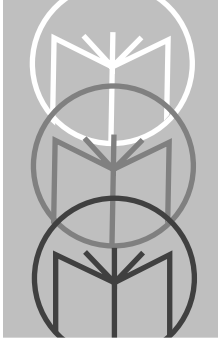


Chapter 4 Scanning Operations

Introduction

This chapter describes SCAN4100 and SCAN4122, the scanner drivers that enable applications running on PPT 4100/4110/4140 (referred to as PPT 41XX) terminals to read bar-coded data. For more information on bar code concepts and operations, see *The Bar Code Book, Second Edition, 1991* written by Roger C. Palmer and published by Helmers Publishing, Inc., 174 Concord Street, Peterborough, NH.

Product Overview

This manual covers both versions of the PPT 4100 scanner drivers:

- SCAN4100 is used with terminals with model number PPT41X0S0XX01
- SCAN4122 is used with terminals with model number PPT41X0S0XX00

The Scanner Type Identifier Program (SCANTYPE) describes how to use the scanner type identification program (SCANTYPE.EXE) to load the correct scanner driver in mixed terminal configurations.

"SCAN41XX" is used when driver functions in SCAN4100 and SCAN4122 are common; differences between the drivers are noted by their individual names.

The PPT 41XX scan driver programs are TSRs that enable applications running on PPT 41XX terminals to read bar-coded data. They are loaded once upon terminal initialization and control the internal scan module.

SCAN41XX uses the SE-1000 scan module. Its functions are to:

- control the scanning device (See ***Note*** below.)
- decode data (SCAN4100 only)
- pass decoded data on to the application program

Note: SCAN41XX can accommodate additional scanner devices.

Scanner Driver Features

The SCAN41XX driver is based on the Series 3000 scanner driver with the following differences.

Some API conventions of SCAN3000 are modified to reorganize decoder control variables.

SCAN41XX is not compatible with programs which use the SCAN3000 TSR on the Series 3000 type terminals. SCAN3000's command structure differs significantly from that of the SCAN41XX drivers used on PPT 41XX terminals.

Refer to *Porting Series 3000 Applications to PPT 41xx Terminals* for a description of the differences between the Series 3000 scanner driver and SCAN41XX.

The user may customize the operation of the SCAN41XX TSR by defining the appropriate parameters in a scanner driver configuration file. Refer to the *Execution and Configuration* section in this chapter.

Programming Guidelines

Following are recommendations for using the Scanner Driver programming interface. This section identifies potential programming errors and promotes future software compatibility between various driver releases and the applications that use them.

Programming Pitfalls

SCBs (Scanner Control Blocks) and data buffers should be located in permanently mapped static memory. Most 'C' programs, for example, allocate a function's local work area on the stack when the function is invoked; this area is de-allocated when the function terminates. Thus, if a function creates an SCB or data buffer in local space, it ceases to exist when the function terminates. Locating SCBs or data buffers in temporarily mapped paged memory results in losing the associated data structure if the appropriate memory page were swapped out before the command is complete. Refer to *User Interface* for a description of an SCB and its function in the interaction between the scanner driver and application programs.

A command is not actually completed until the post processing routine is executed, so a post processing routine can not modify or delete the SCB. The SCAN41XX Driver accesses the SCB after the post processing routine completes to perform housekeeping tasks. The post processing routine can delete or modify any data buffers, since all processing on data buffers is complete by this time.

When using No Wait commands, an application can initiate multiple requests and check the status field for a final completion code (less than 0xF0). When polling SCB status, keep in mind that, although the driver completes commands in the order requested, it may complete more than one SCB in the background. Refer to *User Interface* for definitions of immediate and queued, wait and no wait as these terms apply to the operation of the scanner driver.

An application ensures that all of the SCBs that it issues are returned or flushed by the SCAN41XX driver before the application terminates. The application must verify that pending commands are complete or were successfully canceled before relinquishing its program space. In the DOS environment, all address space used by an application is returned to the free chain upon program termination. (TSR programs retain some memory space.) Unpredictable results occur if the driver continues to access memory that was part of the application address space returned to the system when the program terminated.

Programming Conventions

Before using the Scanner Driver, the application should verify that the driver is installed. It should first check the interrupt vector for a non-zero far address, then either make an out-of-range (0xFF) command request and look for the Unknown Command return code (1), or issue a ***Get Version Information*** request and check for the correct version of the driver.

To ensure compatibility with future driver releases, the application should clear (zero) reserved or undefined fields before invoking the driver. The new driver can then use a zero value for compliance with older releases while using non-zero values to add functionality.

The recommended method of modifying one or more fields in a selected information block is to:

1. read the current values of the information block into a buffer by using the associated ***Get*** command, allowing the caller to determine both the actual information block size and which fields are available in this driver release.
2. modify the desired fields in the information block
3. use the associated ***Set*** command to write the updated information block to the driver

Execution and Configuration

This section describes the execution and configuration of the PPT 41XX Scanner Driver on the terminal under the following main headings:

- *Scanner Driver Execution* describes conditions and procedures for loading the scanner driver on a PPT 41XX terminal, and specifies the command line format and optional command-line parameters.
- *Load Time Messages* lists and describes messages displayed on the terminal screen to indicate the load status of the driver and to report any configuration file errors detected during the loading process.
- *Scanner Configuration File* describes the content and structure of a scanner configuration (.CFG) text file and provides a table of parameter names and values.

Scanner Driver Execution

The PPT 41XX Scanning Driver is shipped as an executable TSR program along with a default configuration file (SCAN41XX.CFG). The Extended Symbol BIOS Services driver (XSYMBIOS.EXE) must be loaded before loading SCAN41XX. XSYMBIOS contains functions which allow SCAN41XX to interface with the PPT 41XX hardware platform.

Place the driver program SCAN41XX and a configuration file in the terminal either on the flash disk or on a PCMCIA card. If the default configuration file name is to be used (SCAN41XX.CFG), be sure it is located on the same drive and directory as SCAN41XX.EXE. If an alternate configuration file is to be used, specify the file as an command-line parameter with **SCAN41XX**, and specify its entire path and drive letter. If SCAN41XX cannot locate the default configuration file, it uses default values to parameterize SCAN41XX. See Tables 4-3 through 4-8 for parameter default values. Table 4-1 shows possible invocations of SCAN41XX and the associated driver actions.

Table 4-1. SCAN41XX Invocations and Associated Driver Actions

Command Line Usage	Driver Action
SCAN4100 or **SCAN4122**	The driver attempts to find and process the default configuration file (SCAN4100.CFG or SCAN4122.CFG as appropriate) on the current drive and directory. If it does not locate this file, it uses the default settings (see Tables 4-3 through 4-8).
SCAN4100 *user*.**CFG** or **SCAN4122** *user*.**CFG**, where *user* is a user-specified name for an alternate configuration file.	The driver attempts to find and process the specified alternate configuration file (*user*.**CFG**) on the current drive and directory. If the file is not located, the driver displays an error message and does not load. See Table 4-2 for a list of possible error messages.
SCAN4100 *dr:\dir\user*.**CFG** or **SCAN4122** *dr:\dir\user*.**CFG**, where *user* is a user-specified name for an alternate configuration file, *dr* is the drive letter and *dir* is the directory on which the configuration file resides.	The driver attempts to find and process the alternate configuration file on the drive and directory specified on the command line. If it does not locate the file, the driver displays an error message and does not load. See Table 4-2 for a list of possible error messages.

Initial invocation of SCAN41XX causes the configuration file to be read in and verified. If any errors are found in the configuration file, the driver displays the error messages on the screen. These messages are among those listed and described in Table 4-2. After SCAN41XX is parameterized, the program terminates, removing the initialization and parameterization portion of code, and stays resident. SCAN41XX can be loaded once; further attempts to load display the SCAN41XX banner and the error message: "ERROR: Scan driver already installed."

Load Time Messages

SCAN41XX displays messages to the PPT 41XX screen to indicate the load status of the driver and configuration file errors. When an error occurs, the terminal beeps twice, displays an error message, and terminates the loading process. Correct the error and cold-boot the PPT 41XX.

Table 4-2 lists SCAN41XX status messages that can be displayed when SCAN41XX is invoked and a brief description of the cause.

Table 4-2. SCAN41XX Status Messages

Status Message	Cause
PPT 41XX Scanner Driver Version *x.xx* Copyright Symbol Technologies, Inc. 1993-1997. All rights reserved.**?**	Displayed on every invocation of SCAN41XX. x.xx specifies the current version and release numbers of the driver.
PPT 41XX Scan driver successfully loaded.	No errors occurred during the load process. The driver has been installed.
ERROR: Scan driver already installed.	The driver is already in memory. The driver in memory is retained.
ERROR: Cannot communicate with scanner card.	The driver cannot detect the scanner device.
ERROR: XSYMBIOS is not loaded.	Extended Symbol BIOS Services were not loaded prior to the invocation of SCAN41XX.
ERROR: Power management hook error.	XSYMBIOS did not allow power management registration.
No errors detected in configuration file.	The configuration file has been located and contains no errors.
No configuration file -- using defaults.	SCAN41XX.CFG was not found; default settings have been used.
ERROR: Configuration file does not exist.	The configuration file specified on the command line was not found.
CFG FILE ERROR: Line -xxx- Extra characters on line.	Extra characters were found on Line xxx in the configuration file.
CFG FILE ERROR: Line -xxx- Missing parameter value.	No value has been specified for the parameter at Line xxx in the configuration file.
CFG FILE ERROR: Line -xxx- Invalid parameter value.	The value for the parameter at Line xxx in the configuration file is not valid.
CFG FILE ERROR: Line -xxx- No starting section name.	No section name is specified at the beginning of the configuration file.

Table 4-2. SCAN41XX Status Messages (Continued)

Status Message	Cause
CFG FILE ERROR: Line -xxx- Invalid keyword for current section.	An invalid parameter name was specified for the current section in the configuration file.
CFG FILE ERROR: Line -xxx- Missing Parameter_Index setting.	A **parameter_index** was not specified in the **Reader_Parameters** section in the configuration file.
CFG FILE ERROR: Line -xxx- Parameter value out of range.	The parameter value specified at Line xxx in the configuration file is out of range.

Scanner Configuration File

A scanner configuration file is a text file that customizes the scanner driver. A default configuration file (SCAN4100.CFG or SCAN4122.CFG as appropriate) is shipped with the scanner driver, however any configuration file, including SCAN4100.CFG and SCAN4122.CFG, may be edited by any PC text editor to allow customization of the driver.

The structure of a scanner configuration file is similar to that of a Windows™ initialization (WIN.INI) file. It has the following format:

[Parameter_Section_Heading1]

parameter name1 = parameter value1

parameter name2 = parameter value2

.
.
.

For example, the section on Scan Parameters might look like this:

[Scan_Parameters]

bidir_redundancy = 1

xmit_code_id = 1

auto_iata = 1

The following are some additional items to note about the scanner configuration file and the way it functions:

- Enclose section headings in square brackets.
- Section names and keyword names are **not** case sensitive.
- If a section is omitted from the configuration file, the scanner driver applies the default values for its parameters. See Tables 2-3 through 2-8 for scanner parameters and their default values.
- Within any section, any parameters that are omitted keep their default values. Place only those parameters that are to be changed in the file, and under the appropriate section.
- If more than one instance of a parameter is specified, only the last setting of that parameter is used by the scanner driver.
- Place comments on a separate line by starting the line with a semicolon.
- Blank lines are ignored.

To change the reader profile, the **parameter_index** keyword must be specified prior to any parameter settings. If multiple reader profiles require initialization, specify each in the **Reader_Parameters** section by setting values after the associated parameter index. (See Table 4-3, below, for valid values for **parameter_index**.)

The following example illustrates multiple reader parameter settings in the configuration file:

[Reader_Parameters]

;Set up SE-1022 parameters

parameter_index = 4

dec_feedb_time = 2

;Set up SMC parameters

parameter_index = 3

dec_feedb_time = 4

;Use SE-1022 parameters as default

default_parameter = 4

Refer to Table 4-2 (SCAN41XX Status Messages) for messages displayed on the terminal screen when SCAN41XX does not locate a configuration file or finds errors in a file it locates during the scanner driver loading process.

Tables 4-3 through 4-8 list the available scanner configuration sections, along with associated parameter names and values for use in configuration files, as follows:

Note: Parameters followed by an asterisk (*) in the following tables are provided for future expansion and are not supported in the first versions of SCAN4122 and SCAN4100.

Parameters that are followed by a number sign (#) in the following tables are provided for future expansion are not supported in the first versions of SCAN4122. These parameters are, however, supported in SCAN4100.

Table 4-3. Reader Parameters

Table 4-4. Scan Parameters

Table 4-5. Decoder Parameters

Table 4-6. UPC/EAN General Parameters

Table 4-7. Trigger Mode Parameters

Table 4-8. Decoder Enable Parameters

Each table contains:

- the parameters associated with the section heading
- the default value the scanner driver applies for each parameter if the parameter or its section is omitted from the configuration file
- the value or range of values for each parameter
- a definition for each parameter value

Table 4-3. Reader Parameters

***Section:* [Reader_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
default_parameter	4 for SCAN4122; 3 for SCAN4100	1 to 10	Set parameter index to use as default.
		255	Auto-discriminate between laser and contact.
parameter_index	4 for SCAN4122; 3 for SCAN4100	1	Set Standard Laser Gun Parameters.
		2	Set Standard Contact Wand Parameters.
		3	Set SMC Card.
		4	Set SE-1022.
		5 to 10	Set User Parameters.
scan_mode	5 for SCAN4122; 4 for SCAN4100	1	Standard Contact Wand
		2	Standard Laser Gun.
		3	CCD Gun.
		4	Scan Module Card.
		5	SE-1022 Module.
		6	Unknown Scanner.
enable_settle_time*	0	0 to 65535	Time in microseconds.
power_settle_time*	0	0 to 65535	Time in microseconds.
inverse_label_flag*	0	0	Normal (black on white)
		1	Inverse (white on black)
white_data_logic_lvl*	1	0	Low Level.
		1	High Level.
dec_beep_time	110	0 to 65535	Time in milliseconds.
dec_beep_freq	2048	0 to 65535	Frequency in megahertz.
scans_per_label*	1	1 to 255	Number of scans.
clk_speed_toggle*	0	0	Do not toggle.
		1	Toggle.
trans_resolution*	8	2, 4, 8, 16	Clock divisor value.
subseq_scan_time*	3	0 to 65	Time in seconds.

Table 4-3. Reader Parameters (Continued)

***Section:* [Reader_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
no_data_time	3	0 to 65	Time in seconds.
post_dec_action*	0	0	Pulse / Wait.
		1	Acquire.
		2	Stop.
dec_fail_action*	0	0	Pulse / Wait.
		1	Acquire.
		2	Stop.
prod_trigger*	1	0	Does not trigger.
		1	Can trigger.
two_stage_trigger*	0	0	Does not have two stage trigger.
		1	Does have two stage trigger.
multiple_scan*	1	0	Does not multiple scan.
		1	Does multiple scan.
prod_direction*	1	0	Does not give direction.
		1	Does give direction.
dec_feedb_time	3	0 to 65	Time in seconds.
dec_feedb_lvl	1	0	Low level.
		1	High level.
scan_led_ctl*	0	0	Cannot control scanning LED.
		1	Can control scanning LED.
scan_led_lvl*	1	0	Low level.
		1	High level.
KE_enable*	0	0	Klasse Eins disabled.
		1	Klasse Eins enabled.
qz_ratio*	0	0 to 255	Quiet zone ratio value.
init_scan_time*	0	0 to 65	Time in seconds.
pulse_delay*	0	0 to 65535	Time in milliseconds.

Table 4-4. Scan Parameters

***Section:* [Scan_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
bidir_redundancy	0	0	Disabled.
		1	Enabled.
xmit_code_id	0	0	Disabled.
		1	Transmit Symbol Technologies Code ID.
		2	Transmit AIM Code ID. Note: This value for the xmit_code_id parameter is included for future expansion.

Table 4-5. Decoder Parameters

***Section:* [Decoder_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
UPCE0_minlength	0	0 to 65535	Minimum length number of characters.
UPCE0_maxlength	0	0 to 65535	Maximum length number of characters.
UPCE0_ret_CD	0	0	Do not return check digit.
		1	Return check digit.
UPCE0_preamble	0	0	No number system, no country code.
		1	No number system, country code.
		2	Number system, country code.
UPCE0_convert	0	0	Do not convert UPC E0 to UPC A.
		1	Convert UPC E0 to UPC A.
UPCE1_minlength[#]	0	0 to 65535	Minimum length number of characters.
UPCE1_maxlength[#]	0	0 to 65535	Maximum length number of characters.

Table 4-5. Decoder Parameters (Continued)

***Section:* [Decoder_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
UPCE1_ret_CD#	0	0	Do not return check digit.
		1	Return check digit.
UPCE1_preamble#	0	0	No number system, no country code.
		1	No number system, country code.
		2	Number system, country code.
UPCE1_convert#	0	0	Do not convert UPC E1 to UPC A.
		1	Convert UPC E1 to UPC A.
UPCA_minlength	0	0 to 65535	Minimum length number of characters.
UPCA_maxlength	0	0 to 65535	Maximum length number of characters.
UPCA_ret_CD	1	0	Do not return check digit.
		1	Return check digit.
UPCA_preamble	1	0	No number system, no country code.
		1	No number system, country code.
		2	Number system, country code.
MSI_minlength	4	0 to 65535	Minimum length number of characters.
MSI_maxlength	55	0 to 65535	Maximum length number of characters.
MSI_redundancy	0	0	Disabled.
		1	Enabled.
MSI_CD	1	1 to 2	Number of check digits.
MSI_ret_CD	0	0	Do not return check digit.
		1	Return check digit.
EAN8_minlength	0	0 to 65535	Minimum length number of characters.
EAN8_maxlength	0	0 to 65535	Maximum length number of characters.

Table 4-5. Decoder Parameters (Continued)

***Section:* [Decoder_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
EAN8_convert	0	0	Do not convert EAN 8 to EAN 13.
		1	Convert EAN 8 to EAN 13.
EAN13_minlength	0	0 to 65535	Minimum length number of characters.
EAN13_maxlength	0	0 to 65535	Maximum length number of characters.
Codabar_minlength	0	0 to 65535	Minimum length number of characters.
Codabar_maxlength	0	0 to 65535	Maximum length number of characters.
Codabar_redundancy	0	0	Disabled.
		1	Enabled.
Codabar_CLSI	0	0	Disabled.
		1	Enabled.
Codabar_NOTIS	0	0	Disabled.
		1	Enabled.
Code39_minlength	0	0 to 65535	Minimum length number of characters.
Code39_maxlength	0	0 to 65535	Maximum length number of characters.
Code39_CD	0	0	Disabled.
		1	Enabled.
Code39_concat	0	0	Disabled.
		1	Enabled.
Code39_full_ASCII	0	0	Disabled.
		1	Enabled.
Code39_redundancy	0	0	Disabled.
		1	Enabled.
D2of5_minlength[#]	0	0 to 65535	Minimum length number of characters.

Table 4-5. Decoder Parameters (Continued)

***Section:* [Decoder_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
D2of5_maxlength#	14	0 to 65535	Maximum length number of characters.
D2of5_redundancy#	0	0	Disabled.
		1	Enabled.
I2of5_minlength	14	0 to 65535	Minimum length number of characters.
I2of5_maxlength	10	0 to 65535	Maximum length number of characters.
I2of5_redundancy	0	0	Disabled.
		1	Enabled.
Code11_minlength#	4	0 to 65535	Minimum length number of characters.
Code11_maxlength#	55	0 to 65535	Maximum length number of characters.
Code11_redundancy#	0	0	Disabled.
		1	Enabled.
Code11_CD#	1	0 to 2	Number of check digits.
Code11_ret_CD#	0	0	Do not return check digit.
		1	Return check digit.
Code93_minlength#	0	0 to 65535	Minimum length number of characters.
Code93_maxlength#	0	0 to 65535	Maximum length number of characters.
Code93_redundancy#	0	0	Disabled
		1	Enabled.
Code128_minlength	0	0 to 65535	Minimum length number of characters.
Code128_maxlength	0	0 to 65535	Maximum length number of characters.
Code128_redundancy	0	0	Disabled.
		1	Enabled.

Table 4-6. UPC/EAN General Parameters

***Section:* UPC/EAN General Parameters**			
Keyword	**Default**	**Values**	**Value Definition**
security_level	0	0	No security checking.
		1	Check ambiguous characters.
		2	Check all characters.
supp_2	0	0	Disabled.
		1	Enabled.
supp_5	0	0	Disabled.
		1	Enabled.
supp_autodiscriminate	0	0	Disabled.
		1	Enabled.
supp_retry	5	2 to 10	Retry count before reporting.
linear_decode	0	0	Disabled.
		1	Enabled.

Table 4-7. Trigger Mode Parameters

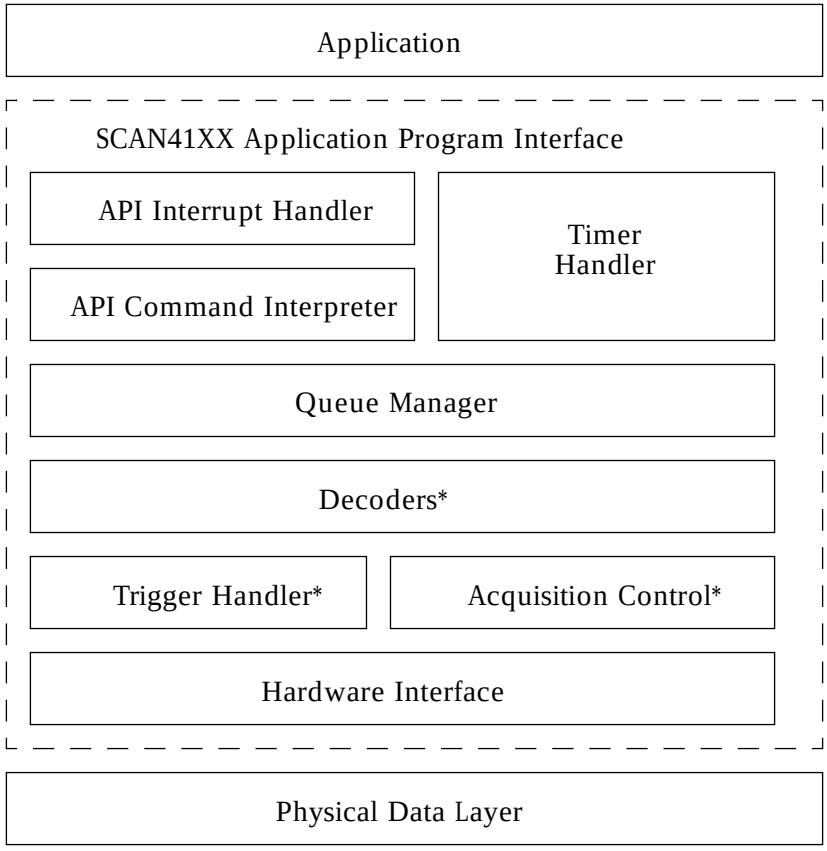
***Section:* [Trigger_Mode_Parameters]**			
Keyword	**Default**	**Values**	**Values**
trigger_mode	2	0	Ignore external trigger.
		1	Reserved.
		2	Enable laser when trigger pressed.

Table 4-8. Decoder Enable Parameters

***Section:* [Decoder_Enable_Parameters]**			
Keyword	**Default**	**Values**	**Value Definition**
UPCE0	1	0	Disabled.
		1	Enabled.
UPCE1[#]	0	0	Disabled.
		1	Enabled.
UPCA	1	0	Disabled.
		1	Enabled.
MSI	1	0	Disabled.
		1	Enabled.
EAN8	1	0	Disabled.
		1	Enabled.
EAN13	1	0	Disabled.
		1	Enabled.
Codabar	1	0	Disabled.
		1	Enabled.
Code39	1	0	Disabled.
		1	Enabled.
D2of5[#]	0	0	Disabled.
		1	Enabled.
I2of5	1	0	Disabled.
		1	Enabled.
Code11[#]	0	0	Disabled.
		1	Enabled.
Code93[#]	0	0	Disabled.
		1	Enabled.
Code128	1	0	Disabled.
		1	Enabled.

Theory of Operation

Figure 4-1 shows the software layers of the SCAN41XX driver from the point of view of its interface with an application.



* SCAN4100 only.

Figure 4-1. SCAN41XX Software Layers

The dashed line box encloses the program modules that carry out the driver's tasks. The *Physical Data Layer* is the link layer between SCAN41XX and the SE-1000 scan module.

Initialization of the scanner driver:

- sets up the interrupt routine
- acquires a timer for the timer handler routine
- initializes queue management and bar code variables to default values
- processes the user configuration file

The *SCAN41XX Application Program Interface (API)* module processes application requests and passes them down to lower layers of the driver. The application communicates with the SCAN41XX by way of Scanner Control Blocks (SCBs) and a dedicated interrupt. The *API Interrupt Handler* expects the register pair ES:BX to point to the SCB that the application wants it to process. See *Supported API Commands* for the commands delivered in SCB structures and supported by SCAN41XX.

After a command is validated, the *API Command Interpreter* either submits the command for immediate processing or places the SCB into the First In First Out (FIFO) Command Queue. Refer to *User Interface* for a description of *immediate* and *queued* commands.

The *Timer Handler* module dispatches non-immediate requests from the SCB Command Queue. After completion of the driver request, the SCB is moved from the Command Queue to the FIFO SCB Complete Queue. Post processing routines are dispatched from the SCB Complete Queue one at a time and the next routine is not dispatched until the current post processing routine has completed. This prevents re-entrancy problems from occurring on the application side (for example, two **Read** commands may use the same post processing routine). When the post processing routine is complete, return status conditions are set, and the SCB is removed from the SCB Complete Queue.

The *Queue Manager* keeps track of the head and tail of the SCB Command and Complete queues. It places new requests at the head of the queue, and processes commands from the tail of the queue. The queues are circular and can contain up to 65,536 SCB requests. The size of the queue is large enough to allow for future PDF 417 expansion capabilities. If a queue becomes full, queue full status is reported to the application, and the driver rejects subsequent SCBs until space exists in the queue.

Table 4-9 lists the decoder types supported by SCAN41XX.

Table 4-9. Decoder Types Supported by SCAN41XX

Decoder	Type Number
UPC E0	0x30
UPC E1*	0x31
UPC A	0x32
MSI	0x33
EAN 8	0x34
EAN 13	0x35
Codabar	0x36
Code 39	0x37
D 2 of 5*	0x38
I 2 of 5	0x39
Code 11*	0x3A
Code 93*	0x3B
Code 128	0x3C
Iata 2 of 5*	0x3E
EAN 128*	0x3F

* Not supported by SCAN4122.

For differences between the API for SCAN41XX (see *Supported API Commands*) and the Series 3000 API, see *Differences Between SCAN3000 and SCAN41XX*.

User Interface

Introduction and Overview

Application programs running on PPT 41XX terminals make requests to SCAN41XX through a dedicated software interrupt. A scanning application loads a pointer to the start of a Scanner Control Block (SCB) into the register pair ES:BX and issues an INT 0x62.

There are two types of SCBs: *Immediate* and *Queued*. The queued type is subdivided into *Wait* and *No Wait*. The differences between immediate commands and wait or no wait commands are described as follows:

- An *immediate* command is processed immediately and is *not* placed in the SCB command pending queue. Control returns to the application program after the command is complete. Table 4-10 lists the immediate commands supported by SCAN41XX and the hexadecimal numerals for the command codes applications must use to invoke them. See *Supported API Commands* for a description of these commands.

Table 4-10. SCAN41XX Supported Immediate Commands

Command	Command Code
Get Version Information	0x00
Get Reader Parameters	0x01
Set Reader Parameters	0x02
Get Scan Parameters	0x03
Set Scan Parameters	0x04
Get Decoder Parameters	0x05
Set Decoder Parameters	0x06
Get UPC/EAN General Parameters	0x07
Set UPC/EAN General Parameters	0x08
Get Scan Status	0x0B
Get Trigger Mode	0x0C
Set Trigger Mode	0x0D
Set Soft Trigger	0x0F
Clear Soft Trigger	0x10
Get Supported Decoders	0x11

Table 4-10. SCAN41XX Supported Immediate Commands (Continued)

Command	Command Code
Get Enabled Decoders	0x12
Set Enabled Decoders	0x13
Cancel SCB	0x14
Flush	0x15
Get Number of Pending SCBs	0x16

- A *no wait* command is placed in the SCB command pending queue and is handled on a timer tick basis by the timer handler routine.
- A *wait* command is placed in the SCB command queue and control is not released until all previous no wait commands and this wait command are complete.

Table 4-11 lists the queued commands (*wait* and *no wait*) supported by SCAN41XX. For descriptions for these commands see *Supported API Commands*.

Table 4-11. SCAN41XX Supported Queued Commands

Command	Command Code
Enable Scanning (Wait)	0x09
Disable Scanning (Wait)	0x0A
Read Label (Wait)	0x0E
Enable Scanning (No Wait)	0x89
Disable Scanning (No Wait)	0x8A
Read Label (No Wait)	0x8E

Upon entry, the interrupt handler expects ES:BX register pair to point to the start of the SCB which will be in the application's data space. This directly monitors and accesses the contents of the SCB. For example, the application can monitor the **status** and **retcode** fields and get data from the data buffer as needed.

Scanner Control Block (SCB)

The Scanner Control Block (**SCB_type** structure) is defined as follows:

```
typedef struct SCB_struc
{
    unsigned char     command;      / / Command code.
    unsigned char     retcode;      / / Return code.
    unsigned char     status;       / / Command completion status.
    unsigned short    timeout;      / / Command timeout
    void far          *cmdparam;    / / Pointer to command parameters.
    unsigned char     user[4];      / / Post processing user parameters.
    void far*         (*process) ( ); / / Post processing dispatch address.
    char              reserved[21]; / / Reserved space.
} SCB_type;
```

Following are definitions of the fields in this SCB structure:

command

Set by the application to the desired sub-command function. In general, control is not returned to the application until the selected command is complete. However, if the most significant bit of the command field is set (NO WAIT), the scanner driver submits the command in the command queue and returns to the application. The timer handler routine processes these queued commands in the background. NO WAIT is ignored by commands that do not support this mode.

retcode

Set by the scanner driver after the command has completed processing. Zero indicates successful completion; non-zero indicates an error condition. Set this field prior to the dispatch of any post processing routine. See Table 4-18 for completion and error codes.

status

While this field is 0xFF, the command is pending or processing and the application must not modify any data within the SCB or associated data buffers. This field is set to the same value as **retcode** after any post processing routine is complete. Application programs should monitor **status** completion of an SCB. Refer to Table 4-18 for a list of possible return values and their meanings.

timeout

Set by the user to indicate, in milliseconds, the maximum time for the scanner driver to wait for the command to complete before reporting an error. If set to zero,

the timeout function is disabled. Queued commands do not begin their associated timeout cycle until commands ahead of them have completed processing.

cmdparam

A far pointer to the parameter structure of the associated command. Not all commands have command parameters. If a command has parameters, see *Supported API Commands* for its parameter structure.

user

This space is reserved for use by the application as a parameter area for the post processing routine. It is defined as 4 bytes to allow a far pointer to be placed in the field if more than 4 bytes of data are required by the post processing routine. The **process** routine extracts the **user** parameters from the SCB. The scanner driver ignores all values placed here.

process

Set by the application, if desired, to the far address of a post processing routine. If it is non-zero, the driver calls the routine after the SCB command is complete. The AL register is set to the SCB **retcode** field, and the address of the calling SCB is placed in the ES:BX register pair prior to the call to the post processing routine. If a post processing routine is *not* required, the **process** field *must be set to zero*; otherwise, an invalid pointer is called producing unpredictable results. Post processing routines must be declared as an *_interrupt* and must return with an *iret*.

reserved

This area is reserved for future expansion of the scanner driver. It should *not* be accessed or altered by any application program.

A Day in the Life of an SCB

Scanner control blocks (SCBs) are the only means of communication between application programs and the PPT 41XX scanner driver. The entry point for this exchange of information is SCANXX's dedicated API interrupt, **INT 0x62**.

Immediate Command Processing

Refer to Figure 4-2 (for Steps 1-4)and Figure 4-3 (for Steps 5-6) as you read through the following description of the life of an immediate SCB:

1. The user application allocates a buffer for the SCB and associated command parameters. The SCB **command**, **timeout**, **cmdparam**, **user**, and **process** fields are set; any command parameters are also set.
2. The application initializes ES:BX to point to the SCB and issues the scanner interrupt, 0x62. The API interrupt handler receives the interrupt in SCANXX.
3. The API command interpreter verifies the SCB and checks that the SCB is not already in use.
4. After the command is processed, the **retcode** field of the SCB is set.
5. SCANXX then sets ES:BX to point to the SCB just processed and loads AL with the contents of the return code. The flags register is pushed onto the stack and the SCB's post processing routine is called. If no post processing routine was specified (i.e., **SCB.process** = NULL), skip to scanner driver processing in Step 6. The post processing routine in the user application performs tasks specific to the completion of the SCB.

   Note: Post processing routines must not make any calls which require DOS. Because this routine is being executed as an interrupt, DOS may not be reentered.

6. When post processing is complete, the user application issues an *iret* to return to the scanner driver which sets the SCB status code (**status**) to the SCB return code (**retcode**). The SCB is completed.

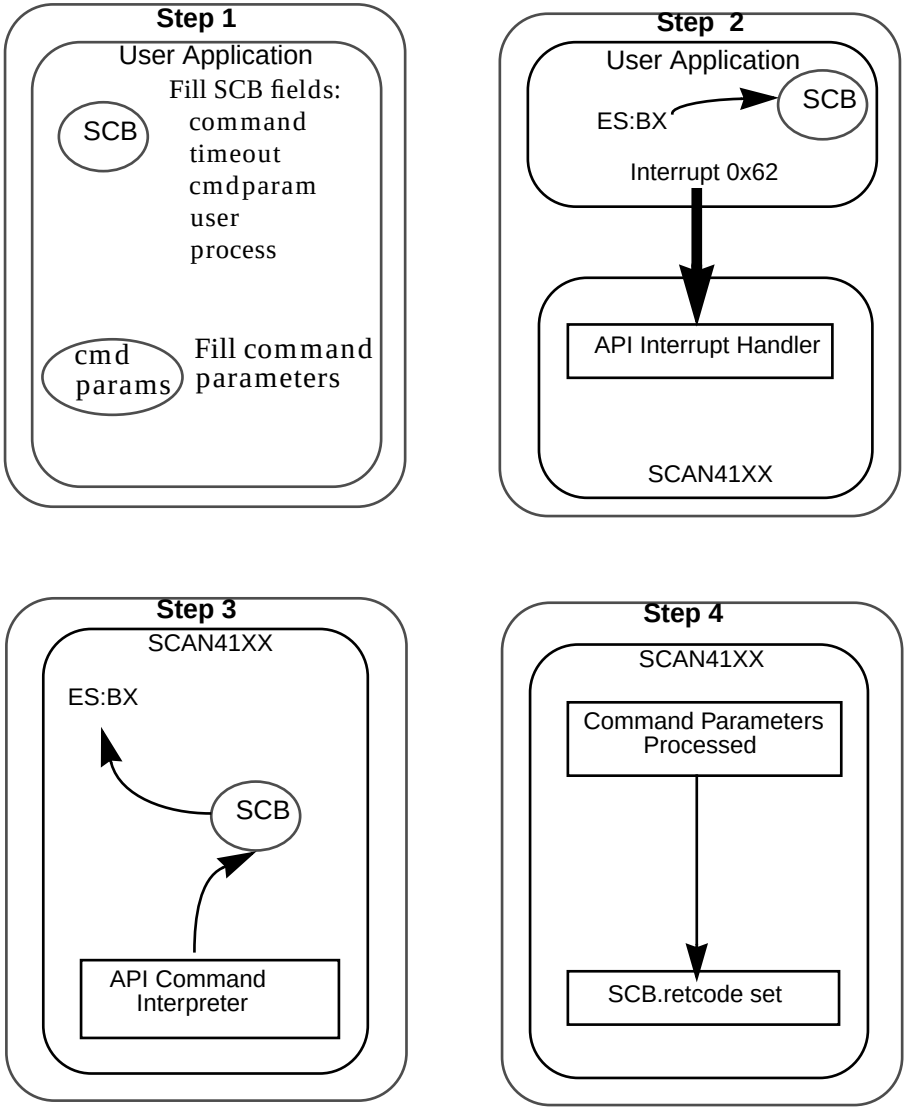


Figure 4-2. Immediate Command Processing (Steps 1-4)

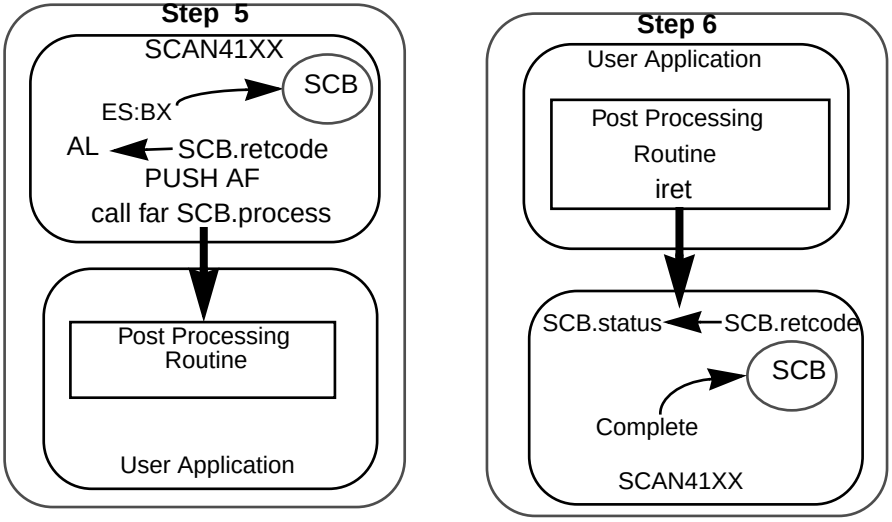


Figure 4-3. Immediate Command Processing (Steps 5-6)

Queued Command Processing

Refer to Figure 4-4 (for Steps 1-4) and Figure 4-5 (for Steps 5-8) as you read through the following description of the life of a queued SCB:

1. The user application allocates a buffer for an SCB and associated command parameters. The SCB **command**, **timeout**, **cmdparam**, **user**, and **process** fields are set; any command parameters are also set.
2. The application initializes ES:BX to point to the SCB and issues the scanner interrupt, 0x62. The API interrupt handler receives the interrupt in SCAN41XX.
3. The API command interpreter verifies the SCB and checks that the SCB is not already in use.
4. After command parameters are verified, the SCB is placed at the tail of the command queue. If the command was specified as *no wait*, control returns to the user application. If command was specified as *wait*, control returns to the user application only after completion of the command.

   Note: The SCB may not be submitted again until it has been removed from the queue.

5. Every 55 milliseconds, XSYMBIOS calls the queued command processor module of SCAN41XX. When the SCB submitted in Step 2 reaches the head of the command queue, it is removed.
6. After the command is processed, the **retcode** field of the SCB is set.
7. SCAN41XX then sets ES:BX to point to the SCB just processed and loads AL with the contents of the return code. The flags register is pushed onto the stack and the SCB's post processing routine is called. If no post processing routine was specified (i.e., **SCB.process** = NULL), skip to SCAN41XX processing in Step 8. The post processing routine in the user application performs tasks specific to the completion of the SCB.

   Note: Post processing routines must not make any calls which require DOS. Because this routine is being executed as an interrupt, DOS may not be reentered. The post processing routine may not call the scanner driver.

8. When post processing is complete, the user application issues an *iret* to return to the scanner driver which sets the SCB status code (**status**) to the SCB return code (**retcode**). The SCB is now complete.

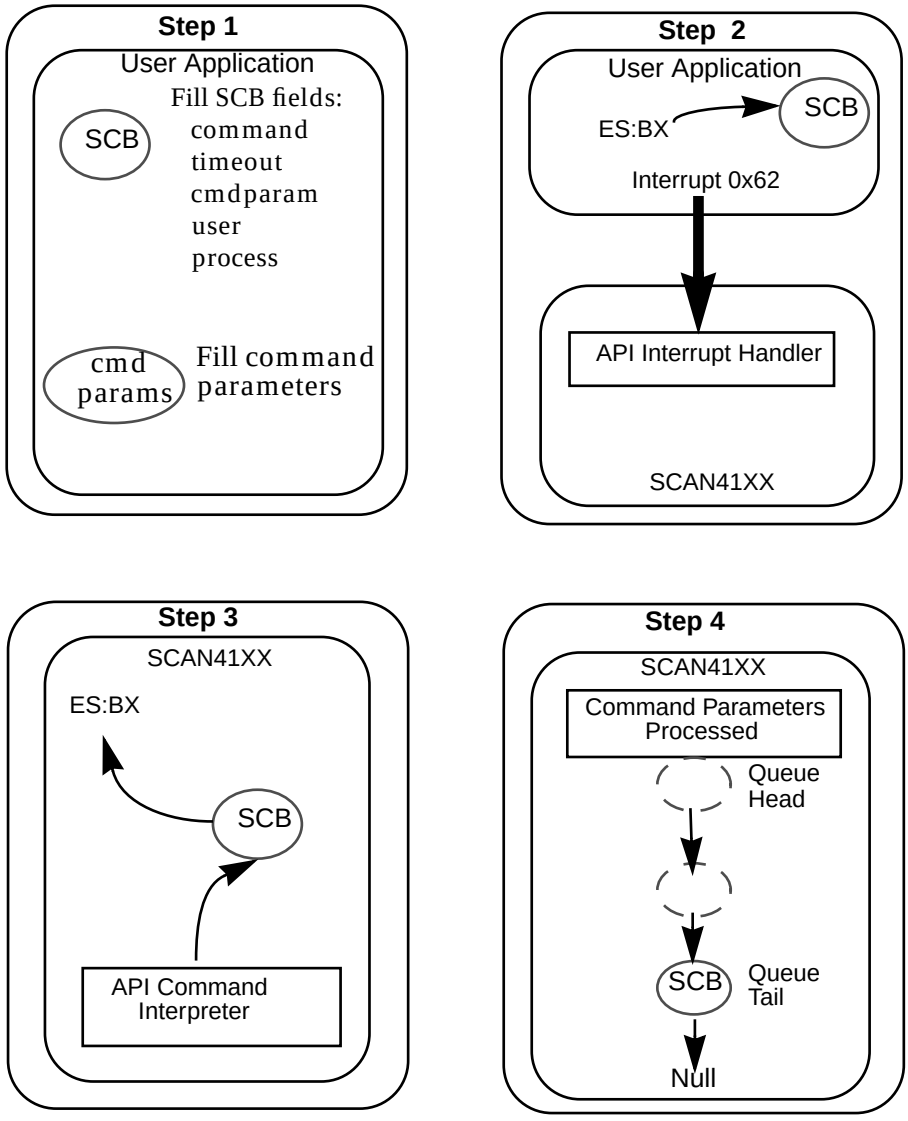


Figure 4-4. Queued Command Processing (Steps 1-4)

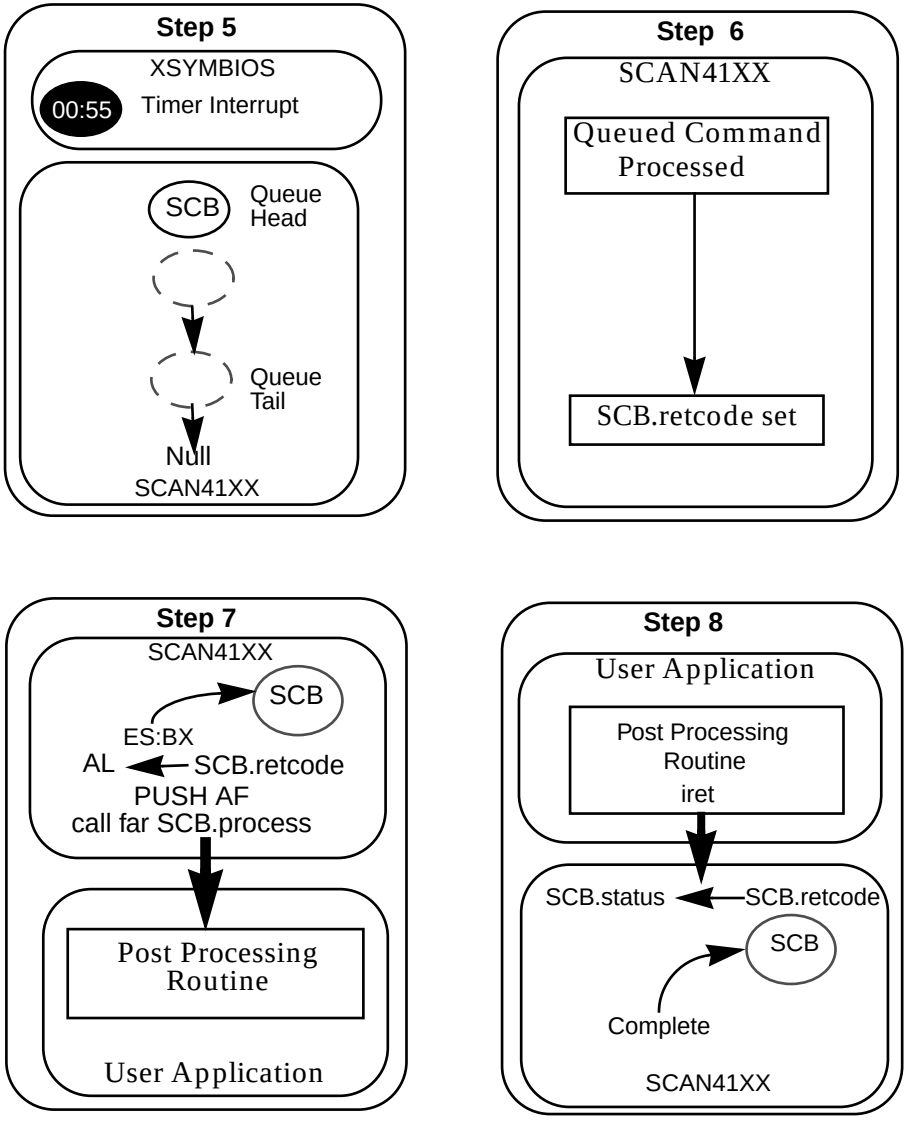


Figure 4-5. Queued Command Processing (Steps 5-8)

Supported API Commands

This section consists of three main parts:

- *SCB Commands (List)* lists the API commands (immediate and wait/no wait) supported by SCAN41XX
- *SCB Commands (Descriptions)* describes each API command supported by SCAN41XX which are available for programmers to use in applications they create for PPT 41XX terminals
- *Completion and Error Codes* lists the error and completion codes that may be returned by SCAN41XX when SCB requests presented by an application are complete

SCB Commands (List)

Table 4-12 is a list of SCB commands supported by SCAN41XX. The list is sorted by the hexadecimal numeral for the function code (command code) the application assigns to the **command** variable in the SCB structure when it invokes the associated scanning service. Descriptions of supported commands and their parameter definitions are provided in the *SCB Commands (Descriptions)* section that follows this table. Queued commands have been specified in the table as "Wait" or "No Wait."

Table 4-12. Scanner Driver SCB Commands (INT 0x62)

Command	Type	Command Code
Get Version Information	Immediate	0x00
Get Reader Parameters	Immediate	0x01
Set Reader Parameters	Immediate	0x02
Get Scan Parameters	Immediate	0x03
Set Scan Parameters	Immediate	0x04
Get Decoder Parameters	Immediate	0x05
Set Decoder Parameters	Immediate	0x06
Get UPC/EAN General Parameters	Immediate	0x07
Set UPC/EAN General Parameters	Immediate	0x08
Enable Scanning	Wait	0x09
Disable Scanning	Wait	0x0A
Get Scan Status	Immediate	0x0B

Table 4-12. Scanner Driver SCB Commands (INT 0x62) (Continued)

Command	Type	Command Code
Get Trigger Mode	Immediate	0x0C
Set Trigger Mode	Immediate	0x0D
Read Label	Wait	0x0E
Set Soft Trigger	Immediate	0x0F
Clear Soft Trigger	Immediate	0x10
Get Supported Decoders	Immediate	0x11
Get Enabled Decoders	Immediate	0x12
Set Enabled Decoders	Immediate	0x13
Cancel Pending SCB	Immediate	0x14
Flush All Pending SCBs	Immediate	0x15
Get Number of Pending SCBs	Immediate	0x16
Enable Scanning	No Wait	0x89
Disable Scanning	No Wait	0x8A
Read Label	No Wait	0x8E

SCB Commands (Descriptions)

The following descriptions of the SCB commands in Table 4-12 are given in function code order.

Each description consists of:

- the command name
- the **Command Code** (hex numeral) specified in the SCB **command** field of the associated SCB data structure.
  See *Scanner Control Block (SCB)* for a definition of the **SCB_type** data structure and descriptions of its fields.

  Note: An application communicates with SCAN41XX via an SCB (in which the value of the **command** field specifies the API command to be processed) and a dedicated software interrupt (INT 0x62). The interrupt handler expects the ES:BX register pair to point to the SCB that the application wants SCAN41XX to process.
- the **Type** (i.e., immediate, wait, or no wait) of SCB associated with the command
- the **Action** taken by the scanner driver when an application invokes the command
- definitions of **Parameters** (if any) and definitions of associated data structures (if any)
- an **Example**, i.e., a code sample to illustrate the use of the command

Get Version Information

Command Code: 0x00

Type: Immediate

Action

Returns SCAN41XX version information.

Parameters

Version information in a data structure defined as follows:

```
typedef struct VERSION_INFO_struc
{
    void far        *version_str;     // Pointer to version string
    unsigned short  version;          // Major and minor version numbers
} VERSION_INFO_type;
```

The fields in this data structure are defined below:

version_buf
Pointer to a null terminated version string. For SCAN41XX the default string is:

> **PPT 41*XX* Scanner Driver Version *x.xx***
> **Copyright (C) Symbol Technologies, Inc.**
> **1993-1997. All rights reserved.**

version
Major and minor revision numbers of the scanner driver as follows:

Upper Byte = Hex numeral for the major revision number
Lower Byte = Hex numeral for the minor revision number

Example

For a code sample illustrating the **Get Version Information** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Reader Parameters

Command Code: 0x01

Type: Immediate

Action

Returns information about the reader parameters based on the current **parameter_index.**

Parameters

Information on reader parameters specified in a data structure defined as follows:

```
struct READER_MODE
{
  unsigned char      parameter_index;      / /Index to the scanner
                                           / / parameters to use
/ / Common reader parameters
  unsigned char      scan_mode;            / / Scanner class type attached
  unsigned short     enable_settle_time;   / / Reader enable settling time
  unsigned short     power_settle_time;    / / Power settling time, used
                                           / / if enable_settle_time > 0
  unsigned char      inverse_label_flag;   / / Inverse data label flag
  unsigned char      white_data_logic_lvl; / / White data logic level
  unsigned short     dec_beep_time;        / / Decode beep time
  unsigned short     dec_beep_freq;        / / Decode beep frequency
  unsigned char      scans_per_label;      / / Number of successful
                                           / / decodes for a label
                                           / / before reporting
  unsigned char      clk_speed_toggle;     / / Clock speed toggle
  unsigned char      trans_resolution;     / / Transition resolution
                                           / / value
  unsigned char      subseq_scan_time;     / / Subsequent scan time
  unsigned char      no_data_time;         / / No data time value
  unsigned char      post_decode_action;   / / Post decode action
  unsigned char      dec_fail_action;      / / Decode fail action

  / / Laser specific parameters
  unsigned char      prod_trigger;         / / Produces trigger signal
  unsigned char      two_stage_trigger;    / / Two stage trigger flag
  unsigned char      multiple_scan;        / / Automatic multiple scan
  unsigned char      prod_direction;       / / Produces direction signal
```

```
    unsigned char     dec_feedback_time:    / / Decode LED feedback
                                            / / time
    unsigned char     dec_feedback_lvl;     / / Decode LED feedback
                                            / / level
    unsigned char     scan_led_ctl;         / / Scanning LED control flag
    unsigned char     scan_led_lvl;         / / Scanning LED logic level
    unsigned char     KE_enable;            / / Enable Klasse Eins flag
    unsigned short    KE_time_used;         / / Klasse Eins time used
    unsigned short    KE_time_left;         / / Klasse Eins time left

    / / Contact specific parameters
    unsigned char     qz_ratio;             / / Quiet zone ratio X:1 to 1:X.
    unsigned char     init_scan_time;       / / Initial scan time.
    unsigned short    pulse_delay;          / / Pulse delay time.
} READER_MODE_type;
```

The fields in this data structure are defined below:

parameter_index
Index into reader parameter table to get/set. There may be up to ten reader parameter settings. If this value is set to 0xFF, the driver autodiscriminates between the contact settings in index 1 and the laser settings in index 2. These indices are preset to the default values for a contact wand and a laser gun, respectively. Index 3 has parameters set for SCAN4100. Index 4 has parameters set for SCAN4122. Indices 1 and 2 are not supported in the first revision. If this value is 0, the current active parameters are get/set. This field must be set for parameters to be get or set. Following are the values for this field and their definitions:

Value	Definition
0	Current Parameters
1	Contact Parameters
2	Laser Parameters
3	SE-1000 Scan Module Card (SMC) Parameters
4	SE-1022 Parameters
5 to 10	User Definable Parameters
255	Autodiscriminate between contact and laser

Common Reader Parameters

Note: Parameters followed by an asterisk (*) in the following list are provided for future expansion and are not

supported in the first version of SCAN41XX.

scan_mode
Describes the scanner type associated with the current **parameter_index**. Valid values are:

1 = Contact wand (set common and contact parameters)

2 = Generic laser gun (set common and laser parameters)

3 = CCD gun (set all reader parameters)

4 = Scan Module Card (set common and laser parameters)

5 = SE-1022 (set common and laser parameters)

6 = Undefined scanner (set all reader parameters)

enable_settle_time*
If the scanner has enable line, this is the time in microseconds after enable is applied that the terminal waits before accessing the scanner.

If the time is set to zero, the scanner does not use enable line. If the time is set to 0xFFFF, the enable line is enabled immediately and remains on while scanning is enabled.

power_settle_time*
If the scanner has enable line, this is the time in microseconds that the terminal waits after power is applied to the scanner before asserting the enable line.

If time is set to zero, the scanner does not use enable line. If time is set to 0xFFFF, the power line is enabled immediately and remains on while scanning is enabled.

inverse_label_flag*
This value indicates whether or not labels are being printed in normal or inverse mode. Valid values are:

0 = Normal (black on white)
1 = Inverse (white on black)

white_data_logic_lvl*
This value indicates the logic level the data line is at when the scanner is detecting white. Valid Values are:

0 = Low level
1 = High level

dec_beep_time
Zero specifies that no decode beep is used. A non-zero value specifies the duration, in milliseconds, of the good decode beep.

dec_beep_freq
Beeper frequency in megahertz if **dec_beep_time** is non-zero.

scans_per_label*
Number of successful decodes for a label before it is reported to the application.

clk_speed_toggle*
Clock speed toggle. Valid values are:

0 = Do not toggle

1 = Toggle

trans_resolution*
Specifies the clock rate divisor *n* to obtain data sampling rate. The formula for the resulting sample rate is:

Sample Rate = System Clock / *n*, where *n* = 2, 4, 8, or 16

subseq_scan_time*
Maximum scan time in seconds that the reader remains powered up in the acquire mode waiting for a bar code after a successful decode.

no_data_time
The amount of time to wait for a decode after the scanner is enabled (most commonly known in other Symbol products as *laser on* time). After this time interval, **dec_fail_action** is taken.

post_dec_action*
Reader state to switch to after a successful decode attempt. Valid values are:

0 = pulse / wait	Contact wands pulse; laser guns wait for a trigger (same as acquire)
1 = acquire	Contact wands go to full illumination and look for a bar code; laser guns attempt to scan.
2 = stop	Contact wands are powered off; laser triggers are ignored.

dec_fail_action*
Reader state to switch to after an unsuccessful decode attempt. Valid values are those specified above in **post_dec_action**.

Figure 4-5 shows a state diagram illustrating reader state transitions.

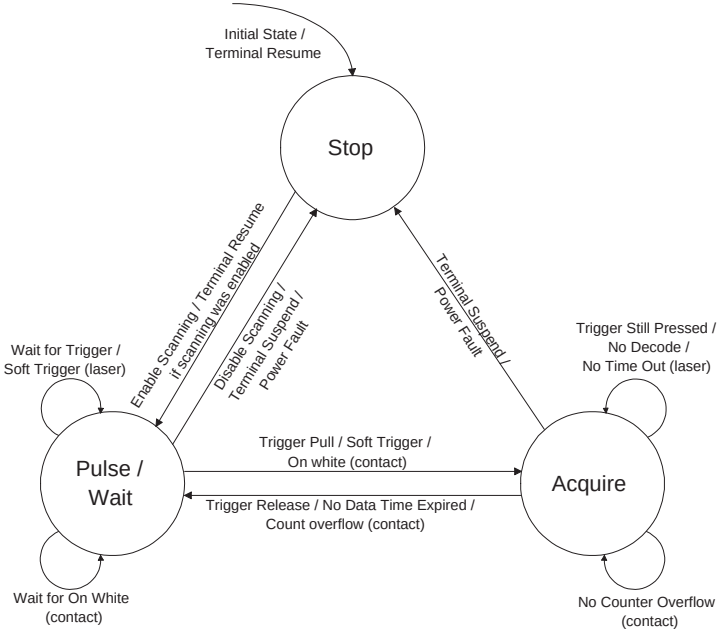


Figure 4-6. Reader State Transition Diagram

Laser Specific Parameters

Note: Parameters followed by an asterisk (*) in the following list are provided for future expansion and are *not* supported in the first version of SCAN41XX.

prod_trigger*
Indicates whether or not the scanner type defined for **scan_mode** produces a trigger signal. Valid values are:

0 = Does not trigger
1 = Can trigger

two_stage_trigger*
Indicates whether or not the scanner has a two-stage trigger. Valid values are:

0 = Does not have a two-stage trigger
1 = Has a two-stage trigger

multiple_scan*
Indicates whether or not the scanner type defined for **scan_mode** allows automatic multiple scanning. Valid values are:

0 = Does not multiple scan
1 = Does multiple scan

prod_direction*
Indicates whether or not the scanner type defined for **scan_mode** produces a direction signal. Valid values are:

0 = Does not give direction
1 = Gives direction

dec_feedb_time
Decode feedback time in seconds to illuminate the good decode LED. If the value is zero, the reader has no good decode LED.

dec_feedback_lvl
If **dec_feedb_time** is non-zero, this describes the line assertion value of the decode LED. Valid values re:

0 = Low level
1 = High level

scan_led_ctl*
Scanning LED control value. This value determines whether or not the driver can control the scanning LED. Valid values are:

0 = Cannot control scanning LED
1 = Can control scanning LED

scan_led_lvl*
If the value of **scan_led_ctl** is non-zero, this parameter indicates the line assertion

level of the scanning LED. Valid values are:

0 = Low level
1 = High level

KE_enable*
Specifies whether Klasse Eins laser on time function is enabled. Valid values are:

0 = Klasse Eins disabled
1 = Klasse Eins enabled

KE_time_used*
Amount of scan time used in seconds of scanning for Klasse Eins mode. This is a read-only value.

KE_time_left*
Amount of scan time left in seconds for scanning for Klasse Eins mode. This is a read-only value.

Contact Specific Parameters

Note: Parameters that are followed by an asterisk (*) in the following list are provided for future expansion and are not supported in the first version of SCAN41XX.

qz_ratio*
Quiet zone ratio X:1 to 1:X. This parameter applies only to contact wand data.

init_scan_time*
Maximum scan time in seconds that the reader remains powered up in the acquire mode waiting for a bar code. If this time expires, the **dec_fail_action** is taken.

Pulse_delay*
Pulse delay time in milliseconds to pause between pulses of a contact wand.

Example

For a code sample illustrating the **Get Reader Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set Reader Parameters

Command Code: 0x02

Type: Immediate

Action

Sets information about the reader parameters based on the value passed in **parameter_index**.

> **Note:** This command should not be issued if the command queue is not empty.

Parameters

Information on reader parameters specified in a data structure. See **Get Reader Parameters** (**0x01**) for a definitions of the data structure and each of its fields.

Example

For a code sample illustrating the **Set Reader Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Scan Parameters

Command Code: 0x03

Type: Immediate

Action

Gets basic scanning parameters.

Parameters

Information on scanning parameters from a data structure defined as follows:

```
typedef struct SCAN_PARMS_struc
{
  unsigned char    bidir_redundancy;    / / Bi-directional redundancy.
  unsigned char    xmit_code_id;        / / Transmit code id character
                                        / / before decoded data.
} SCAN_PARMS_type;
```

The fields in this structure are defined below:

bidir_redundancy
Sets bidirectional redundancy status of decoders. For decoders that have simple redundancy enabled, each decoded scan must come from opposite directions. Valid values are:

0 = No bidirectional redundancy
1 = Bidirectional redundancy

xmit_code_id
Indicates whether or not to append the code identifier character or characters before the decoded data prior to transmission. Valid values are:

0 = Do not transmit code identifier
1 = Transmit Symbol Technologies code identifier (see Table 4-13)
2 = Transmit AIM (Automatic Identification Manufacturers) identifier (see Table 4-14).

Note: AIM identifiers are not supported in the first version of SCAN41XX.

Table 4-13. Symbol Technologies Code Identifiers

Bar Code Symbology	ID Character
UPC/EAN	A
Code 39	B
Codabar	C
Code 128	D
Code 93	E
Interleaved 2 of 5	F
Discrete 2 of 5, Iata 2 of 5	G
Code 11	H
MSI	J
EAN 128	K
PDF 417	X

The Automatic Identification Manufacturers (AIM) have specified an industry-wide standard for bar code identifiers. These code identifiers are a three-character string preamble consisting of:

- a right bracket character (])
- an alpha character to specify the bar code symbology
- a numeric character to specify the decode option used to decode the bar code

The AIM code identifiers are shown in Table 4-14.

.

Table 4-14. AIM Code Identifiers

Bar Code Symbology	ID String	Description
UPC/EAN	]E0	Standard data packet in full EAN code format, i.e., 13 digits for EAN 13, UPC-A, and UPC-E. (Does not include supplemental data.)
	]E1	Two-digit supplemental data only (transmitted separately from 13-digit UPC/EAN data packet).
	]E2	Five-digit supplemental data only (transmitted separately from 13-digit UPC/EAN data packet).
	]E3	Combined data packet comprising 13 digits from EAN-13, UPC-A and UPC-E symbol and two or five digits from supplementary data.
	]E4	EAN-8 data packet.
	]E8	UPC-D3 data packet.
Code 39	]A0	No check character or full ASCII processing.
	]A1	Modulo 43 check character has been validated.
	]A2	Modulo 43 check character has been stripped.
	]A3	Modulo 43 check character has been validated and stripped.
	]A4	Full ASCII character conversion was made.

Table 4-14. AIM Code Identifiers (Continued)

Bar Code Symbology	ID String	Description
Code 39 (Continued)	]A5	Full ASCII character conversion was made and modulo 43 check character has been validated.
	]A6	Full ASCII character conversion was made and modulo 43 check character has been stripped.
	]A7	Full ASCII character conversion was made and modulo 43 check character has been validated and stripped.
Codabar	]F0	No options specified at this time.
Code 128	]C0	Standard data packet. No Function Code 1 in first or second character position after start character.
	]C1	EAN-128 symbol - Function code 1 in first symbol character position after start character.
	]C2	Function code 1 in second symbol character position after start character.
	]C4	ISBT 128 concatenation has been performed.
Code 93	]G0	No options specified at this time.
Interleaved 2 of 5	]I0	No check digit processing.
	]I1	Modulo 10 check character has been validated.
	]I2	Modulo 10 check character has been stripped.
	]I3	Modulo 10 check character has been validated and stripped.

Table 4-14. AIM Code Identifiers (Continued)

Bar Code Symbology	ID String	Description
Discrete 2 of 5	]R0	No check digit processing.
	]R1	Modulo 7 check character has been validated.
	]R2	Modulo 10 check character has been stripped.
	]R3	Modulo 10 check character has been validated and stripped.
Iata 2 of 5	]S0	No options specified at this time
Code 11	]H0	Single modulo 11 check character has been validated.
	]H1	Two modulo 11 check characters have been validated.
	]H2	All check character(s) have been stripped.
	]H3	Two modulo 11 check characters have been validated and stripped.
MSI	]M0	Single modulo 10 check character has been validated.
	]M1	Two modulo 10 check characters have been validated.
	]M2	All check character(s) have been stripped.
	]M3	Two modulo 10 check characters have been validated and stripped.

Example

For a code sample illustrating the **Get Scan Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set Scan Parameters

Command Code: 0x04

Type: Immediate

Action

Sets basic scanning parameters.

> **Note:** Do not issue this command if the command queue is not empty.

Parameters

Information on scanning parameters specified in a data structure. See **Get Scan Parameters** (**0x03**) for definitions of the data structure and each of its fields.

Example

For a code sample illustrating the **Set Scan Parameters** command, refer to the sample scanning program (SCANSAMP.C) listed in *Appendix 1* of this chapter.

Get Decoder Parameters

Command Code: 0x05

Type: Immediate

Action

Returns the information about the decoder settings of the selected bar code type (symbology).

Parameters

The decoder parameters are those (see **Note** below) returned in a data structure defined as follows:

```
typedef struct DECODER_PARMS_struc
{
    unsigned char       labeltype;          / / Bar code symbology
                                            / / type
    unsigned short      minlength;          / / Minimum length of
                                            / / barcode.
    unsigned short      maxlength;          / / Maximum length of
                                            / / barcode.
    unsigned char       alloc_specific;     / / Number of allocated
                                            / / parameters in
                                            / / decoder_specific
    char                decoder_specific[20]; / / Symbology specific
                                            / / variables.
} DECODER_PARMS_type;
```

When getting decoder parameters, the application must specify only the desired labeltype and alloc_specific; the remaining settings in this data structure are returned.

The fields in this structure are defined below:

labeltype
The bar code symbology type value to get decoder information.Valid ASCII char values for **labeltype** and symbology type numbers (as hexadecimal numerals) are given in the following table:

Table 4-15. labeltype Values for DECODER_PARMS_struc

Bar Code Type	ASCII char Value for labeltype	Hex Value
UPC E0	0	0x30
UPC E1 #	1	0x31
UPC A	2	0x32
MSI	3	0x33
EAN 8	4	0x34
EAN 13	5	0x35
Codabar	6	0x36
Code 39	7	0x37
D 2 of 5 #	8	0x38
I 2 of 5	9	0x39
Code 11 #	:	0x3A
Code 93 #	;	0x3B
Code 128	<	0x3C

Note: Bar code types followed in this table by a # are not supported by SCAN4122.

minlength, maxlength

Minimum and maximum allowable decode lengths. Length specification is:

Variable
minlength = maxlength = 0

Range (from **a** to **b**, including **a** and **b**)
minlength = a
maxlength = b

Dual (either **a** or **b**, given **a <b**)
minlength =a
maxlength = b

Single (only **a**)
minlength = maxlength = a

Special Cases:

For Code 128, the specified lengths are the number of data characters encoded in the bar code, *not* the number of human readable characters.

For Codabar, the specified lengths include the start and stop characters.

For UPC/EAN types, lengths specified are ignored.

For Interleaved 2 of 5 and SCAN4122, the maximum length that may be decoded is 30 characters and only dual length mode is supported.

alloc_specific
The number of parameters in **decoder_specific** that are used. This value must be supplied to Get and Set commands. The returned value is the number of parameters that were Set or Get.

decoder_specific
Each symbology has its own specific variables to control functions within the symbology's decoder. These parameters map directly to the *_param* structure used by the 16-bit decoder package. These variables are described below:

UPC E0	*byte 0*	Return check digit. Valid values are:

0 = do not return
1 = return

byte 1 UPC E0 preamble. Valid values are:

0 = no country code, no number system
1 = no country code, number system
2 = country code, number system

byte 2 Convert UPC E0 to UPC A. Valid values are:

0 = do not convert
1 = convert

UPC E1 # *byte 0* Return check digit. Valid values are:

0 = do not return
1 = return

byte 1 UPC E1 preamble. Valid values are:

0 = no country code, no number system
1 = no country code, number system
2 = country code, number system

byte 2 Convert UPC E1 to UPC A. Valid values are:

0 = do not convert
1 = convert

UPC A *byte 0* Return check digit. Valid values are:

0 = do not return
1 = return

byte 1 UPC A preamble. Valid values are:

0 = no country code, no number system
1 = no country code, number system
2 = country code, number system

MSI *byte 0* Redundancy. Valid values are:

0 = simple redundancy disabled
1 = simple redundancy enabled

	byte 1	Number of check digits. Valid values are: 1 = One check digit 2 = Two check digits
	byte 2	Return check digit. Valid values are: 0 = do not return 1 = return
EAN 8	*byte 0*	Convert EAN 8 to EAN 13. Valid values are: 0 = do not convert 1 = convert
EAN 13	There are no associated decoder specific parameters.	
Codabar	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
	byte 1	CLSI formatting. Valid values are: 0 = return with no CLSI formatting 1 = return with CLSI formatting
	byte 2	NOTIS formatting. Valid values are: 0 = return with no NOTIS formatting 1 = return with NOTIS formatting
Code 39	*byte 0*	Code 39 check digit. Valid values are: 0 = disable Code 39 check digit 1 = enable Code 39 check digit
	byte 1	Code 39 concatenation. Valid values are: 0 = disable Code 39 concatenation 1 = enable Code 39 concatenation
	byte 2	Code 39 Full ASCII. Valid values are: 0 = disable Code 39 full ASCII 1 = enable Code 39 full ASCII

	byte 3	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
D 2 of 5 #	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
I 2 of 5	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
Code 11 #	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
	byte 1	Number of check digits. Valid values are: 0 = No check digits 1 = One check digit 2 = Two check digits
	byte 2	Return check digit. Valid values are: 0 = do not return 1 = return
Code 93 #	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
Code 128	*byte 0*	Redundancy. Valid values are: 0 = simple redundancy disabled 1 = simple redundancy enabled
Iata 2 of 5 #	Uses Discrete 2 of 5's **minlength**, **maxlength**, and **decoder_specific** settings. It may *not* be enabled / disabled.	
EAN 128	Uses Code 128's **minlength**, **maxlength**, and **decoder_specific** settings. It may *not* be enabled / disabled.	

Example

For a code sample illustrating the **Get Decoder Parameters** command, refer to the sample scanning program (SCANSAMP.C) listed in *Appendix 1* of this chapter.

Set Decoder Parameters

Command Code: 0x06

Type: *Immediate*

Action

Sets information for the decoder settings of the selected bar code type.

> **Note:** This command should not be issued if the command queue is not empty.

Parameters

Information on decoder parameters specified in a data structure. See **Get Decoder Parameters** (**0x05**) for a definition of the data structure and definitions of its fields.

Example

For a code sample illustrating the **Set Decoder Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get UPC/EAN General Parameters

Command Code: 0x07

Type: Immediate

Action

The UPC/EAN symbology family has several general parameters that affect all functions and capabilities of UPC/EAN decoders. This command returns the current setting for these parameters.

Parameters

The UPC/EAN general parameters contained in a data structure defined as follows:

```
typedef struct UPC_GEN_PARMS_struc
{
  unsigned char     security_level;   / / Security level value
  unsigned char     supp_2;           / / Two digit supplementals
  unsigned char     supp_5;           / / Five digit supplementals
  unsigned char     supp_auto_d;      / / Auto discriminate
                                      / / supplemental bar
                                      / / codes.
  unsigned char     supp_retry;       / / Retry count for auto
                                      / / discriminate
                                      / / supplementals.
  unsigned char     linear_decode;    / / Linear decode all UPC types.
} UPC_GEN_PARMS_type;
```

The fields in this structure are defined below:

security_level
Decoder security level value. This parameter tightens the decoder algorithms to prevent misreads on poorly printed UPC/EAN labels. Valid values are:

0 = No security checking
1 = Check ambiguous characters
2 = Check all characters

supp_2
This parameter causes all UPC/EAN type bar codes either to read or not read 2-digit supplementals. Valid values are:

0 = Do not decode 2-digit supplementals
1 = Decode 2-digit supplementals

supp_5
This parameter causes all UPC/EAN type bar codes either to read or not read 5-digit supplementals. Valid values are:

0 = Do not decode 5-digit supplementals
1 = Decode 5-digit supplementals

supp_auto_d
Autodiscriminate supplemental bar codes. Valid values are:

0 = Do not autodiscriminate supplementals
(i.e., ignore supplementals)
1 = autodiscriminate supplementals

supp_retry
If **supp_auto_d** is 1, retry count from 2 to 10 before reporting to ensure that no supplemental bar code follows a UPC/EAN bar code.

linear_decode
Perform linear decode on all UPC/EAN types. This increases security in decoding when multiple UPC/EAN labels are scanned at one time. Valid values are:

0 = Do not perform linear decode
1 = Perform linear decode

Example

For a code sample illustrating the **Get UPC/EAN General Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set UPC/EAN General Parameters

Command Code: 0x08

Type: Immediate

Action

This command sets the UPC/EAN general decoder parameters.

Note: This command should not be issued if the command queue is not empty.

Parameters

Information on decoder parameters specified in a data structure. See **Get UPC/EAN General Parameters** (**0x07**) for a definition of the data structure and definitions of its fields.

Example

For a code sample illustrating the **Set UPC/EAN General Parameters** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Enable Scanning with Wait

Command Code: 0x09

Type: Wait

Note: There is a No Wait version of this command (Command Code: 0x89).

Action

Enables scanning in SCAN41XX. All physical and soft triggers are processed.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Enable Scanning with Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Disable Scanning with Wait

Command Code: 0x0A

Type: Wait

Note: There is a No Wait version of this command (Command Code: 0x8A).

Action

Disables scanning SCAN41XX. All physical and soft triggers are ignored and all read command requests are rejected.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Disable Scanning with Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Scan Status

Command Code: 0x0B

Type: Immediate

Action

Gets the current scanning status of SCAN41XX. See Figure 4-5 in **Get Reader Parameters** (**0x01**) for a diagram of state transitions for the reader.

Parameters

Scanning status information stored in a data structure defined as follows:

```
typedef struct SCAN_STATUS_struc
{
  unsigned char     scan_state;     / / Scan state.
  unsigned char     status;         / / Current scan status.
} SCAN_STATUS_type;
```

The fields in this structure are defined as follows:

scan_state
The current scanning enable state of the SCAN41XX. Valid values are:

0 = Scanning disabled
1 = Scanning enabled

status
Current scan status. Valid values are:

1 = Acquiring (laser is on / contact is fully illuminated)
2 = Pulsing (if contact); Waiting (if laser)
3 = Stopped
4 = Timer expired (Klasse Eins only)

Example

For a code sample illustrating the **Get Scan Status** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Trigger Mode

Command Code: 0x0C

Type: Immediate

Action

Gets the current trigger mode of SCAN41XX.

Parameter

Trigger mode information stored in a data structure defined as follows:

```
typedef struct TRIG_MODE_struc
{
  unsigned char      trigger_mode;     //Current trigger mode
} TRIG_MODE_type;
```

The field in this structure is defined as follows:

trigger_mode
Triggering mode to process the physical trigger. Valid values are:

0 = Ignore external trigger (laser may be activated only by soft trigger)
1 = Reserved

2 = Enable laser when triggered

Example

For a code sample illustrating the **Get Trigger Mode** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set Trigger Mode

Command Code: 0x0D

Type: *Immediate*

Action

Sets the current trigger mode of SCAN41XX.

> **Note:** This command should not be issued if the command queue is not empty.

Parameter

Information to be stored in a data structure that sets the trigger mode for processing the physical trigger. See **Get Trigger Mode** (**0x0C**) for a definition of the data structure and a definition of its field (**trigger_mode**).

Example

For a code sample illustrating the **Set Trigger Mode** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Read Label with Wait

Command Code: 0x0E

Type: *Wait*

Note: There is a No Wait version of this command (Command Code: 0x8E).

Action

This command causes a decode attempt when a triggering command is received. When **Read Label** is specified as type *Wait*, SCAN41XX waits up to **timeout** milliseconds for a label to be scanned. (**timeout** is a field in the structure **SCB_type**, defined in *Scanner Control Block (SCB)*.

Parameters

Read parameters that are contained in a data structure defined as follows:

```
typedef struct READ_struc
{
  unsigned char     labeltype;       / / Returned label type
  void far          *data_buf_ptr;   / / Pointer to data buffer
  unsigned short    data_buf_len;    / / Length of data buffer
  unsigned short    label_length;    / / Length of data placed
                                     / / in buffer
  unsigned char     scan_direction;  / / Decode direction
  unsigned char     read_status;     / / Scanning status for this
                                     / / read
} READ_type;
```

The fields in this structure are defined as follows:

labeltype
The symbology (bar code) type of the returned label. Valid ASCII char values for **labeltype** and symbology type numbers (as hexadecimal numerals) are given in the following table:

Table 4-16. labeltype Values for READ_struc

Bar Code Type	ASCII char Value for labeltype	Hex Value
UPC E0	0	0x30
UPC E1	1	0x31
UPC A	2	0x32
MSI	3	0x33
EAN 8	4	0x34
EAN 13	5	0x35
Codabar	6	0x36
Code 39	7	0x37
D 2 of 5	8	0x38
I 2 of 5	9	0x39
Code 11	:	0x3A
Code 93	;	0x3B
Code 128	<	0x3C
Iata 2 of 5	>	0c3E
EAB 128	?	0x3F

data_buf_ptr
Pointer to a data buffer to store the decoded bar code.

Note: This must be a far pointer. Initialize this field before submitting the **Read** command.

data_buf_len
Length of data buffer pointed to be **data_buf_ptr.**

Note: Initialize this field before submitting the **Read**

command.

label_length
Length of decode data placed in the buffer pointed to by **data_buf_ptr**.

scan_direction
Decode direction of bar code. Valid values are:

1 = Forward
2 = Reverse

read_status
Current scan status for the current **Read** command. Valid values are:

1 = Acquiring
2 = Pulsing (if contact); Waiting (if laser)
3 = Stopped
4 = Timer expired (Klasse Eins only)

Example

For a code sample illustrating the **Read Label with Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set Soft Trigger

Command Code: 0x0F

Type: Immediate

Action

Sets the soft trigger flag in the SCAN41XX. The next **Read** command processed issues a trigger regardless of the status of the physical trigger buttons. The soft trigger flag is cleared when either a bar code is decoded or the laser beam times out. The **Set Soft Trigger** command may be issued after *No Wait* **Read** commands. However, if a *Wait* **Read** command is issued, the **Set Soft Trigger** has no effect because control has not been returned to the application program.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Set Soft Trigger** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Clear Soft Trigger

Command Code: 0x10

Type: *Immediate*

Action

Clears the soft trigger flag in the SCAN41XX.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Clear Soft Trigger** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Supported Decoders

Command Code: 0x11

Type: *Immediate*

Action

Returns null terminated string containing the decoder types that are in SCAN41XX whether they are currently enabled or not.

Parameter

Information returned in the **decoder_str** field of a data structure defined as follows:

```
typedef struct DECODER_STRING_struc
{
  char      decoder_str[25];      / / Null terminated decoder type
                                  / / string.

DECODER_STRING_type;
```

The field in this structure is defined as follows:

decoder_str
Null terminated string of decoder types. Valid ASCII characters for use in specifying decoders in **decoder_str** and the decoder type numbers as hexadecimal numerals are given in Table 4-17

Table 4-17. decoder_str Values

Bar Code Type	ASCII char Value for decoder_str	Hex Value
UPC E0	0	0x30
UPC E1	1	0x31
UPC A	2	0x32
MSI	3	0x33
EAN 8	4	0x34
EAN 13	5	0x35
Codabar	6	0x36
Code 39	7	0x37
D 2 of 5	8	0x38
I 2 of 5	9	0x39
Code 11	:	0x3A
Code 93	;	0x3B
Code 128	<	0x3C
Supp 2	Y	0x59
Supp 5	Z	0x5A

Example

For a code sample illustrating the **Get Supported Decoders** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Enabled Decoders

Command Code: 0x12

Type: *Immediate*

Action

Returns a null terminated string containing the decoder types that are enabled in SCAN41XX.

Parameters

The parameter (**decoder_str**) for this command is the same as that for the **Get Supported Decoders** (**0x11**) command. See **Get Supported Decoders** for the format of the decoder string.

Example

For a code sample illustrating the **Get Enabled Decoders** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Set Enabled Decoders

Command Code: 0x13

Type: Immediate

Action

A null terminated string containing the decoder types to be enabled is passed to SCAN41XX. If the decoder is not currently supported, the enable command is ignored for that decoder. If a decoder is *not* specified in **decoder_str**, it is disabled.

> **Note:** This command should not be issued if the command queue is not empty.

Parameters

The parameter (**decoder_str**) for this command is the same as that for the **Get Supported Decoders** (**0x11**) command.

Example

For a code sample illustrating the **Set Enabled Decoders** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Cancel Pending SCB

Command Code: 0x14

Type: Immediate

Action

Cancels a Scanner Control Block from the SCB command queue and sets the **status** and **retcode** of the canceled SCB to 10 (Command Canceled).

Parameter

SCB_ptr, which is a field in a data structure that is defined as follows:

```
typedef struct CAN_SCB_struc
{
    SCB_type far     *SCB_ptr;        / / Pointer to SCB to be cancelled

} CAN_SCB_type;
```

The field in this structure is defined as follows:

SCB_ptr
Far pointer to the SCB to be canceled from the SCB command queue.

Example

For a code sample illustrating the **Cancel Pending SCB** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Flush All Pending SCBs

Command Code: 0x15

Type: *Immediate*

Action

Cancels all queued entries from the SCB command queue. The **retcode** and **status** fields of all canceled SCBs are set to 10 (Command Canceled).

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Flush All Pending SCBs** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Get Number of Pending SCBs

Command Code: 0x16

Type: Immediate

Action

Returns the number of pending SCBs in the SCB command queue.

Parameter

The number of entries in the command queue is returned in the **pending** field of a data structure defined as follows:

```
typedef struct NUM_PEND_struc
{
  unsigned short      pending;      / / Number of entries in
                                    / / the command queue
} NUM_PEND_type;
```

Example

For a code sample illustrating the **Get Number of Pending SCBs** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Enable Scanning with No Wait

Command Code: 0x89

Type: No Wait

Note: There is a Wait version of this command (Command Code: 0x09).

Action

Enables scanning in SCAN41XX. All physical and soft triggers are processed.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Enable Scanning with No Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Disable Scanning with No Wait

Command Code: 0x8A

Type: No Wait

Note: There is a Wait version of this command (Command Code 0x0A).

Action

Disables scanning in SCAN41XX. All physical and soft triggers are ignored.

Parameters

There are no user parameters associated with this command.

Example

For a code sample illustrating the **Disable Scanning with No Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Read Label with No Wait

Command Code: 0x8E

Type: No Wait

Note: There is a Wait version of this command (Command Code 0x0E).

Action

This command causes a decode attempt when a triggering command is received.

Parameters

Read parameters contained in a data structure defined in the description of the **Read Label with Wait** command (**Command Code 0x0E**).

Example

For a code sample illustrating the **Read Label with No Wait** command, refer to the sample scanning program (SCANSAMP.C) in *Appendix 1* of this chapter.

Completion and Error Codes

Table 4-18 lists the error and completion codes returned by SCAN41XX when an SCB request presented by an application is complete.These values are returned in the **retcode** and **status** fields of the SCB.

Table 4-18. SCAN41XX Completion and Error Codes

Code	Explanation
0	**Success.** The associated command completed successfully.
1	**Unknown Command.** The command code passed in the **command** field of the SCB is not valid.
2	**Invalid SCB.** Either the address of the command parameter buffer passed in the SCB was not valid (zero), or the parameter buffer does not fit within a single segment (i.e., buffer offset + length > 0xFFFF).
3	**Invalid Command Parameter Pointer.** Either the address of the command parameter buffer passed in the SCB is invalid, or the parameter buffer does not fit within a single segment (i.e., buffer offset + command length > 0xFFFF).
5	**Invalid Parameter.** An invalid value has been passed to the scanner driver in the command parameter block of an SCB command.
9	**Command Time-out.** The command did not complete within the time specified in the **timeout** field of the SCB.
10	**Command Cancelled.** The command was cancelled by either the **Cancel SCB** (**0x14**) command or the **Flush** (**0x15**) command.
12	**Command Already Completed.** A command was completed before an attempt was made to cancel it, or the SCB was not found.
13	**Could Not Enable Scanning.** An attempt to enable the attached scanner failed.
14	**Scanning Not Enabled.** An attempt to issue a **Read** (**0x0E** or **0x8E**) command was made before scanning was enabled.
15	**Scanning Aborted.** Scanning was disabled while a scan was in progress.
16	**Physical Device Failure.** The designated scanning device is either not present or not responding to commands.

Table 4-18. SCAN41XX Completion and Error Codes (Continued)

Code	Explanation
17	**Data Too Large.** Decoded data were larger than the buffer size (**data_buf_length**) passed in a **Read** (**0x0E** or **0x8E**) command. Data is copied up to the buffer size, and this error is reported in the SCB.
18	**SCB Command Queue Full.** The SCB command queue is full. All subsequent SCB requests are ignored until there is space in the command queue**.**
19	**Scanning Already Enabled.** An attempt was made to enable scanning after a successful **Enable** (**0x09** or **0x89**) command had completed.
20	**Scanning Already Disabled.** An attempt was made to disable scanning after a successful **Disable** (**0x0A** or **0x8A**) command had completed.
21	**Invalid Data Buffer Pointer.** Either the address (***data_buf_ptr**) of the data buffer passed in the **Read** (**0x0E** or **0x8E**) command parameter block was not valid (zero), or the data buffer does not fit within a single segment (i.e., buffer offset + command length > 0xFFFF).
22	**Invalid SCB Pointer to Cancel.** Either the address of the SCB pointer to cancel was not valid (zero), or the data buffer does not fit within a single segment (i.e., buffer offset + command length > 0xFFFF).
23	**No Data From Scanner.** The attached scanning device was enabled and issued a Start Scan command, but no transition data was received from the scanner.
24	**SCB Currently In Use.** The SCB submitted is currently in use by another command. **Note:** This is a serious error and should be corrected during the application development phase.
25	**Command Queue Not Empty.** This scanner API command may not be issued while a **Read with No Wait** command is pending. The scanner driver returns this error code when any of the following commands is issued when the command queue is not empty: **Set Reader Parameters** (**0x02**), **Set Scan Parameters** (**0x04**), **Set Decoder Parameters** (**0x06**), **Set UPC/EAN General Parameters** (**0x08**), **Set Trigger Mode** (**0x0D**), **Set Enabled Decoders** (**0x13**).
255	**Command Pending.** The command is currently being processed by the SCAN41XX.

The Scanner Type Identifier Program (SCANTYPE)

Because both older (SE-1022 based) and newer (SE-1000 based) PPT 4100/4110/4140 terminals may exist at the same site, a scanner type identification program has been developed to identify the scanner type in a PPT 4100/4110/4140 terminal.

The identifier program (SCANTYPE) checks for the presence of the SE-1000 scanner module. If the SE-1000 is not present, the program reports that the SE-1022 module is installed. Because no other scanner types are supported for this revision of SCAN41XX, these are the only scanner modules that are identified.

Type Identifier Program Implementation and Usage

SCANTYPE attempts to enable the SE-1000 scanner module. If the program is successful, it reports a DOS ERRORLEVEL of 2. Otherwise, it reports an ERRORLEVEL of 1 to indicate that the SE-1022 is installed.

The Scanner Type Identification Program may be run as part of the AUTOEXEC.BAT file to determine which scanner driver to load. The following sample DOS batch file illustrates the use of ***SCANTYPE***.

```
Rem Load Symbol BIOS Extensions
XSYMBIOS

Rem Determine which scanner type is present
SCANTYPE
Rem Branch to correct scan driver loading or error detection
if ERRORLEVEL 3 goto ERRSCAN
if ERRORLEVEL 2 goto LOAD4100
if ERRORLEVEL 1 goto LOAD4122
if ERRORLEVEL 0 goto ERRSCAN

:LOAD4100
Rem Load the SE-1000 version of the PPT 4100 scanner driver
SCAN4100
goto CONTINUEAUTO

:LOAD4122
Rem Load the SE-1022 version of the PPT 4100 scanner driver
SCAN4122
goto CONTINUEAUTO

:ERRSCAN
ECHO Error in scanner type identification. Scan driver not loaded.

Rem Continue AUTOEXEC.BAT
:CONTINUEAUTO
```

Parameter Menu Scanning

Use the bar code menus in the appendix to enable and disable decoders and parameters in place of API commands. For parameter value definitions and defaults refer to Tables 4-3 through 4-8 in *Execution and Configuration*.

For best results, the application should parameterize the scanner driver as required. Using parameter bar codes can cause the following situations:

1. ***Do not use parameter bar codes with SCAN4122***. With SCAN4122, the parameter bar codes can change the way the scanner driver and the scanner communicate, which can prevent the scanner from operating properly.
2. Use parameter bar codes with SCAN4100 ***only when absolutely necessary and only with great caution***. With SCAN4100, the parameter bar codes can change the settings of parameters set via the API. If a Get is performed after the parameter bar code is scanned, it reflects the changes made by the parameter bar code. These changes will *not* survive a reboot unless saved by the application and restored via the API.

The following chart shows the beeper sequence, beeper pitch variation, and the associated indication for bar code scans in this section.

BEEPER SEQUENCE	PITCH VARIATION	INDICATION
3 Beeps	High-Low-High	Correct entry scanned.
2 Beeps	High-Low	Value entered (additional entries are expected).
6 Beeps	High-Low-High-Low-High-Low	Input error, or "CANCEL" is scanned.

Table 4-19 through Table 4-22 list the parameter defaults included in SCAN4100. The bar code menus associated with these parameters are displayed with captions on the pages that follow.

Table 4-19. SCAN4100 Parameter Defaults: Code Types

Code Type	Length	Default Value
UPC-A	0	Enabled
UPC-E0	0	Enabled
UPC-E1	0	Disabled
EAN-8	0	Enabled
EAN-13	0	Enabled
D 2 of 5	0-14	Enabled
I 2 of 5	14 and 10	Enabled
Code 39	0	Enabled
Codabar	0	Enabled
Code 128	0	Enabled
Code 93	0	Disabled
Code 11	4 to 55	Disabled
MSI Plessey	4 to 55	Disabled

Table 4-20. SCAN4100 Parameter Defaults: Decode Options

Decode Option	Default Value
Transmit UPC-E0 Check Digit	Disabled
Transmit UPC-E1 Check Digit	Disabled
Transmit UPC-A Check Digit	Enabled
Convert UPC-E0 to UPC-A	Disabled
Convert UPC-E1 to UPC-A	Disabled
EAN Zero Extend	Disabled
Code 39 Full ASCII	Disabled
CLSI Editing	Disabled
NOTIS Editing	Disabled
MSI Plessey Check Digit	One
Transmit MSI Plessey Check Digit	Disabled
Code 11 Check Digit	One

Table 4-20. SCAN4100 Parameter Defaults: Decode Options (Continued)

Decode Option	Default Value
Transmit Code 11 Check Digit	Disabled
Verify Code 39 Check Digit	Disabled
Decode Redundancy for UPC/EAN without Supplementals	5
UPC/EAN Security level	0

Table 4-21. SCAN4100 Parameter Defaults: Special Decode Options

Special Decode Option	Default Value
Simple Redundancy:	
Code 39	Disabled
D 2 of 5	Disabled
I 2 of 5	Disabled
Code 128	Disabled
Codabar	Enabled
Code 11	Disabled
MSI Plessey	Disabled
Code 93	Disabled
Bi-Directional Redundancy	Disabled
Linear UPC/EAN Decode	Enabled

Table 4-22. SCAN4100 Parameter Defaults: Miscellaneous

Parameter	Default Value
UPC-E0 Preamble	None
UPC-E1 Preamble	None
UPC-A Preamble	System Character
Transmit Code ID Character	Disabled

Set Default Parameter

Defaults are those listed in Table 4-19 through Table 4-22

SET ALL DEFAULTS

Code Type

Add or delete specific code types by scanning the appropriate bar code(s).

ENABLE ALL CODE TYPES

DELETE ALL CODE TYPES

ENABLE COMMON CODE TYPES ONLY

ADD CODE 39

DELETE CODE 39

ADD UPC-A

DELETE UPC-A

ADD UPC-E0

DELETE UPC-E0

ADD UPC-E1

DELETE UPC-E1

ADD EAN-8

DELETE EAN-8

ADD EAN-13

DELETE EAN-13

ADD I 2 OF 5

DELETE I 2 OF 5

ADD CODABAR

DELETE CODABAR

ADD CODE 128

DELETE CODE 128

ADD MSI/Plessey*

DELETE MSI/Plessey

ADD CODE 11**

DELETE CODE 11

ADD CODE 93

DELETE CODE 93

ADD D 2 OF 5

DELETE D 2 OF 5

*After adding MSI / Plessey you must select either one or two check digits from the ***Decode Options***.

After adding Code 11, you must select none, one or two check digits from the ***Decode Options*.

Code Lengths

To select two lengths for each code type:

1. Scan the desired option.
2. Scan two bar codes for each desired length. For example, for a length of “12”, scan “1” then “2”. For a length of “3”, scan “0”, then “3”. **You must always scan two bar codes for each length**.
3. If you make an error, or wish to change your selection, scan **CANCEL**

CODE 39 ANY LENGTH

CODE 39 LENGTH WITHIN RANGE

CODE 39 1 DISCRETE LENGTH

CODE 39 2 DISCRETE LENGTHS

CODABAR ANY LENGTH

CODABAR LENGTH WITHIN RANGE

CODABAR 1 DISCRETE LENGTH

CODABAR 2 DISCRETE LENGTHS

CODE 128 ANY LENGTH

CODE 128 LENGTH
WITHIN RANGE

CODE 128
1 DISCRETE LENGTH

CODE 128
2 DISCRETE LENGTHS

D 2 OF 5 ANY LENGTH

D 2 OF 5 LENGTH WITHIN RANGE

D 2 OF 5 1 DISCRETE LENGTH

D 2 OF 5 2 DISCRETE LENGTHS

CODE 93 ANY LENGTH

CODE 93 LENGTH WITHIN RANGE

CODE 93 1 DISCRETE LENGTH

CODE 93 2 DISCRETE LENGTHS

I 2 OF 5 ANY LENGTH*

I 2 OF 5 LENGTH WITHIN RANGE

I 2 OF 5 1 DISCRETE LENGTH

I 2 OF 5 2 DISCRETE LENGTHS

*Choosing I 2 of 5 ANY LENGTH may lead to misread codes.

MSI/Plessey
ANY LENGTH

MSI/Plessey
LENGTH WITHIN RANGE

MSI/Plessey 1 DISCRETE
LENGTH

MSI/Plessey 2 DISCRETE
LENGTHS

CODE 11
ANY LENGTH

CODE 11 LENGTH WITHIN
RANGE

CODE 11 1 DISCRETE
LENGTH

CODE 11 2 DISCRETE
LENGTHS

0

1

2

3

4

5

6

7

8

9

CANCEL

Decode Options

TRANSMIT UPC-E0 CHECK DIGIT

DO NOT TRANSMIT UPC-E0 CHECK DIGIT

TRANSMIT UPC-E1 CHECK DIGIT

DO NOT TRANSMIT UPC-E1 CHECK DIGIT

TRANSMIT UPC-A CHECK DIGIT

DO NOT TRANSMIT UPC-A CHECK DIGIT

CONVERT UPC-E0 TO UPC-A

DO NOT CONVERT UPC-E0 TO UPC-A

CONVERT UPC-E1 TO UPC-A

DO NOT CONVERT UPC-E1 TO UPC-A

ENABLE EAN ZERO EXTEND

DISABLE EAN ZERO EXTEND

ENABLE CODE 39 FULL ASCII

DISABLE CODE 39 FULL ASCII

DECODE CODE 11 WITH NO CHECK DIGITS

DECODE CODE 11 WITH 1 CHECK DIGIT

DECODE CODE 11 WITH 2 CHECK DIGITS

TRANSMIT CODE 11 CHECK DIGIT(S)

DO NOT TRANSMIT CODE 11 CHECK DIGIT(S)

DECODE MSI/Plessey WITH 1 CHECK DIGIT

DECODE MSI/Plessey WITH 2 CHECK DIGITS

TRANSMIT MSI/Plessey CHECK DIGITS

DO NOT TRANSMIT MSI/Plessey CHECK DIGITS

DECODEUPC/EAN SUPPLEMENTALS

IGNORE UPC/EAN SUPPLEMENTALS

AUTODISCRIMINATE UPC/EAN WITH SUPPLEMENTALS

ENABLE CLSI EDITING

DISABLE CLSI EDITING

ENABLE NOTIS EDITING

DISABLE NOTIS EDITING

VERIFTY CODE 39 CHECK DIGIT

DO NOT VERIFY CODE 39 CHECK DIGIT

TRANSMIT CODE ID CHARACTER

DO NOT TRANSMIT CODE ID CHARACTER

UPC/EAN SECURITY LEVEL 0

UPC/EAN SECURITY LEVEL 1

UPC/EAN SECURITY LEVEL 2

UPC/EAN SECURITY LEVEL 3

Decode Redundancy for UPC/EAN without Supplementals

To set a decode redundancy value:

1. Scan the DECODE REDUNDANCY bar code below.
2. Scan two bar codes on the next page that represent the desired number of times. For single digit numbers, include a leading zero.

3. If you make an error, or wish to change your selection, scan CANCEL.

DECODE REDUNDANCY for UPC/EAN without SUPPLEMENTALS

Decode Redundancy for UPC/EAN without Supplementals

0

1

2

3

4

5

6

7

8

9

CANCEL

UPC-A Preamble

Select one option for UPC-A preamble by scanning the appropriate bar code.

NONE

SYSTEM CHARACTER

SYSTEM CHARACTER &
COUNTRY CODE

UPC-E0 Preamble

Select one option for UPC-E0 preamble by scanning the appropriate bar code.

NONE

SYSTEM CHARACTER

SYSTEM CHARACTER &
COUNTRY CODE

UPC-E1 Preamble

Select one option for UPC-E1 preamble by scanning the appropriate bar code.

NONE

SYSTEM CHARACTER

SYSTEM CHARACTER &
COUNTRY CODE

Special Decode Options

Simple Redundancy

ENABLE CODE 39
SIMPLE REDUNDANCY

DISABLE CODE 39
SIMPLE REDUNDANCY

ENABLE CODE 128
SIMPLE REDUNDANCY

DISABLE CODE 128
SIMPLE REDUNDANCY

ENABLE I 2 OF 5
SIMPLE REDUNDANCY

DISABLE I 2 OF 5
SIMPLE REDUNDANCY

ENABLE CODABAR
SIMPLE REDUNDANCY

DISABLE CODABAR
SIMPLE REDUNDANCY

ENABLE MSI/ Plessey
SIMPLE REDUNDANCY

DISABLE MSI/Plessey
SIMPLE REDUNDANCY

ENABLE CODE 11
SIMPLE REDUNDANCY

DISABLE CODE 11
SIMPLE REDUNDANCY

ENABLE D 2 of 5
SIMPLE REDUNDANCY

DISABLE D 2 of 5
SIMPLE REDUNDANCY

ENABLE CODE 93
SIMPLE REDUNDANCY

DISABLE CODE 93
SIMPLE REDUNDANCY

Bi-Directional Redundancy

Enable or disable bi-directional redundancy for codes with ***Simple Redundancy*** enabled.

ENABLE BIDIRECTION-AL REDUNDANCY

DISABLE BIDIRECTIONAL REDUNDANCY

Linear UPC/EAN Decode

ENABLE LINEAR UPC DECODE

DISABLE LINEAR UPC DECODE

Differences Between SCAN3000 and SCAN41XX

This section describes differences between the Series 3000 Scanner Driver (SCAN3000) and the PPT 4100/4110/4140 Scanner Driver (SCAN41XX). It is meant to aid the transition to SCAN41XX for application programmers who already know how to access scanning functions with SCAN3000. It consists of the following subsections:

- Overview of SCAN3000 Architecture
- Overview of SCAN41XX Architecture
- Series 3000 Scanning Driver Generation
- PPT 4100/4110/4140 Scanning Driver Generation
- Series 3000 Scanning Driver Configuration
- PPT 4100/4110/4140 Scanning Driver Configuration
- API Translations from SCAN41XX to SCAN3000

Overview of SCAN3000 Architecture

Figure 4-7 shows the basic architecture of SCAN3000 and how application programs interact with it.

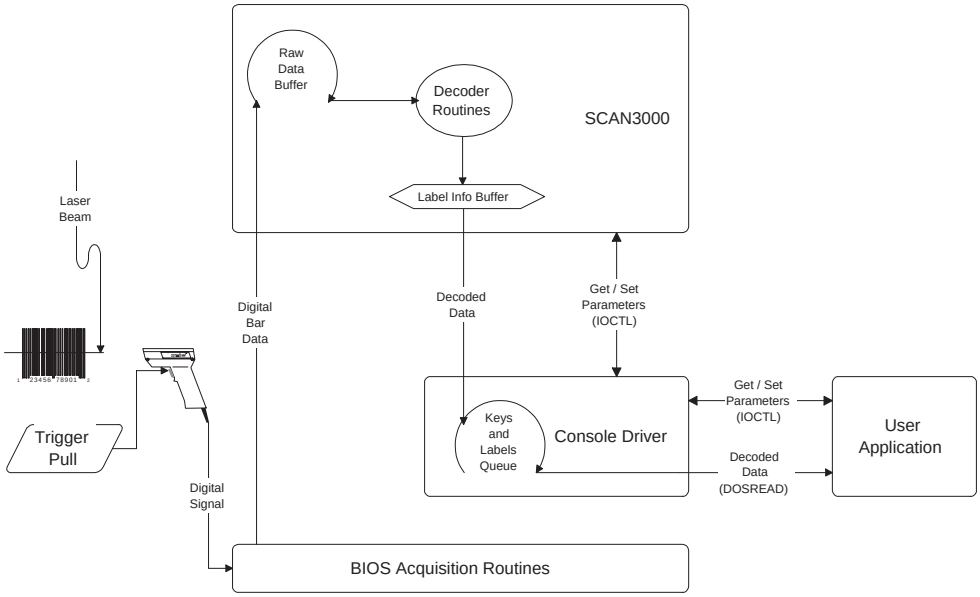


Figure 4-7. SCAN3000 Driver/Application Interaction

SCAN3000 is loaded as a Terminate and Stay Resident (TSR) program from the AUTOEXEC.BAT file or from an application program. Once loaded, the SCAN3000 TSR operates as follows:

- It contains the code necessary to interface with the BIOS to control laser guns and contact wands to acquire bar code data.
- It decodes the bar code data.
- It places the decoded data into the console queue.

Application programs must retrieve the decoded data from the console queue and copy it to a local area.

SCAN3000 may be invoked with two optional parameters. The first parameter controls the size of the raw data buffer used to hold the acquired bar and space data; its default is 2048 bytes. The second parameter controls the size of the label information buffer passed on to the console driver.

To establish a logical connection to SCAN3000, application programs must first open the driver via the ***DosOpen*** command. The handle that is returned must be used by all commands that wish to communicate with SCAN3000. The ***IOCTL*** (i.e., Input/Output Control) function calls ***DosIoCtrlRdData***. ***DosIoCtrlWrData*** may then be used to retrieve and modify scanning parameters. The ***DosRead*** command gets decoded data from the console queue, allowing multiple decodes to be queued up in the console queue. However, if the application program does not retrieve the decoded data from the console queue, the queue eventually fills up and further scanned data is lost. ***IOCTL*** commands issued to SCAN3000 are first processed by the console driver, which passes on to SCAN3000 related commands to the driver.

Overview of SCAN41XX Architecture

The PPT 4100/4110/4140 Scanner Driver (SCAN41XX) is loaded as a TSR either from the AUTOEXEC.BAT file or by an application program. The SCAN41XX TSR contains the control code to interface with attached scanner devices and to manipulate scanner driver parameters. The Extended Symbol BIOS TSR (XSYMBIOS) must be loaded prior to loading SCAN41XX. XSYMBIOS contains functions which enable SCAN41XX to interface with the PPT 4100/4110/4140 hardware platform. Refer to *Execution and Configuration* for the procedures for loading SCAN41XX.

SCAN41XX may be invoked with an optional command line parameter which denotes the name of an alternate configuration file. If used, this parameter must specify the entire path name along with the drive. Refer to *Execution and Configuration* for further information on the configuration file.

Applications communicate with SCAN41XX by loading a pointer to a Scanner Control Block (SCB) in the register pair ES:BX and issuing a dedicated software interrupt (INT 0x62). SCAN41XX processes SCBs in two ways – immediately and queued. The Wait option submits immediate commands, the Wait/No Wait option submits queued commands. For a command submitted with the Wait option, SCAN41XX returns control to the application only after the specified command is complete. It places No Wait commands into a First-In-First-Out (FIFO) queue and returns control to the application. Queued commands are processed in the background. The application program must monitor return and status codes to check for command complete result. After a queued command has completed execution, it is removed from the command queue.

To enable the scanner and scan a bar code, an application must at least submit the following SCBs:

1. ***Enable Scanning*** Command Number 0x09 (Wait option) or 0x89 (No Wait option)
2. ***Read Label*** Command Number 0x0E (Wait option) or 0x8E (No Wait option)
3. ***Disable Scanning*** Command Number 0x0A (Wait option) or 0x8A (No Wait option)

Resubmit the ***Read Label*** command for each bar code to be scanned. For the details of SCB and command parameter structures, see the *User Interface* and *Supported API Commands* sections.

Figure 4-8 depicts the basic architecture of SCAN41XX. This release of SCAN41XX supports the internal scanner module SE-1000. The driver performs the actual decode process, allowing software updates to be sent to decoders as they become available. To use the software decoders, upgrade existing hardware decoder modules to shared memory acquisition cards. The acquisition card consists of a hardware interface that assists SCAN4100 by performing the low level acquisition of bar and space data and places them into a memory area shared by the hardware and SCAN4100. This bar / space data is decoded by SCAN4100 and passed on to the application program.

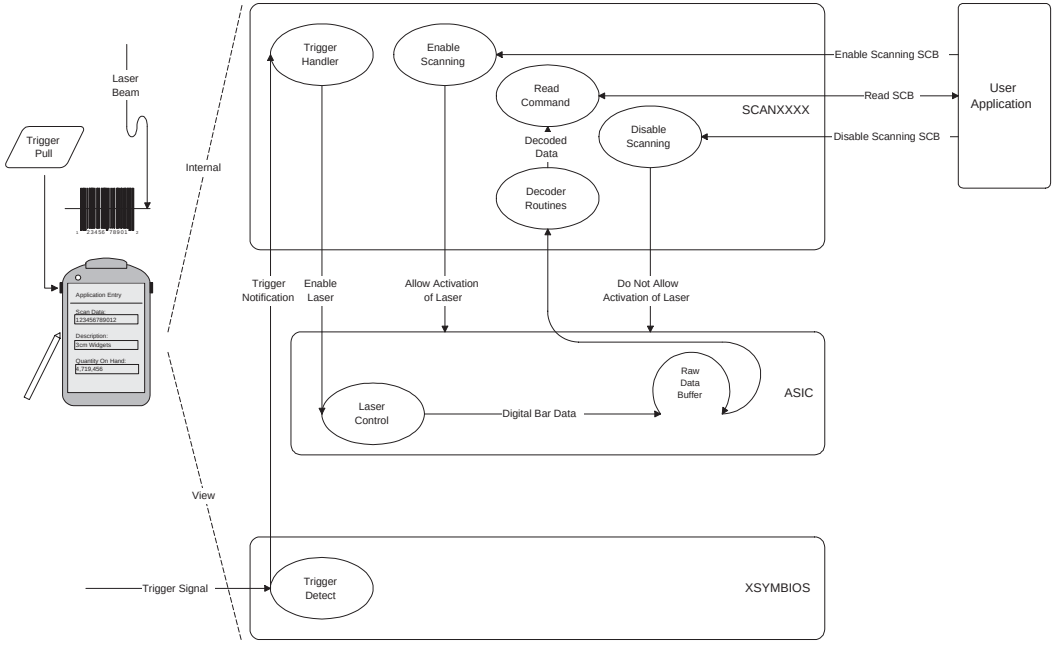


Figure 4-8. SCAN41XX Driver/Application Interaction

Series 3000 Scanning Driver Generation

SCAN3000 is shipped as a set of object files, which usually reside in the C:\3000\FILES directory, and a program generator, which usually resides in the C:\3000 directory. The Microsoft C (Version 6.0) compiler must also be installed to assist in the TSR generation process. The program generator, BLDSCAN.EXE, is a text-based, menu-driven program that customizes a scanning TSR for a user's specific needs. Parameters such as decoder types, desired lengths, check digits, and data formats are selected while using BLDSCAN.

When the ***Execute*** program option is selected, two files are generated:

- a linker response file which contains the names of the object modules of the selected decoders and control routines

  Only the decoders enabled are actually placed in the resulting SCAN3000 TSR. This allows for the smallest possible program size. Minimum program size is an important factor because the TSR program must reside in the TPA (Transient Program Area) memory space of the terminal.

- a C subroutine is generated which contains the parameter values selected during BLDSCAN

  This subroutine is compiled and linked along with the other object modules in the linker response file. If PDF was enabled, another C subroutine is generated with PDF-specific parameters and becomes part of SCAN3000.

The initial invocation of SCAN3000 calls the C parameterization subroutines and causes registration with the console driver to allow scanning. SCAN3000 may only be loaded once, further attempts to load SCAN3000 are ignored.

PPT 4100/4110/4140 Scanning Driver Generation

The PPT 4100/4110/4140 Scanning Driver is shipped as an executable TSR program SCAN41XX.EXE along with the default configuration file SCAN41XX.CFG. The configuration file is a text file with the following format:

```
[Parameter_Section_Heading1]

parameter name1 = parameter value1

parameter name2 = parameter value2

    .

    .

    .
```

To customize the driver, edit the configuration file with any PC text editor. See *Execution and Configuration* for a description of the content and structure of a scanner configuration file.

Place the SCAN41XX and a configuration file in the terminal either on the flash disk or on a PCMCIA card. If you are using the default configuration file, it must reside on the same drive and directory as SCAN41XX.EXE. If you are using an alternate configuration file, specify its name as a command line parameter to SCAN41XX, including its entire path and drive letter. If SCAN41XX cannot locate a default configuration file, it uses default values to parameterize the driver. Refer to Tables 4-3 through 4-8 in *Execution and Configuration* for lists of scanning parameters and the default values used by SCAN41XX.

The initial invocation of SCAN41XX causes the configuration file to be read in and verified. If any errors are found in the configuration file, the program load aborts and an error message displays detailing the error. Refer to Table 4-2 in *Execution and Configuration* for a list of load time messages, including those displayed to report configuration file errors.

After SCAN41XX is parameterized, the program terminates, removing the initialization and parameterization portion of code, and stays resident. Load SCAN41XX once; further attempts to load SCAN41XX are ignored.

Series 3000 Scanning Driver Configuration

After SCAN3000 is loaded, application programs may manipulate parameters at runtime. However, decoders that were not enabled by BLDSCAN cannot be enabled by application programs. These unselected decoders are not linked to the driver and therefore do not exist.

An application program uses the ***DosOpen*** command to open the scanner driver. The handle returned must be used by all commands that wish to communicate with SCAN3000. The IOCTL (Input / Output Control) function calls ***DosIoCtrlRdData*** and ***DosIoCtrlWrData*** may then be used to access and modify scanning parameters. The parameters that are passed to these functions are the handle value that was established by the **DosOpen** command, a pointer to an IOCTL block, and the length of the data to be accessed.

Bar code input devices are configured through the Get / Set Reader Characteristics, Get / Set Reader Parameters, Get / Set Scan Parameters subcommands. Part of each decoder's parameters must be get / set individually through the Get / Set Decoder Parameters subcommands. Use the full decoder symbolic name (e.g. CODE_39, CODABAR, UPC_A, etc.) to reference a decoder. This command can configure the following parameters: enable status, minimum and maximum length and one decoder specific parameter. Other parameters are accessed by the Get / Set Redundancy, Get / Set Check Digit Information, Get / Set UPC Parameters, Get / Set Data Formatting subcommands. For specific command formats refer to the Series 3000 Application Development Kit documentation.

PPT 4100/4110/4140 Scanning Driver Configuration

After SCAN41XX is loaded, application programs may manipulate scanning parameters at runtime. All decoder types supported by the SE-1000 decode module are accessible and controllable.

The following features of the PPT 4100 / 4110 / 4140 Scanning Driver highlight significant differences from the Series 3000 Scanning Driver:

- Use the dedicated software interrupt 0x62 and scanner control blocks (SCB) to access scanning parameters.
- Overall parameters are rearranged or modified to place them in more logical groupings and orderings than they are in the Series 3000 driver.
- The ***Get/Set Reader Parameters*** command controls bar code input devices.
- Parameters are grouped together by categories:

  - common reader parameters
  - laser specific parameters
  - contact wand specific parameters
- Up to ten scanner configurations may be saved at one time and switched to by setting only the parameter index field of the SCB.
- The ***Get/Set Scan Parameters*** commands control general decoder parameters.
- The ***Get/Set Decoder Parameters*** commands control parameters specific to a decoder type. Decoders are referenced by their standard decoder values (e.g. 0x37 = Code 39, 0x36 = Codabar, 0x32 = UPC A, etc.).
- The ***Get/Set UPC/EAN General Parameters*** commands group and control general parameters specific to the UPC/EAN code types.

For the format of scanner control blocks and command structures see *User Interface* and *Supported API Commands*.

API Translations from SCAN41XX to SCAN3000

SCAN41XX combines functions of the Series 3000 BIOS, console driver, and SCAN3000. Series 3000 functions that control trigger handling, scanner acquisition, and reporting decode data to applications are part of SCAN41XX. Although much of the API to SCAN3000 has been retained, some parameters are rearranged or modified for clarity. Table 4-23 shows the API translation from SCAN41XX to SCAN3000. Note in this table that Get/Set commands are treated together and that the hexadecimal values of command codes are given in parentheses after the associated commands. See *Supported API Commands* for descriptions of the SCAN41XX commands listed in Table 4-23

Table 4-23. API Translations from SCAN41XX to SCAN3000

SCAN41XX Command	Series 3000 API Translation
GET VERSION INFORMATION (0x00) *version_str version	Not implemented in SCAN3000.
GET/SET READER PARAMETERS (0x01 & 0x02)	
parameter_index	
	GET/SET SCAN MODE (0x01 & 0x01)
scan_mode	scan_mode
	GET/SET SCANNER PARAMETERS (0x05 & 0x05)
enable_settle_time	enable_used & enable_settle_time
power_settle_time	power_settle_time
inverse_label_flag	inverse_label_flag
white_data_logic_lvl	white-data_logic_lvl
	GET/SET SCAN PARAMETERS (0x06 & 0x06)
dec_beep_time	dec_beep & dec_beep_time
dec_beep_freq	beep_freq
scans_per_label	scans_per_label
clk_speed_toggle	clk_speed_toggle
trans_resolution	trans_resolution

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
subseq_scan_time	subseq_scan_time
no_data_time	no_data_time
post_dec_action	post_dec_action
dec_fail_action	dec_fail_action
	GET/SET SCANNER CHARACTERISTICS (0x04 & 0x04)
prod_trigger	prod_trigger
two_stage_trigger	This is a BIOS function in Series 3000
multiple_scan	multiple_scan
prod_direction	prod_direction
	GET/SET SCAN PARAMETERS (0x06 & 0x06)
dec_feedb_time	decode_feedback & dec_feedb_time
	GET/SET SCANNER CHARACTERISTICS (0x04 & 0x04)
dec_feedback_lvl	dec_feedback_lvl
scan_led_ctrl	Not implemented in SCAN3000
scan_led_lvl	Not implemented in SCAN3000
KE_enable	Not implemented in SCAN3000
	GET/SET SCAN PARAMETERS (0x06 & 0x06)
KE_time_used	KE_time_used
KE_time_left	KE_time_left

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
qz_ratio	qz_ratio
init_scan_time	init_scan_time
pulse_delay	pulse_delay
GET/SET SCAN PARAMETERS (0x03 & 0x04)	**GET/SET REDUNDANCY (0x0C & 0x09)**
bidir_redundancy	bidir_redundancy
	GET/SET RETURN FORMAT (0x0F & 0x0C)
xmit_code_id	xmit_code_id
GET/SET DECODER PARAMETERS (0x05 & ox06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
label_type = 0x30	**label_type = UPC_E0**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET UPC PARAMETERS (0x0E & 0x0B)
decoder_specific[0]	upc_e_chk_b
decoder_specific[1]	upce_preamble
	GET/SET DECODER PARAMETERS (0x07 & 0x07)
decoder_specific[2]	decoder_specific[0](cvt_upce0_ to_upca)

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x31	**labeltype = UPC_E1**
minlength maxlength	minlength maxlength
alloc_specific	
	GET/SET UPC PARAMETERS (0x0E & 0x0B)
decoder-specific[0]	upc_e1_chk_b
decoder_specific[1]	upce1_preamble
	GET/SET DECODER PARAMETERS (0x07 & 0x07)
decoder_specific[2]	decoder_specific[0](cvt_upce1_to_upca)
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x32	**labeltype = UPC_A**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET UPC PARAMETERS (0x0B & 0x0E)
decoder_specific[0]	upc_a_chk_b
decoder_specific[1]	upca_preamble

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET/SET DECODER PARAMETERS (0x05 & 0x06) **labeltype = 0x33**	**GET/SET DECODER PARAMETERS (0x07 & 0x07)** **labeltype = MSI**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	cmsi_red_enabled
	GET/SET DECODER PARAMETERS (0x07 & 0x07)
decoder_specific[1]	decoder_specific[0](msi_num_cd)
	GET/SET CHECKS (0x0D & 0x0A)
decoder_specific[2]	report_msi_chk
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x34	**labeltype = EAN_8**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET UPC PARAMETERS (0x0E & 0x0B)
decoder_specific[0]	conv_ean8to13_b

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x35	**labeltype = EAN_13**
minlength	minlength
maxlength	maxlength
alloc_specific	
There are no associated decoder specific parameters.	
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x36	**labeltype = CODABAR**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	cbar_red_enabled
	GET/SET RETURN FORMAT (0x0F & 0x0C)
decoder_specific[1]	clsi_editing
decoder_specific[2]	notis_editing

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x37	**labeltype = CODE_39**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET CHECKS (0x0D & 0x0A)
decoder_specific[0]	code39_chk_b
decoder_specific[1] (code39_concatenation)	Not controllable in SCAN3000
	GET/SET DECODER PARAMETERS (0x07 & 0x07)
decoder_specific[2]	decoder_specific[0](code39_full_ascii)
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[3]	c39_red_enabled
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x38	**labeltype = CODE_D25**
minlength	minlength
maxlength	maxlength
alloc_specific	

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	cd25_red_enabled
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x39	**labeltype = CODE_I25**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	ci25_red_enabled
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x3A	**labeltype = CODE_11**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
decoder_specific[0]	ci25_red_enabled
	GET/SET DECODER PARAMETERS (0x07 & 0x07)
decoder_specific[1]	decoder_specific[0](c11_chk_dgt)
	GET/SET CHECKS (0x0D & 0x0A)
decoder_specific[2]	c11_chk_dgt
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x3B	**labeltype = CODE_93**
minlength	minlength
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	c93_red_enabled
GET/SET DECODER PARAMETERS (0x05 & 0x06)	**GET/SET DECODER PARAMETERS (0x07 & 0x07)**
labeltype = 0x3C	**labeltype = CODE_128**
minlength	minlength

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
maxlength	maxlength
alloc_specific	
	GET/SET REDUNDANCY (0x0C & 0x09)
decoder_specific[0]	c128_red_enabled
GET/SET UPC/EAN GENERAL PARAMETERS (0x07 & 0x08)	**GET/SET UPC PARAMETERS (0x0E & 0x0B)**
security_level8	security_level
supp_2	supp_2
supp_5	supp_5
supp_auto_d	supp_auto_d
supp_retry	supp_retry
linear_decode	linear_decode
Enable Scanning (0x09 & 0x89)	**SET SCAN STATE (0x02)**
DISABLE SCANNING (0x0A & 0x8A)	**SET SCAN STATE (0x02)**
GET SCAN STATUS (0x0B)	**GET SCAN STATE (0x02)**
scan_state	scan_state
status	status

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET/SET TRIGGER MODE (0x0C & 0x0D)	No matching command in SCAN3000
trigger_mode	
READ LABEL (0x0E)	**GET LAST CHAR READ STATUS (0x03)**
labeltype	labeltype
	DOSREAD(IN CONSOLE DRIVER)
*data_buf_ptr	*data_buf_ptr
data_buf-len	N / A
	GET LAST READ CHAR STATUS (0x03)
label_length	label_length
scan_direction	scan_direction
	GET SCAN STATE (0x02)
read_status	status
SET SOFT TRIGGER (0x0F)	**SET SOFT TRIGGER (0x03)**
CLEAR SOFT TRIGGER (0x10)	**RESET SOFT TRIGGER (0x08)**

Table 4-23. API Translations from SCAN41XX to SCAN3000 (Continued)

SCAN41XX Command	Series 3000 API Translation
GET SUPPORTED DECODERS (0x11) decoder_str	GET NUMBER OF DECODERS (0x08) in conjunction with GET NEXT DECODER NAME (0x09)
GET/SET ENABLED DECODERS (0x12 & 0x13)	Not implemented in SCAN3000
decoder_str	
CANCEL SCB (0x14)	Not implemented in SCAN3000
FLUSH COMMAND QUEUE (0x15)	Not implemented in SCAN3000
GET NUMBER OF PENDING SCBs (0x16)	Not implemented in SCAN3000

Appendix 1. SCANSAMP.C

This appendix contains the sample application program referred to in **Example** sections of *SCB Commands (Descriptions)*. It is also contained in the SDK file

C:\SDK4100\SAMPLES\SCANNER\SCANSAMP.C

where **C:\SDK4100** is the default installation directory.

The files scandef.h and scanprot.h can be found in *Appendix 2* and *Appendix 3*, respectively, of this chapter.

```
#include <ctype.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "scandef.h"          // Scan Driver Definition File (See Appendix 2)
#include "scanprot.h"         // Scan Driver Prototypes File (See Appendix 3)

SCB_type        SCB1, SCB2, SCB3;  // SCB variables

union REGS      inregs, outregs;     // Used by int86x calls
struct SREGS    segregs;             // Used by int86x calls

VERSION_INFO_type   driver_version;   // Area for driver version
                                      // information

READ_type           READDATA;  // Area for read label information
#define DATA_LEN    30         // Length of data buffer

char  data[DATA_LEN];      // Data buffer for decoded data

char  ok_2_scan;           // Scan flag
```

```
// Submits an SCB to the scanner driver *********************************

void submit_SCB (SCB_type far *SCB)
{
  //
  // Set up ES:BX to point to the SCB
  //
  inregs.x.bx = FP_OFF (SCB);
  segregs.es = FP_SEG (SCB);

  //
  // Issue the interrupt to the scanner driver
  //
  int86x (SCAN_INT, &inregs, &outregs, &segregs);

  return;
}

// Checks to ensure that the scanner driver is loaded ***************

char scan_driver_check (SCB_type far *SCB)
{
  // Declare an area to place the scanner driver vector
  void (_interrupt _far *scanvect)(void);

  //
  // Get the scanner driver vector value
  //
  scanvect = _dos_getvect (SCAN_INT);

  //
  // If scan driver vector is zero then the driver is not loaded
  //
  if (scanvect == NULL)
    return (FALSE);

  //
  // Set up SCB - set invalid cmd number
  //
  SCB->command = CMD_INVALID;
```

```
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  //
  // If the unknown command return code is in the SCB, then
  // the driver is loaded; return TRUE. Otherwise, return FALSE.
  //
  if (SCB->status == SRTN_UNKNOWN_CMD)
    return (TRUE);
  else
    return (FALSE);
}

// Get Version Information function ****************************

void get_version_info (SCB_type far *SCB,
                       VERSION_INFO_type far *ver_data)
{
  //
  // Set up SCB - get version information, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_VERSION_INFO;
  SCB->cmdparam = ver_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Get Reader Parameters function **********************************

void get_reader_parms (SCB_type far *SCB, unsigned char index,
                       READER_MODE_type far *reader_data)
{
  //
  // Set parameter_index field to desired index.
  //
  reader_data->parameter_index = index;

  //
  // Set up SCB - get reader parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_READER_PARMS;
  SCB->cmdparam = reader_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Set Reader Parameters function ********************************

void set_reader_parms (SCB_type far *SCB, unsigned char index,
                       READER_MODE_type far *reader_data)
{
  //
  // Set parameter_index field to desired index.
  //
  reader_data->parameter_index = index;

  //
  // Set up SCB - set reader parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_SET_READER_PARMS;
```

```
  SCB->cmdparam = reader_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Scan Parameters function ******************************************

void get_scan_parms (SCB_type far *SCB,
                     SCAN_PARMS_type far *scan_data)
{
  //
  // Set up SCB - get scan parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_SCAN_PARMS;
  SCB->cmdparam = scan_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
1}
```

```
// Set Scan Parameters function ************************************

void set_scan_parms (SCB_type far *SCB,
                     SCAN_PARMS_type far *scan_data)
{
  //
  // Set up SCB - set scan parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_SET_SCAN_PARMS;
  SCB->cmdparam = scan_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Decoder Parameters function ****************************

void get_decoder_parms (SCB_type far *SCB,
                        DECODER_PARMS_type far *decoder_data)
{
  //
  // Set up SCB - get decoder parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_DECODER_PARMS;
  SCB->cmdparam = decoder_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Set Decoder Parameters function ********************************

void set_decoder_parms (SCB_type far *SCB,
                        DECODER_PARMS_type far *decoder_data)
{
  //
  // Set up SCB - set decoder parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_SET_DECODER_PARMS;
  SCB->cmdparam = decoder_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get UPC/EAN General Parameters function ************************

void get_upcean_parms (SCB_type far *SCB,
                       UPC_GEN_PARMS_type far *upcean_data)
{
  //
  // Set up SCB - get UPC/EAN parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_UPCEAN_PARMS;
  SCB->cmdparam = upcean_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Set UPC/EAN General Parameters function *****************

void set_upcean_parms (SCB_type far *SCB,
                       UPC_GEN_PARMS_type far *upcean_data)
{
  //
  // Set up SCB - set UPC/EAN parameters, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_SET_UPCEAN_PARMS;
  SCB->cmdparam = upcean_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Enable Scanning with Wait function ***************************

void enable_scanning_wait (SCB_type far *SCB)
{
  //
  // Set up SCB - enable scanning with wait, 3 second timeout
  // and no post process routine.
  //
  SCB->command = CMD_ENABLE_SCANNING_WAIT;
  SCB->timeout = THREE_SECONDS;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Disable Scanning with Wait function ***************************

void disable_scanning_wait (SCB_type far *SCB)
{
  //
  // Set up SCB - disable scanning with wait, 3 second timeout
  // and no post process routine.
  //
  SCB->command = CMD_DISABLE_SCANNING_WAIT;
  SCB->timeout = THREE_SECONDS;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Scan Status function ***************************************

void get_scan_status (SCB_type far *SCB,
                      SCAN_STATUS_type far *scanstat_data)
{
  //
  // Set up SCB - get scan status, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_SCAN_STATUS;
  SCB->cmdparam = scanstat_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Trigger Mode function ***************************************
```

```
void get_trig_mode (SCB_type far *SCB, TRIG_MODE_type far *trig_data)
{
  //
  // Set up SCB - get trigger mode, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_TRIG_MODE;
  SCB->cmdparam = trig_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Set Trigger Mode function ****************************************

void set_trig_mode (SCB_type far *SCB, TRIG_MODE_type far *trig_data)
{
  //
  // Set up SCB - get trigger mode, set command parameter field
  //  and no post process routine.
  //
  SCB->command = CMD_SET_TRIG_MODE;
  SCB->cmdparam = trig_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Read Label with Wait function ************************************
```

```
void read_label_wait (SCB_type far *SCB, READ_type far *read_data,
                      void far *buffer, unsigned short buffer_len)
{
  //
  // Set *data_buf_ptr and data_buf_len fields in *read_data
  //
  read_data->data_buf_ptr = buffer;
  read_data->data_buf_len = buffer_len;

  //
  // Set up SCB - set read with wait, set command parameter field, set
  // 3 second timeout and no post process routine.
  //
  SCB->command = CMD_READ_LABEL_WAIT;
  SCB->cmdparam = read_data;
  SCB->timeout = THREE_SECONDS;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Set Soft Trigger function ****************************************

void set_softtrig (SCB_type far *SCB)
{
  //
  // Set up SCB - set software trigger and no post process routine.
  //
  SCB->command = CMD_SET_SOFTTRIG;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Clear Soft Trigger function ****************************************

void clr_softtrig (SCB_type far *SCB)
{
  //
  // Set up SCB - clear software trigger and no post process routine.
  //
  SCB->command = CMD_CLR_SOFTTRIG;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Get Supported Decoders function ************************************

void get_supported_decoders (SCB_type far *SCB,
                             DECODER_STRING_type far *support_data)
{
  //
  // Set up SCB - get supported decoders, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_SUPP_DEC;
  SCB->cmdparam = support_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Enabled Decoders function *********************************

void get_enabled_decoders (SCB_type far *SCB,
                           DECODER_STRING_type far *enable_data)
{
  //
  // Set up SCB - get enabled decoders, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_ENAB_DEC;
  SCB->cmdparam = enable_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Set Enabled Decoders function ************************************

void set_enabled_decoders (SCB_type far *SCB,
                           DECODER_STRING_type far *enable_data)
{
  //
  // Set up SCB - set enabled decoders, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_SET_ENAB_DEC;
  SCB->cmdparam = enable_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Cancel Pending SCB function ************************************

void cancel_scb (SCB_type far *SCB, CAN_SCB_type far *cancel_data,
                 SCB_type far *canSCB)
{
  //
  // Set SCB_ptr to SCB to be canceled
  //
  cancel_data->SCB_ptr = canSCB;

  //
  // Set up SCB - cancel SCB, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_CANCEL_SCB;
  SCB->cmdparam = cancel_data;
  SCB->process = NULL;

  //
  // Submit the SCB
```

```
  //
  submit_SCB (SCB);

  return;
}

// Flush All Pending SCBs function **********************************

void flush_queue (SCB_type far *SCB)
{
  //
  // Set up SCB - flush queue and no post process routine.
  //
  SCB->command = CMD_FLUSH;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Get Number of Pending SCBs function ******************************

void get_pending (SCB_type far *SCB, NUM_PEND_type far *pend_data)
{
  //
  // Set up SCB - get number of pending SCBs, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_GET_PEND_SCB;
  SCB->cmdparam = pend_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);
```

```
  return;
}

// Enable Scanning with No Wait function ************************

void enable_scanning_nowait (SCB_type far *SCB)
{
  //
  // Set up SCB - enable scanning with no wait and no post process routine
  //
  SCB->command = CMD_ENABLE_SCANNING_NOWAIT;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

// Disable Scanning with No Wait function ************************

void disable_scanning_nowait (SCB_type far *SCB)
{
  //
  // Set up SCB - disable scanning with no wait and no post process routine.
  //
  SCB->command = CMD_DISABLE_SCANNING_NOWAIT;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}
```

```
// Read Label with No Wait function ***********************************

void read_label_nowait (SCB_type far *SCB, READ_type far *read_data,
                        void far *buffer, unsigned short buffer_len)
{
  //
  // Set *data_buf_ptr and data_buf_len fields in *read_data
  //
  read_data->data_buf_ptr = buffer;
  read_data->data_buf_len = buffer_len;

  //
  // Set up SCB - set read with no wait, set command parameter field
  // and no post process routine.
  //
  SCB->command = CMD_READ_LABEL_NOWAIT;
  SCB->cmdparam = read_data;
  SCB->process = NULL;

  //
  // Submit the SCB
  //
  submit_SCB (SCB);

  return;
}

//
// The following function is called in the main program
// to display the barcode label type
//
```

```
void show_labeltype (unsigned char ltype)
{
  printf ("Label type :   ");
  switch(ltype)
  {
    case TYPE_UPCE0:
      printf("UPCE0");
      break;

    case TYPE_UPCE1:
      printf("UPCE1");
      break;

    case TYPE_UPCA:
      printf("UPCA");
      break;

    case TYPE_MSI:
      printf("MSI");
      break;

    case TYPE_EAN8:
      printf("EAN8");
      break;

    case TYPE_EAN13:
      printf("EAN13");
      break;

    case TYPE_CODABAR:
      printf("CODABAR");
      break;

    case TYPE_CODE39:
      printf("CODE 3 OF 9");
      break;

    case TYPE_D2OF5:
      printf("D 2 OF 5");
      break;
```

```
      case TYPE_I2OF5:
        printf("I 2 OF 5");
        break;

      case TYPE_CODE11:
        printf("CODE 11");
        break;

      case TYPE_CODE93:
        printf("CODE 93");
        break;

      case TYPE_CODE128:
        printf("CODE 128");
        break;

      default:
        printf("Unknown %.2X",ltype);
        break;
    };
    printf ("\n");

    return;
}

//
// The following function is called in the main program.
// It submits a read and waits for either decode data or a
// keypress before exiting. It the 'T' key is pressed, a soft trigger
// command is issued.
//
```

```
void get_label (void)
{

  unsigned char key;              / / Hold area for keypress

  printf ("=====================================\n");
  printf ("Scan a label or strike a key to quit...\n");

  / /
  / / Submit a read with no wait
  / /
  read_label_nowait (&SCB2, &READDATA, &data, DATA_LEN);

  / /
  / / Wait for status to change or ok_2_scan flag goes false
  / /
  while ((SCB2.status == SRTN_PENDING) && ok_2_scan)
  {
    / /
    / / If a key was pressed, then process it
    / /
    if (kbhit())
    {
      key = getch();              / / Get the key value

      if (toupper (key) == 'T')     / / If it was 'T'
        set_softtrig (&SCB3);       / / then set the soft trigger
      else
        ok_2_scan = FALSE;          / / Set flag to FALSE
    }
  }


  / /
  / / If the read status was good and no user abort
  / /
  if ((SCB2.status == SRTN_SUCCESS) && ok_2_scan)
  {
    / /
    / / Display the decoded data information
```

```
        / /
        printf("Status :     %3d \n",SCB2.status);

        printf ("Direction :    ");
        if (READDATA.scan_direction == 1)
          printf("FWD \n");
        else
              if (READDATA.scan_direction == 2)
                    printf("RVS \n");
              else
                    printf ("Error");

        printf("Decoded data : %s \n",data);

        printf("Length :      %d \n",READDATA.label_length);

        show_labeltype (READDATA.labeltype);
      }

      return;
    }
```

```
//
// The following main program establishes a connection to
// the scanner and submits reads until a user abort is issued.
//

void main ( void )
{

  //
  // Ensure that driver has been loaded
  //
  if (!scan_driver_check (&SCB1))
  {
    printf ("Scan driver is not loaded. Scan sample program terminating...\n");
    exit(1);
  }

  //
  // Get the scanner driver version and display on the screen
  //
  get_version_info (&SCB1, &driver_version);
  printf("%Fs",driver_version.version_str);
  printf("Version number : %4x\n", driver_version.version);

  //
  // Enable scanning with wait
  //
  enable_scanning_wait (&SCB1);

  //
  // If successful or scanning was already enabled then continue
  //
  if ((SCB1.retcode == SRTN_SUCCESS) ||
     (SCB1.retcode == SRTN_ALREADY_ENABLED))
  {
    ok_2_scan = TRUE;          // Set flag to true

    //
    // While it is OK to scan, submit a read and wait for either a barcode
    // to be scanned or a key pressed to exit.
    //
```

```
    while (ok_2_scan)
      get_label ();

    / /
    / / Clear the SCB command queue
    / /
    flush_queue (&SCB1);

    / /
    / / Disable scanning with wait
    / /
    disable_scanning_wait (&SCB1);
  }
  else
    printf ("Cannot connect with scanner card.\n");

  return;
}
```

Appendix 2. SCANDEF.H

This appendix contains the header file with structure definitions and defines used in the sample scanning program (scansamp.c) in *Appendix 1* of this chapter.

It is also contained in the SDK file

C:\SDK4100\SAMPLES\SCANNER\SCANDEF.H

where **C:\SDK4100** is the default installation directory.

```
//
// Only include file if SCANDEF_H has not been defined yet
//
#ifndef  SCANDEF_H
#define  SCANDEF_H

//************************************************************
// Nested Include Files ************************************
//
//  None.

// Defines, Typedefs, etc. *********************************



//
// Data must be byte packed
//
#pragma pack (1)
```

```
//
// Define structure for an SCB
//
typedef struct SCB_struc
{
  unsigned char command;        // Command code
  unsigned char retcode;        // Return code
  unsigned char status;         // Command completion status
  unsigned short timeout;       // Command timeout
  void far      *cmdparam;      // Pointer to command parameters
  unsigned char user[4];        // Post processing user parameters
  void far*     (*process)();   // Post processing dispatch address
  char          reserved[21];   // Reserved space
} SCB_type;

//
// Define structure used for Get Version Information subcommand
//
typedef struct VERSION_INFO_struc
{
  void far      *version_str;   // Pointer to version string
  unsigned short version;       // Major and minor version numbers
} VERSION_INFO_type;

//
// Define structure used for Get/Set Reader Parameters subcommands
//
typedef struct READER_MODE_struc
{
  unsigned char parameter_index;        // Index to scanner parameters

  // Common reader parameters
  unsigned char scan_mode;              // Scanner class type attached
  unsigned short enable_settle_time;    // Reader enable settling time
  unsigned short power_settle_time;     // Power settling time. Used only if
                                        // enable_settle_time > 0
  unsigned char inverse_label_flag;     // Inverse data label flag
  unsigned char white_data_logic_lvl;   // White data logic level
  unsigned short dec_beep_time;         // Decode beep time
  unsigned short dec_beep_freq;         // Decode beep frequency
  unsigned char scans_per_label;        // Number of successful decodes for
```

```
                                        // a label before reporting
  unsigned char clk_speed_toggle;       // Clock speed toggle
  unsigned char trans_resolution;       // Transition resolution value
  unsigned char subseq_scan_time;       // Subsequent scan time
  unsigned char no_data_time;           // No data time value
  unsigned char post_dec_action;        // Post decode action
  unsigned char dec_fail_action;        // Decode failure action

  // Laser specific parameters
  unsigned char prod_trigger;           // Produces trigger signal
  unsigned char two_stage_trigger;      // Two stage trigger flag
  unsigned char multiple_scan;          // Automatic multiple scan
  unsigned char prod_direction;         // Produces direction signal
  unsigned char dec_feedb_time;         // Decode LED feedback time
  unsigned char dec_feedback_lvl;       // Decode LED feedback logic level
  unsigned char scan_led_ctl;           // Scanning LED control flag
  unsigned char scan_led_lvl;           // Scanning LED logic level
  unsigned char KE_enable;              // Enable Klasse Eins
  unsigned short KE_time_used;          // Klasse Eins time used
  unsigned short KE_time_left;          // Klasse Eins time left

  // Contact specific parameters
  unsigned char qz_ratio;               // Quiet zone ratio X:1 to 1:X
  unsigned char init_scan_time;         // Initial scan time
  unsigned short pulse_delay;           // Pulse delay time
} READER_MODE_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands

//
typedef struct UPCE0_PARMS_struc
{
  unsigned char upc_e_chk_b;        // Report UPC E0 check byte
  unsigned char upce_preamble;      // UPC E0 preamble
  unsigned char conv_upce2a_b;      // Convert UPC E0 to UPC A
} UPCE0_PARMS_type;
```

```
//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct UPCE1_PARMS_struc
{
  unsigned char upc_e1_chk_b;       // Report UPC E1 check byte
  unsigned char upce1_preamble;     // UPC E1 preamble
  unsigned char conv_upce1_2a_b;    // Convert UPC E1 to UPC A
} UPCE1_PARMS_type;



//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct UPCA_PARMS_struc
{
  unsigned char upc_a_chk_b;        // Report UPC A check byte
  unsigned char upca_preamble;      // UPC A preamble
} UPCA_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct MSI_PARMS_struc
{
  unsigned char cmsi_red_enabled;   // MSI Redundancy enabled
  unsigned char msi_chk_dgt;        // Number of MSI check digits
  unsigned char report_msi_chk;     // Report MSI check digits
} MSI_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct EAN8_PARMS_struc
{
  unsigned char conv_ean8to13_b;    // Convert EAN 8 to EAN 13
} EAN8_PARMS_type
```

```
//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct CBAR_PARMS_struc
{
  unsigned char cbar_red_enabled;       // Codabar Redundancy enabled
  unsigned char clsi_editing;           // Enable clsi editing
  unsigned char notis_editing;          // Enable notis editing
} CBAR_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct C39_PARMS_struc
{
  unsigned char code39_chk_b;           // Enable/Report Code 39 check digit
  unsigned char buffer_c39;             // Enable Buffer Code 39
  unsigned char code39_full_ascii;      // Enable Code 39 full ASCII
  unsigned char c39_red_enabled;        // Code 39 Redundancy enabled
} C39_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct D25_PARMS_struc
{
  unsigned char cd25_red_enabled;     // D 2 of 5 Redundancy enabled
} D25_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct I25_PARMS_struc
{
  unsigned char ci25_red_enabled;   // I 2 of 5 Redundancy enabled
} I25_PARMS_type;
```

```
//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct C11_PARMS_struc
{
  unsigned char c11_red_enabled;  // Code 11 Redundancy enabled
  unsigned char c11_chk_dgt;      // Number of Code 11 check digits
  unsigned char report_c11_chk;   // Report Code 11 check digits
} C11_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct C93_PARMS_struc
{
  unsigned char c93_red_enabled;   // Code 93 Redundancy enabled
} C93_PARMS_type;

//
// Decoder specific parameters for Get/Set Decoder Parameters subcommands
//
typedef struct C128_PARMS_struc
{
  unsigned char c128_red_enabled;  // Code 128 Redundancy enabled
} C128_PARMS_type;
```

```
//
// Define structure used for Get/Set Decoder Parameters subcommands
//
typedef struct DECODER_PARMS_struc
{
  unsigned char labeltype;          // Barcode Symbology Type
  unsigned short minlength;         // Minimum Length of Barcode
  unsigned short maxlength;         // Maximum Length of Barcode
  unsigned char alloc_specific;     // Number of allocated parameters
                                    // in decoder_specific
  union
  {
    UPCE0_PARMS_type upce0_parms;   // UPC E0 specific parameters
    UPCE1_PARMS_type upce1_parms;   // UPC E1 specific parameters
    UPCA_PARMS_type  upca_parms;    // UPC A specific parameters
    MSI_PARMS_type   msi_parms;     // MSI specific parameters
    EAN8_PARMS_type  ean8_parms;    // EAN 8 specific parameters
    CBAR_PARMS_type  cbar_parms;    // Codabar specific parameters
    C39_PARMS_type   c39_parms;     // Code 39 specific parameters
    D25_PARMS_type   d25_parms;     // D 2 of 5 specific parameters
    I25_PARMS_type   i25_parms;     // I 2 of 5 specific parameters
    C11_PARMS_type   c11_parms;     // Code 11 specific parameters
    C93_PARMS_type   c93_parms;     // Code 93 specific parameters
    C128_PARMS_type  c128_parms;    // Code 128 specific parameters
  } decoder_specific;
} DECODER_PARMS_type;

//
// Define structure used for Get/Set UPC/EAN General Parameters
//subcommands
//
typedef struct UPC_GEN_PARMS_struc
{
  unsigned char security_level;     // Security Level Value
  unsigned char supp_2;             // Two Digit Supplementals
  unsigned char supp_5;             // Five Digit Supplementals
  unsigned char supp_auto_d;        // Auto-discriminate Supplementals
  unsigned char supp_retry;         // Retry count for auto-discriminate
                                    // supplementals
  unsigned char linear_decode;      // Linear Decode all UPC types
} UPC_GEN_PARMS_type;
```

```
//
// Define structure used for Get Scan Status subcommand
//
typedef struct SCAN_STATUS_struc
{
  unsigned char scan_state;          // Scan State
  unsigned char status;              // Current scan status
} SCAN_STATUS_type;

//
// Define structure used for Get/Set Trigger Mode subcommands
//
typedef struct TRIG_MODE_struc
{
  unsigned char trigger_mode;      // Trigger mode
} TRIG_MODE_type;

//
// Define structure used for Read Label subcommand
//
typedef struct READ_struc
{
  unsigned char labeltype;           // Returned Label Type
  void far      *data_buf_ptr;       // Pointer to Data Buffer
  unsigned short data_buf_len;       // Length of Data Buffer
  unsigned short label_length;       // Length of Data in Buffer
  unsigned char scan_direction;      // Decode Direction
  unsigned char read_status;         // Scanning status for this read
} READ_type;

//
// Define structure used for Get Supported Decoders, Get/Set Enabled
// Decoders subcommands
//
typedef struct DECODER_STRING_struc
{
  char decoder_str[25];              // Null terminated decoder type
                                     // string
} DECODER_STRING_type;
```

```
//
// Define structure used for Cancel SCB subcommand
//
typedef struct CAN_SCB_struc
{
  SCB_type far  *SCB_ptr;            // Pointer to SCB to be canceled
} CAN_SCB_type;

//
// Define structure used for Get Number of Pending SCBs subcommand
//
typedef struct NUM_PEND_struc
{
  unsigned short pending;            // Number of entries in command
                                     // queue
} NUM_PEND_type;

//
// Subcommand number definitions
//
#define CMD_GET_VERSION_INFO      0x00  // Get Version Information
#define CMD_GET_READER_PARMS      0x01  // Get Reader Parameters
#define CMD_SET_READER_PARMS      0x02  // Set Reader Parameters
#define CMD_GET_SCAN_PARMS        0x03  // Get Scanning Parameters
#define CMD_SET_SCAN_PARMS        0x04  // Set Scanning Parameters
#define CMD_GET_DECODER_PARMS     0x05  // Get Decoder Parameters
#define CMD_SET_DECODER_PARMS     0x06  // Set Decoder Parameters
#define CMD_GET_UPCEAN_PARMS      0x07  // Get UPC/EAN General
                                        // Parameters
#define CMD_SET_UPCEAN_PARMS      0x08  // Set UPC/EAN General
                                        // Parameters

#define CMD_ENABLE_SCANNING_WAIT     0x09    // Enable Scanning
                                             // with wait
#define CMD_DISABLE_SCANNING_WAIT    0x0A    // Disable Scanning
                                             //with wait
```

```
#define CMD_GET_SCAN_STATUS   0x0B  / / Get Scanning Status
#define CMD_GET_TRIG_MODE     0x0C  / / Get Triggering Mode
#define CMD_SET_TRIG_MODE     0x0D  / / Set Triggering Mode
#define CMD_READ_LABEL_WAIT   0x0E  / / Read Label with wait
#define CMD_SET_SOFTTRIG      0x0F  / / Set Soft Trigger
#define CMD_CLR_SOFTTRIG      0x10  / / Clear Soft Trigger
#define CMD_GET_SUPP_DEC      0x11  / / Get Supported Decoders
#define CMD_GET_ENAB_DEC      0x12  / / Get Enabled Decoders
#define CMD_SET_ENAB_DEC      0x13  / / Set Enabled Decoders
#define CMD_CANCEL_SCB        0x14  / / Delete SCB
#define CMD_FLUSH             0x15  / / Flush Command Queue
#define CMD_GET_PEND_SCB      0x16  / / Get Number of Pending SCBs

#define CMD_ENABLE_SCANNING_NOWAIT    0x89  / / Enable Scanning
                                            / / with no wait
#define CMD_DISABLE_SCANNING_NOWAIT   0x8A  / / Disable Scanning
                                            / / with no wait
#define CMD_READ_LABEL_NOWAIT         0x8E  / / Read Label with
                                            / / no wait
#define CMD_INVALID                   0xFF  / / Invalid command
                                            / / number

/ /
/ / SCB return codes
/ /
#define SRTN_SUCCESS              0    / / Successful completion
#define SRTN_UNKNOWN_CMD          1    / / Unknown command
#define SRTN_INVALID_SCB          2    / / Invalid SCB address or length
#define SRTN_INVALID_PARAM_PTR    3    / / Invalid command parameter
                                       / / pointer
#define SRTN_INVALID_PARAM        5    / / Invalid command parameter
#define SRTN_CMD_TIMEOUT          9    / / Command timed-out
#define SRTN_CMD_CANCEL           10   / / Command was canceled
#define SRTN_CMD_COMPLETE         12   / / Command already complete
#define SRTN_NO_CONNECT           13   / / No connection with scan card
#define SRTN_NOT_ENABLED          14   / / Scanning not enabled
#define SRTN_CONN_ABORT           15   / / Scanning aborted
#define SRTN_PHYS_FAIL            16   / / Physical device failure
#define SRTN_DATA_OVERFLOW        17   / / Data overflow from read buffer
#define SRTN_CMDQ_FULL            18   / / SCB command queue full
```

```
#define SRTN_ALREADY_ENABLED    19      / / Scanning already enabled
#define SRTN_ALREADY_DISABLED   20      / / Scanning already disabled
#define SRTN_INVALID_DBUFF_PTR  21      / / Invalid data buffer pointer
#define SRTN_INVALID_DEL_PTR    22      / / Invalid SCB pointer to delete
#define SRTN_NODATA             23      / / No data from scanner
#define SRTN_SCBINUSE           24      / / SCB already in use
#define SRTN_QNOTEMPTY          25      / /Command queue is not empty
#define SRTN_PENDING            255     / / SCB is pending

/ /
/ / Define structure used for Get/Set Scan Parameters subcommands
/ /
typedef struct SCAN_PARMS_struc
{
  unsigned char bidir_redundancy;     / / Bi Directional redundancy
  unsigned char xmit_code_id;         / / Transmit code id character before
                                      / / decoded data
} SCAN_PARMS_type;

/ /
/ / Label type definitions
/ /
#define TYPE_UPCE0      0x30    / / Label type for UPC E0
#define TYPE_UPCE1      0x31    / / Label type for UPC E1
#define TYPE_UPCA       0x32    / / Label type for UPC A
#define TYPE_MSI        0x33    / / Label type for UPC E0
#define TYPE_EAN8       0x34    / / Label type for EAN 8
#define TYPE_EAN13      0x35    / / Label type for EAN 13
#define TYPE_CODABAR    0x36    / / Label type for CODABAR
#define TYPE_CODE39     0x37    / / Label type for Code 39
#define TYPE_D2OF5      0x38    / / Label type for Discrete 2 of 5
#define TYPE_I2OF5      0x39    / / Label type for Interleaved 2 of 5
#define TYPE_CODE11     0x3A    / / Label type for Code 11
#define TYPE_CODE93     0x3B    / / Label type for Code 93
#define TYPE_CODE128    0x3C    / / Label type for Code 128
```

```
//
// Define scan driver interrupt number
//
#define SCAN_INT              0x62       // Scan driver interrupt value


#define THREE_SECONDS     3000       // Three second timeout value

//
// Define TRUE and FALSE
//
#ifndef TRUE
#define TRUE   1
#endif

#ifndef FALSE
#define FALSE 0
#endif

//
// Revert to default packing structures
//
#pragma pack ()

#endif                        // end ifndef SCANDEF_H
```

Appendix 3. SCANPROT.H

This appendix contains the header file with prototype definitions for functions used in the sample scanning program (scansamp.c) in *Appendix 1* of this chapter. It is also contained in the SDK file

C:\SDK4100\SAMPLES\SCANNER\SCANPROT.C

where **C:\SDK4100** is the default installation directory.

```
//
// Only include file if SCANPROT_H has not been defined yet
//
ifndef SCANPROT_H
#define SCANPROT_H

//*************************************************************************
// Function Prototypes

void submit_SCB          (struct SCB_struc _far *SCB);
char scan_driver_check   (struct SCB_struc _far *SCB);
void get_version_info    (struct SCB_struc _far *SCB,
                          struct VERSION_INFO_struc _far *ver_data);
void get_reader_parms    (struct SCB_struc _far *SCB,
                          unsigned char index,
                          struct READER_MODE_struc _far *reader_data);
void set_reader_parms    (struct SCB_struc _far *SCB,
                          unsigned char index,
                          struct READER_MODE_struc _far *reader_data);
void get_scan_parms      (struct SCB_struc _far *SCB,
                          struct SCAN_PARMS_struc _far *scan_data);
void set_scan_parms      (struct SCB_struc _far *SCB,
                          struct SCAN_PARMS_struc _far *scan_data);
void get_decoder_parms   (struct SCB_struc _far *SCB,
                          struct DECODER_PARMS_struc _far *decoder_data);
void set_decoder_parms     (struct SCB_struc _far *SCB,
                          struct DECODER_PARMS_struc _far *decoder_data);
void get_upcean_parms    (struct SCB_struc _far *SCB,
                          struct UPC_GEN_PARMS_struc _far *upcean_data);
void set_upcean_parms    (struct SCB_struc _far *SCB,
                          struct UPC_GEN_PARMS_struc _far *upcean_data);
```

```
void enable_scanning_wait       (struct SCB_struc _far *SCB);
void disable_scanning_wait      (struct SCB_struc _far *SCB);

void get_scan_status     (struct SCB_struc _far *SCB,
                          struct SCAN_STATUS_struc _far *scanstat_data);
void get_trig_mode       (struct SCB_struc _far *SCB,
                          struct TRIG_MODE_struc _far *trig_data);
void set_trig_mode       (struct SCB_struc _far *SCB,
                          struct TRIG_MODE_struc _far *trig_data);
void read_label_wait     (struct SCB_struc _far *SCB,
                          struct READ_struc _far *read_data,
                          void   _far *buffer,unsigned short buffer_len);

void set_softtrig      (struct SCB_struc _far *SCB);
void clr_softtrig      (struct SCB_struc _far *SCB);

void get_supported_decoders (struct SCB_struc _far *SCB,
                                 struct DECODER_STRING_struc _far *support_data);

void get_enabled_decoders     (struct SCB_struc _far *SCB,
                               struct DECODER_STRING_struc _far *enable_data);

void set_enabled_decoders     (struct SCB_struc _far *SCB,
                               struct DECODER_STRING_struc _far *enable_data);

void cancel_scb    (struct SCB_struc _far *SCB,
                    struct CAN_SCB_struc _far *cancel_data,
                    struct SCB_struc _far *canSCB);

void flush_queue    (struct SCB_struc _far *SCB);
void get_pending    (struct SCB_struc _far *SCB,
                     struct NUM_PEND_struc _far *pend_data);

void enable_scanning_nowait (struct SCB_struc _far *SCB);

void disable_scanning_nowait (struct SCB_struc _far *SCB);

void read_label_nowait       (struct SCB_struc _far *SCB,
                              struct READ_struc _far *read_data,
                              void   _far *buffer,
                              unsigned short buffer_len);

void show_labeltype (unsigned char ltype);
```

```
void get_label (void);


#endif                  / / end ifndef SCANPROT_H
```