataTransmissionFormat`

ScanSetTransmitCodeIdCharacter

Purpose Selects a character that identifies the scanned barcode's code type.

Prototype `int ScanSetTransmitCodeIdCharacter (`
`Word code_id);`

Parameters `-> code_id` Must be one of the following values:

`SYMBOL_CODE_ID_`
`CHARACTER`
`AIM_CODE_ID_CHARACTER`
`DISABLE`

Returned Status `STATUS_OK`

If an error occurs, the returned status is one of the following:

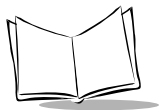
`BAD_PARAM`

`BATCH_ERROR`

Comments 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.

The user may select:

- ♦ No code ID character
- ♦ Symbol Code ID character
- ♦ AIM Code ID character



The Symbol Code ID characters are listed and defined in Table 4-3.

Table 4-3. 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

The definitions for each AIM Code ID character contains a three-character string (in the format **jcm**). These characters are defined in Table 4-4:

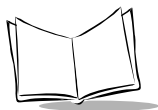
Table 4-4. AIM Code ID Characters

CODE	DEFINITION	REFER TO
l	Flag character (ASCII 93)	N/A
c	Code character	Table 4-5
m	Modifier character	Table 4-6

The Code characters are listed in Table 4-5:

Table 4-5. 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
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



The Modifier characters are listed in Table 4-6:

Table 4-6. 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 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 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 C1

Table 4-6. Modifier Characters

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

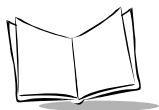
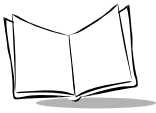


Table 4-6. Modifier Characters

BARCODE TYPE	MODIFIER CHAR	OPTION	EXAMPLE
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	
	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 JS04123
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 JE0012345678905
	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 JX0123456789X

See Also [ScanGetTransmitCodeIdCharacter](#)



ScanSetTriggeringModes

Purpose Sets the type of scan engine trigger.

Prototype `int ScanSetTriggeringModes (`
`Word triggering_mode);`

Parameters -> `triggering_mode` Must be one of the following values:

LEVEL—Only the terminal Scan trigger initiates the scan; the laser is turned off when the trigger is released or the decode was good.

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.

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.

Returned Status [STATUS_OK](#)

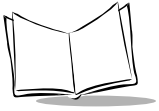
If an error occurs, the returned status is one of the following:

[BAD_PARAM](#)

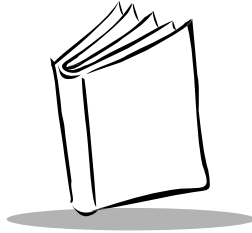
[BATCH_ERROR](#)

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](#)



Symbol Palm Terminal Scanner System Software Manual



Chapter 5

Power Considerations

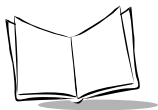
The power-consumption characteristics of the SPT device are different than those of a normal Palm III device, and it is important to keep this in mind when writing your application. The normal low-battery alert is displayed whenever the battery voltage falls below the acceptable operating level. However, 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 SimpleScan) 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

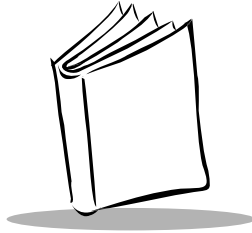
Certain decoder settings affect power consumption, and therefore affect battery life. The “laser pointer” mode set in **ScanCmdAimOn** draws a lot of power. Selecting pulse mode rather than trigger mode generally draws more power. Also, increasing values from the defaults for the following increases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



Chapter 6

Sample Scanning Application

The Scan Manager application described in this chapter is called Simple Scan. 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 Simple Scan 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

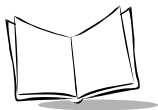
Include Files

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.
*
*****/
DWord PilotMain(Word cmd, Ptr cmdPBP, Word 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

Simple Scan'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

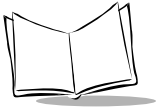
Be sure to check the return value of the **ScanOpenDecoder** call. If it does not return a value of **STATUS_OK**, do **not** proceed with other Scan Manager calls.

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 Simple Scan 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.
*
* PARAMETERS:    None.
*
* RETURNED:      Nothing.
*
*****/
static void StartApplication(void)
{
    Err error;

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

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

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

        // allow software-triggered scans (from Scan button)
        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...
    }
}
```



```

        // Call "ScanCmdSendParams" to send to 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 Simple Scan is the `MainFormHandleEvent` function, which is the event handler for Simple Scan's main form.

```

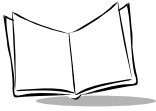
/*****
 *
 * FUNCTION:      MainFormHandleEvent
 *
 * DESCRIPTION:   Handles processing of events for the MainForm form.
 *
 * PARAMETERS:   event - the most recent event.
 *
 * RETURNED:     True if the event is handled, false otherwise.
 *
 *****/
static Boolean MainFormHandleEvent(EventPtr event)
{
    Boolean    bHandled = false;

    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 decoder API to retrieve decoder data.
        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;
}
} //end switch

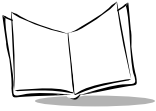
return(bHandled);
}
```

```

/*****
*
* 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( Word menuSel )
{
    switch( menuSel )

```



```
{
    // Options menu
    case OptionsResetDefaults:
        if (ScanIsPalmSymbolUnit())
            ScanCmdParamDefaults();
        break;

    case OptionsAbout:
        OnAbout();
        break;
}

}

/*****
 *
 * FUNCTION:      OnDecoderData (formerly GetSerialData)
 *
 * DESCRIPTION:   Handles processing of events for the main form.
 *
 * PARAMETERS:    event - the most recent event.
 *
 * RETURNED:      True if the event is handled, false otherwise.
 *
 *****/

Boolean OnDecoderData() //GetSerialData()
{
    static Char BarTypeStr[80]=" ";
    MESSAGE decodeDataMsg;

    int refNum = 0;

    int status = ScanGetDecodedData( &decodeDataMsg );

    // if we successfully got the decode data from the API...
    if( status == STATUS_OK )
    {
        // Check to see if this is No Read (NR).
        if( StrNCompare((char*)decodeDataMsg.data,"NR",2)==0)
        {
```

```

        SetFieldText( MainBarTypeField, "No Scan", 30,
                      true );
        SetFieldText( MainScandataField, "No Scan", 79,
                      true );
    }
    else
    {
        ScanGetBarTypeStr( decodeDataMsg.type,
                          BarTypeStr,30); // in Utils.c

        SetFieldText( MainBarTypeField, BarTypeStr, 30,
                      true );

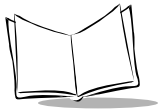
        //Place the scan data into the field and display
        SetFieldText( MainScandataField, (char
                                          *)&decodeDataMsg.data[0], 79, true );
    }
}

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.



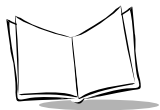
- ♦ **scanBatteryErrorEvent** is another special event issued by Scan Manager to your application. You receive this event whenever a scan operation fails because of low battery levels. When this condition occurs, the scanner is disabled until the batteries are replaced or recharged. Because of this, 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 Simple Scan application does this by issuing an alert.
- ♦ **frmOpenEvent** is a standard event that most Palm organizer developers are familiar with. Simple Scan calls **MainFormOnInit** to initialize and draw the main form.
- ♦ **menuEvent** is another standard Palm event. The **menuEvent** code in Simple Scan 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 Simple Scan, 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

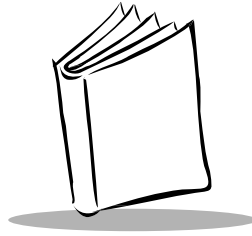
The **StopApplication** function is called at the conclusion of **PilotMain**. This function first uses **ScanIsPalmSymbolUnit** to make sure the application is running on an SPT device. We recommend this check only if your software might be running on both unmodified Palm III devices and SPT devices. If Simple Scan 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 scanner and close Scan Manager library  
        ScanCmdScanDisable();  
        ScanCloseDecoder();  
    }  
}
```



Symbol Palm Terminal Scanner 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
1013	0Dh	\$M	CTRL+M

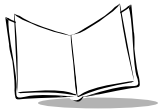


Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
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	%
1038	26h	/F	&
1039	27h	/G	'

Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
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	>
1063	3Fh	%J	?
1064	40h	%V	@
1065	41h	A	A

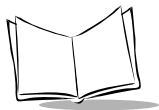


Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
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
1088	58h	X	X
1089	59h	Y	Y
1090	5Ah	Z	Z
1091	5Bh	%K	[

Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
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
1113	71h	+Q	q
1114	72h	+R	r
1115	73h	+S	s
1116	74h	+T	t
1117	75h	+U	u

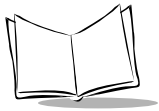
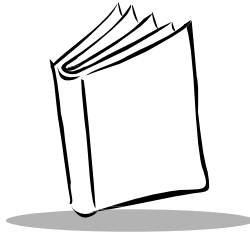


Table A-1. ASCII Equivalents

Scan Value	Hex Value	Full ASCII Code 39 Encode Character	Keystroke
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

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. 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

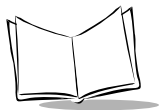


Table B-1. Parameter Definitions

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	

Table B-1. Parameter Definitions

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
TransmitCheckDigit (cont'd)	Code 39	Disable	
	I2of5	Disable	
	MSI Plessey	Disable	
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	

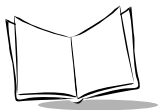


Table B-1. Parameter Definitions

PARAMETER		DEFAULT SETTING	ACCEPTABLE VALUES
BarcodeLengths (cont'd)	MSI Plessey	6-55	
Code39CheckDigitVerification		Disable	
Code39FullAscii		Disable	
I2of5CheckDigitVerification		Disable	
ClsiEditing		Disable	
NotisEditing		Disable	
MsiPlesseyCheckDigits		One	One, Two
MsiPlesseyCheckDigitAlgorithm		Mod 10/Mod 10	
TransmitCodeIdCharacter		None	
PrefixSuffixValues	Prefix	Null	
	Suffix 1	LF	
	Suffix 2	CR	
ScanDataTransmissionFormat		Data as is	
ScanAngle		Wide	Wide, Narrow
DecodeLedOnTime		3 seconds	0 - 99