Symbol Native Application Programming Interface (SNAPI)

Programmer Guide

symbol®
The Enterprise Mobility Company ™

Symbol Native Application Programming Interface Programmer Guide

72E-71370-01

Revision A

March 2006

symbol®
The Enterprise Mobility Company™

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

Patents

This product is covered by one or more of the patents listed on the website: www.symbol.com/patents.

Revision History

Changes to the original manual are listed below:

Change	Date	Description
-01 Rev A	2/2006	Initial Rev A Release.

Table of Contents

About This Guide

Introduction

This guide describes the Symbol Native Application Programming Interface (SNAPI). SNAPI is a development library used to implement USB communication between a Symbol Technologies scanner decoder and a Windows 98/2000/XP host. This guide is intended for the user of a SNAPI product.This includes end users, system administrators, or programmers using this guide as a reference for SNAPI.

Chapter Descriptions

- *Chapter 1, DLL Data, Error Reporting, Messages & Beep Codes* provides a SNAPI overview and information about the SNAPI DLL, messages, error codes and beep codes.
- *Chapter 2, SNAPI Functions* provides a summary of SNAPI functions by type and a detailed alphabetical listing of all SNAPI functions with their syntax, parameters, return values and examples.

Notational Conventions

The following conventions are used in this document:

- "User" refers to anyone using a SNAPI compatible product.
- "You" refers to the End User, System Administrator or Programmer using this manual as a reference for SNAPI.
- *Italics* are used to highlight the following:
  - Chapters and sections in this and related documents
  - Specific items in the general text
  - Dialog boxes and tabs within dialog boxes.
- bullets (•) indicate:
  - Action items
  - Lists of alternatives
  - Lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Documents

- Refer to the *Product Reference Guide* or *Integration Guide* for your scanner for scanner-specific information on SNAPI.

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

Service Information

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

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

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

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

Symbol Support Center

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

Country/Region	Address	Telephone
United States	Symbol Technologies, Inc. One Symbol Plaza Holtsville, New York 11742-1300	1-800-653-5350
Canada	Symbol Technologies Canada, Inc. 5180 Orbitor Drive Mississauga, Ontario, Canada L4W 5L9	1-866-416-8545 (Inside Canada) 905-629-7226 (Outside Canada)
United Kingdom	Symbol Technologies Symbol Place Winnersh Triangle, Berkshire RG41 5TP United Kingdom	0800 328 2424 (Inside UK) +44 118 945 7529 (Outside UK)
Asia/Pacific	Symbol Technologies Asia, Inc. (Singapore Branch) 230 Victoria Street #12-06/10 Bugis Junction Office Tower Singapore 188024	Tel: +65-6796-9600 Fax: +65-6337-6488
Australia	Symbol Technologies Pty. Ltd. 432 St. Kilda Road Melbourne, Victoria 3004	1-800-672-906 (Inside Australia) +61-3-9866-6044 (Outside Australia
Austria/Österreich	Symbol Technologies Austria GmbH Prinz-Eugen Strasse 70 / 2.Haus 1040 Vienna, Austria 01-5055794-0 (Inside Austria)	+43-1-5055794-0 (Outside Austria)
Denmark/Danmark	Symbol Technologies AS Dr. Neergaardsvej 3 2970 Hørsholm	7020-1718 (Inside Denmark) +45-7020-1718 (Outside Denmark)

Country/Region	Address	Telephone
Europe/Mid-East Distributor Operations		Contact your local distributor or call +44 118 945 7360
Finland/Suomi	Oy Symbol Technologies Kaupintie 8 A 6 FIN-00440 Helsinki, Finland	9 5407 580 (Inside Finland) +358 9 5407 580 (Outside Finland)
France	Symbol Technologies France Centre d'Affaire d'Antony 3 Rue de la Renaissance 92184 Antony Cedex, France	01-40-96-52-21 (Inside France) +33-1-40-96-52-50 (Outside France)
Germany/ Deutschland	Symbol Technologies GmbH Waldstrasse 66 D-63128 Dietzenbach, Germany	6074-49020 (Inside Germany) +49-6074-49020 (Outside Germany)
Italy/Italia	Symbol Technologies Italia S.R.L. Via Cristoforo Columbo, 49 20090 Trezzano S/N Navigilo Milano, Italy	2-484441 (Inside Italy) +39-02-484441 (Outside Italy)
Latin America Sales Support	2730 University Dr. Coral Springs, FL 33065 USA	1-800-347-0178 (Inside United States) +1-954-255-2610 (Outside United States) 954-340-9454 (Fax)
Mexico/México	Symbol Technologies Mexico Ltd. Torre Picasso Boulevard Manuel Avila Camacho No 88 Lomas de Chapultepec CP 11000 Mexico City, DF, Mexico	5-520-1835 (Inside Mexico) +52-5-520-1835 (Outside Mexico)
Netherlands/Nederland	Symbol Technologies Kerkplein 2, 7051 CX Postbus 24 7050 AA Varsseveld, Netherlands	315-271700 (Inside Netherlands) +31-315-271700 (Outside Netherlands)
Norway/Norge	Symbol's registered and mailing address: Symbol Technologies Norway Hoybratenveien 35 C N-1055 OSLO, Norway Symbol's repair depot and shipping address: Symbol Technologies Norway Enebakkveien 123 N-0680 OSLO, Norway	+47 2232 4375

Country/Region	Address	Telephone
South Africa	Symbol Technologies Africa Inc. Block B2 Rutherford Estate 1 Scott Street Waverly 2090 Johannesburg Republic of South Africa	11-809 5311 (Inside South Africa) +27-11-809 5311 (Outside South Africa)
Spain/España	Symbol Technologies S.L. Avenida de Bruselas, 22 Edificio Sauce Alcobendas, Madrid 28108 Spain	91 324 40 00 (Inside Spain) +34 91 324 40 00 (Outside Spain) Fax: +34.91.324.4010
Sweden/Sverige	“Letter” address: Symbol Technologies AB Box 1354 S-171 26 SOLNA Sweden Visit/shipping address: Symbol Technologies AB Solna Strandväg 78 S-171 54 SOLNA Sweden	Switchboard: 08 445 29 00 (domestic) Call Center: +46 8 445 29 29 (international) Support E-Mail: Sweden.Support@se.symbol.com

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

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

Chapter 1

DLL Data, Error Reporting, Messages & Beep Codes

Introduction

SNAPI is a transaction-based protocol that enables communication with SNAPI devices over USB. The SNAPI DLL implements:

- USB HID communications
- SNAPI USB Imaging Driver Communications
- SNAPI message building
- SNAPI protocol handling needed to provide a communications link between Symbol scanners and a Windows host.

To set the host communication option to SNAPI, scan the SNAPI Host parameter in the user documentation for the scanner. Scanning this parameter is not required if SNAPI is the default host for the scanner.

A command function returns immediately to the host application while the scanner processes the command. After the command is processed by the scanner, the host application receives a Windows message indicating the command was processed. The host application provides a message handler for the acknowledgement from the connected SNAPI device. Do not initiate another command until the acknowledgement is complete.

The Windows host program also receives Windows messages when the decoder has data to send to the host or when a timeout or error occurs. Further, the Windows host program must allocate memory to be used to return data from the scanner to the application.

All SNAPI API functions are declared in SNAPIdll.h.

Data Returned by DLL

The host application should allocate a destination data buffer for use by the DLL. When the scanner sends data to the DLL, the destination buffer is filled with the scanner's data.

Once the destination buffer is filled by the DLL (if a buffer was set), the application is sent a WM_xxx message with the number of bytes of data that were stored indicated in the WPARAM.

If the buffer was not large enough to hold all the data, the last two bits of WPARAM are set to zero. If no buffer was given to the DLL for the data to be stored in, the last two bits of WPARAM are 01. If the data was stored correctly, the last two bits of WPARAM are 11.

The following #defines are provided for this purpose:

- BUFFERSIZE_MASK – 0x0003
- BUFFERSIZE_GOOD – 0x0003
- BUFFERSIZE_ERROR – 0x0000
- NOBUFFER_ERROR – 0x0001

After the message is sent, the DLL marks the buffer as NULL indicating no user buffer is available for storage. The host application should reset the buffer after a WM_xxx message occurs.

A second call to set the data buffer causes the new buffer to be used for incoming data. For example, after a WM_DECODE message is sent to the application, the application should handle the message and process the data in the destination buffer, then call SNAPI_SetDecodeBuffer again.

Windows Messages Sent to Calling Process

Table 1-1 defines the available Windows messages sent to a calling process.

Table 1-1 *Windows Messages*

Attribute	Description
Message	WM_DECODE See *WM_DECODE on page A-4* for an example of registering for the parameter data event and interpreting incoming parameter data.
Description	Decode data is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetDecodeBuffer.
Value	WM_APP+1
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The buffer status of the data stored. • HIDWORD (wparam) – The length of the data in bytes. • lParam – Handle to the device for which the message was posted.
Data Format	The bar code type is stored in the first word followed by the decoded message. See Appendix A, Bar Code Types & Code Examples for the codes for each bar code type.
Message	WM_IMAGE
Description	Image data is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetImageBuffer.
Value	WM_APP+2
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *) • LODWORD (wparam) – The buffer status of the data stored • HIDWORD (wparam) – The length of the data in bytes • lParam – Handle to the device for which the message was posted
Data Format	Actual image data.
Message	WM_VIDEOIMAGE
Description	A video frame is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetVideoBuffer.
Value	WM_APP+3

Table 1-1 *Windows Messages (Continued)*

Attribute	Description
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *) • LODWORD (wparam) – The buffer status of the data stored • HIDWORD (wparam) – The length of the data in bytes • lParam – Handle to the device for which the message was posted
Data Format	Actual video frame data.
Message	WM_ERROR
Description	An error occurred. This message may be sent in response to an API command or request for data.
Value	WM_APP+4
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *) • LODWORD (wparam) – The error code (cast to signed short). (See *WM_ERROR Codes on page 1-9*). • lParam – Handle to the device for which the message was posted
Message	WM_TIMEOUT
Description	Scanner did not respond to a request from the library within the timeout period during processing of unsolicited data (decode data, image data or video data) from the scanner.
Value	WM_APP+5
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *) • LODWORD (wparam) – Set to zero (reserved for future use). • HIDWORD (wparam) – The request code (int) that did not receive the response. (See *WM_TIMEOUT Codes on page 1-10*.) • lParam – Handle to the device for which the message was posted
Message	WM_CMDCOMPLETEMSG
Description	Verifies that the scanner handled the command that was issued. API functions that request data from the scanner do not trigger this message.
Value	WM_APP+6
Parameters	lParam – Handle to the device for which the message was posted.
Message	WM_XFERSTATUS
Description	Image data is transferring from the scanner.

Table 1-1 *Windows Messages (Continued)*

Attribute	Description
Value	WM_APP+7
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The total number of bytes received so far. • HIDWORD (wparam) – The total number of bytes expected. • lParam – Handle to the device for which the message was posted.
Message	WM_SWVERSION
Description	Software version information is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetVersionBuffer. This message is received in response to an API request for version data.
Value	WM_APP+8
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The buffer status code. • HIDWORD (wparam) – The length of the data in bytes. • lParam – Handle to the device for which the message was posted.
Data Format	Revision data in the following format: <SW_REVISION> <HW_REVISION> <ENGINE_INFO> <CLASS_INFO> where: • SW_REVISION is the release name of the software. • HW_REVISION is the release name of the hardware. • ENGINE_INFO indicates the type of scan engine paired with the decoder. (Refer to the scan engine Integration Guide for the engine code value.) • CLASS_INFO is Device Class info.

Table 1-1 *Windows Messages (Continued)*

Attribute	Description
Message	WM_PARAM See *WM_PARAM on page A-5* for an example of registering for the parameter data event and interpreting incoming parameter data.
Description	Parameter information is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetParameterBuffer. This message is received in response to an API request for parameter data. ***NOTE*** Valid parameter numbers applicable to a scanner can be found in the scanner *Product Reference Guide* or *Integration Guide*. Parameter numbers included in the scanner guides are in SSI (Simple Serial Interface) format and must be converted into the actual parameter number represented as word values. To convert parameter numbers from SSI format to the actual parameter number, the SSI parameter offset value must be taken into consideration. SSI parameters may or may not begin with an offset value (e.g., F0, F1, etc.). Conversion rules apply as follows: - When an SSI parameter number has no offset, the SSI and SNAPI parameter numbers are the same and the SSI parameter number can be used. - When an SSI parameter number has an offset and the offset is F0, add 0x100 to the parameter number. For example, the *Product Reference Guide* includes **Composite Beep Mode** with the SSI parameter number F0h, 8Eh - where F0h (0xF0) is the offset and 8Eh (0x8E) is the number. To convert this SSI parameter to a SNAPI parameter add 0x100 to 0x8E, resulting in 0x18E. When the SSI offset is 0xF1, add 0x200. When the SSI offset is 0xF2, add 0x300. Because all SNAPI parameter numbers are word values, SSI offsets of F4 should be ignored. For example, the top **Crop to Pixel Addresses** SSI parameter number (included in the *Product Reference Guide*) is F4h, F0h, 3Bh. F4h is ignored and the SNAPI parameter number is 0x13B.
Value	WM_APP+9
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The buffer status code. • HIDWORD (wparam) – The length of the parameter data as number of words. • lParam – Handle to the device for which the message was posted.
Data Format	Parameter numbers and values will always be two bytes (one word)) each.
Message	WM_CAPABILITIES

Table 1-1 *Windows Messages (Continued)*

Attribute	Description
Description	Capabilities data is available from the scanner and is stored in the buffer provided by a previous call to SNAPI_SetCapabilitiesBuffer. This message is received in response to an API request for capabilities data.
Value	WM_APP+10
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The buffer status code.•HIDWORD (wparam) – The length of the capabilities data in bytes. • lParam – Handle to the device for which the message was posted.
Message	WM_EVENT
Description	Event data is available from the scanner. No destination buffer is required for this unsolicited data from the scanner; the data is sent in the WPARAM along with the message. Refer to the scanner *Product Reference Guide* or *Integration Guide* for event values.
Value	WM_APP+11
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) – The event data. • HIDWORD (wparam) – The length of data in bytes (always 1). • lParam – Handle to the device for which the message was posted.
Message	WM_DEVICENOTIFICATION
Description	Notification from DLL for attach or removal of USB Scanner.
Value	WM_APP+12
Parameters	• wParam – Pointer to DWPARAM structure (cast to DWPARAM *). • LODWORD (wparam) is either of DEVICE_ARRIVE or DEVICE_REMOVE • lParam – Handle to the device for which the message was posted. ☑ ***NOTE*** This handle may be used as the device handle for API function calls.

Error Reporting

All SNAPI library function calls return zero (0) if successful, or an error code. *Table 1-2* describes the errors that can be reported. If the error code is a fatal error, call SNAPI_Disconnect.

☑ ***NOTE*** Not all error codes in *Table 1-2* apply to SNAPI. Error codes marked “Not Applicable for SNAPI” apply to the SSI protocol only.

Table 1-2 *Error Codes*

Define Name	Value	Description
SSICOMM_NOERROR	0	No error code is set; an API call was successful.
ERR_SSI_NOOBJECT	-1	Not a valid Device Object (Handle)
ERR_SSI_HWND	-2	The hwnd parameter to the SNAPI_Init function was NULL.
SSICOMM_BAD_SETSTATE	-3	The library was unable to set the state of the COM port; no connection established. (Not applicable for SNAPI)
SSICOMM_BADSETTIMEOUTS	-4	The library was unable to set the COM timeouts; no connection established. (Not applicable for SNAPI)
SSICOMM_BAD_GETTIMEOUTS	-5	The library was unable to get the current COM timeouts; no connection established. (Not applicable for SNAPI)
SSICOMM_BAD_GETCOMSTATE	-6	The library was unable to get the current COM state; no connection established. (Not applicable for SNAPI)
SSICOMM_ALREADY_CLOSED	-7	Call to disconnect was made when the device is port is not connected; there is no connection.
SSICOMM_UNABLE_PURGE	-8	Call to purge the COM port before closing it was not successful. (Not applicable for SNAPI)
SSICOMM_THREADS_BADEXIT	-9	Fatal error — the threads didn't exit properly.
SSICOMM_ERROR_CLRDTR	-10	Unable to lower DTR when closing COM port. (Not applicable for SNAPI)
SSICOMM_BAD_CREATEFILE	-11	Unable to open COM port. (Not applicable for SNAPI)
SSICOMM_BAD_READTHREAD	-12	Unable to create the read/status thread; no connection.
SSICOMM_BAD_WRITETHREAD	-13	Unable to create the writer thread; no connection.
SSICOMM_BAD_CREATEEVENT	-14	Call to CreateEvent failed; fatal error.
SSICOMM_BUSY	-15	Not fatal; try request again later.
SSICMD_UNIMPLEMENTED	-16	Not fatal; this command is not implemented in the library.
SSICOMM_ALREADYCONNECTED	-17	If already connected, can't connect without a call to disconnect.
ERR_SSI_MISMATCHHWND	-18	The hwnd parameter for the function does not match the stored hwnd for the connection.
SSICOMM_TOOMUCHDATA	-19	The maximum allowable input data length was exceeded.
SSICOMM_ERRVERSION	-20	Can't run on this version of Windows.
SSI_INPUTQ_FULL	-21	Unable to add new user request to input queue for transmitting to scanner; re-try request.
SSICOMM_BADDATA	-22	Parameter data is in incorrect format.

WM_ERROR Messages

In addition to the failure status returned by library function and error codes, the library can also send or post a WM_ERROR message to the application. The application handles the message and responds appropriately.

The following generate WM_ERROR messages to the calling application. Most are fatal errors occurring during program execution, and are returned in the WPARAM associated with the WM_ERROR message.SSICOMM_BADDATA-22Parameter data is in incorrect format.

Table 1-3 WM_ERROR *Codes*

Define Name	Value	Description
SSICOMM_WAITMOWRITER	-23	Wait for Multiple Objects resulted in WAIT_FAILED in writer procedure; if not fatal, protocol retry may recover.
SSITHREAD_CREATEWEVENT	-24	Failure to create write event; fatal error.
SSITHREAD_OLRESW	-25	Get overlapped result failed; fatal error.
SSITHREAD_WRITEERR	-26	Number of bytes written is not the number requested to be written. If not fatal, retry may recover.
SSITHREAD_WMOW	-27	Wait multiple objects failure in overlapped write; fatal.
SSITHREAD_WRITEFILEFAIL	-28	Call to Write failed, but is not just delayed; fatal error.
SSITHREAD_BADSETEV	-29	Write thread returned error on set event.
SSIRTHREAD_ORESULT	-30	Read thread bad overlapped result; fatal error.
SSIRTHREAD_SETMASK	-31	Read thread bad set mask return; fatal error. (Not applicable for SNAPI)
SSIRTHREAD_BADREAD	-32	Read thread bad read; fatal error.
SSIRTHREAD_CREATEREVENT	-33	Read thread bad create read event; error code set, API call will return false.
SSIRTHREAD_CREATESEVENT	-34	Read thread bad create status event; error code set, API call will return false.
SSIRTHREAD_WAITCEVENT	-35	Read thread wait COM event bad return; fatal error. (Not applicable for SNAPI.)The following error codes are placed in the wParam of WM_ERROR messages during SNAPI protocol handling of scanner messages.
COMMAND_NOTHANDLED	-36	Command not processed successfully by decoder.
ERR_UNSUPPORTED_COMMAND	-37	Command not processed successfully by decoder.
SSI_DATAFORMAT_ERR	-38	Scanner data packet not of correct format from decoder.
ERR_UNEXPECTEDDATA	-39	State machine received data unexpected for the current state.

WM_TIMEOUT Messages

Table 1-4 lists the OPCODES placed in the LPARAM of the SSI WM_TIMEOUT message.

Table 1-4 *WM_TIMEOUT Codes*

Error	Value
DECODE_DATA_TIMEOUT	0xF3
IMAGE_DATA_TIMEOUT	0xB1
VIDEO_DATA_TIMEOUT	0xB4

Beep Codes

Table 1-5 lists the beep codes for the sound beeper function.

Table 1-5 *Beep Codes*

Define Name	Value
ONESHORTHI	0x00
TWOSHORTH	I0x01
THREESHORTH	I0x02
FOURSHORTHI	0x03
FIVESHORTHI	0x04
ONESHORTLO	0x05
TWOSHORTLO	0x06
THREESHORTLO	0x07
FOURSHORTLO	0x08
FIVESHORTLO	0x09
ONELONGHI	0x0A
TWOLONGHI	0x0B
THREELONGHI	0x0C
FOURLONGHI	0x0D
FIVELONGHI	0x0E
ONELONGLO	0x0F
TWOLONGLO	0x10
THREELONGLO	0x11

Table 1-5 *Beep Codes (Continued)*

Define Name	Value
FOURLONGLO	0x12
FIVELONGLO	0x13
FASTHILOHILO	0x14
SLOWHILOHILO	0x15
HILO	0x16
LOHI	0x17
HILOHI	0x18
LOHILO	0x19

Introduction

Table 2-1 lists SNAPI functions by the type of function. Click the page reference in the table to jump directly to the command details.

SNAPI function details (command details), beginning on *page 2-4*, are listed alphabetically.

Table 2-1 *SNAPI Functions by Type*

Command	Brief Description	Page
Data Request Commands		
SNAPI_GetSerialNumber	Returns the serial number of the connected scanner.	*2-9*
SNAPI_RequestParameters	Requests the scanner to send parameter values specified in a given parameter string.	*2-14*
SNAPI_RequestScannerCapabilities	Sends a request to the scanner to send its capabilities data, i.e., the commands it can perform.	*2-16*
SNAPI_TransmitVersion	Sends a request to the scanner to send its software release name.	*2-30*
Host to Scanner Commands		
SNAPI_AbortMacroPdf	Sends a command to the scanner to abort the current MacroPDF scanning session and discard any stored MacroPDF data.	*2-4*
SNAPI_AimOff	Sends a command to the scanner to turn off the aiming pattern of an imager.	*2-5*
SNAPI_AimOn	Sends a command to the scanner to turn on the aiming pattern of an imager.	*2-5*
SNAPI_EnterLowPwrMode	Sends a command to the scanner to enter low power mode.	*2-8*
SNAPI_FlushMacroPdf	Sends a command to the scanner to abort the current MacroPDF scanning session and transmit stored MacroPDF data.	*2-8*
SNAPI_LedOff	Sends a command to the scanner to turn off the LED on the scanner.	*2-11*
SNAPI_LedOn	Sends a command to the scanner to turn on the LED of the scanner.	*2-12*

Table 2-1 *SNAPI Functions by Type (Continued)*

Command	Brief Description	Page
SNAPI_PullTrigger	Sends a command to the scanner to perform a software trigger, causing the scanner to behave as if the trigger were pulled.	2-13
SNAPI_ReleaseTrigger	Sends a command to the scanner to release the software trigger.	2-14
SNAPI_ScanDisable	Sends a command to the scanner to disable the scanner.	2-17
SNAPI_ScanEnable	Sends a command to the scanner to enable the scanner.	2-18
SNAPI_SetParameterDefaults	Sends a command to the scanner to set all the parameters to default values.	2-23
SNAPI_SetParameters	Sends a command to the scanner to change one or more of the scanner's parameter values.	2-24
SNAPI_SetParamPersistance	Sets the persistence quality for any parameter changes requested.	2-25
SNAPI_SnapShot	If the scanner supports imaging, this sends a command to the scanner to enter Image Capture Mode.	2-28
SNAPI_SoundBeeper	Sends a command to the scanner to turn the beeper on.	2-29
SNAPI_TransmitVideo	Sends a command to the imager to enter Video Mode.	2-31
Lifecycle Commands		
SNAPI_Connect	Connects to the scanner using the indicated device handle.	2-6
SNAPI_Disconnect	Terminates the communications link and releases all memory used by the library for the scanner being disconnected.	2-7
SNAPI_Init	Initializing command that registers the window that will receive Windows messages from the library. It returns an array of the Symbol Image scanners that are connected to the host through USB.	2-10
Metadata Command		
SNAPI_ReturnDLLVersion	Returns the major and minor version levels of the DLL.	2-17
Storage Commands		
SNAPI_SetCapabilitiesBuffer	Allows the application to specify the address and the length of a buffer for the DLL to use to store the capabilities data from the scanner.	2-19
SNAPI_SetDecodeBuffer	Sets the decode data buffer and its length for the DLL to use to store decode data from the scanner.	2-20
SNAPI_SetImageBuffer	Sets the image data buffer and its length for the DLL to use to store image data from the scanner.	2-21

Table 2-1 *SNAPI Functions by Type (Continued)*

Command	Brief Description	Page
SNAPI_SetParameterBuffer	Sets the user's destination buffer and its length for the DLL to use to store parameter data from the scanner.	*2-22*
SNAPI_SetVersionBuffer	Sets the user's destination buffer and its length for the DLL to use to store version data from the scanner.	*2-26*
SNAPI_SetVideoBuffer	Sets the user's destination buffer and its length for the DLL to use to store video data from the scanner.	*2-27*

SNAPI Function Descriptions

SNAPI_AbortMacroPdf

Host to Scanner Command

Sends a command to the scanner to abort the current MacroPDF scanning session and discard any stored MacroPDF data.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_AbortMacroPdf(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_AbortMacroPdf(devicehandle[0]);
```

SNAPI_AimOff

Host to Scanner Command

Sends the Aim Off command to the scanner to turn off the aiming pattern (SNAPI_AimOn turns it on).

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_AimOff(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_AimOff(devicehandle[0]);
```

SNAPI_AimOn

Host to Scanner Command

Sends the Aim On command to the scanner to turn on the aiming pattern of an imager (SNAPI_AimOff turns it off).

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_AimOn(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_AimOn(devicehandle[0]);
```

SNAPI_Connect

Lifecycle Command

SNAPI_Connect establishes a SNAPI communication connection with the scanner indicated by the device handle. SNAPI_Connect can be called anytime after the SNAPI_Init call, but it must be called before using any other APIs.

No command is sent to the scanner during this API call. Unless a call to disconnect the scanner is issued, the scanner may send decode data at any time after it is connected.

The host application should call SNAPI_SetDecodeBuffer after a successful call to SNAPI_Connect if it wants to handle unsolicited data from the scanner.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_Connect(
    HANDLE DeviceHandle);
```

Parameters

DeviceHandle

The handle of the device to connect. Handles to existing devices are obtained from the output parameter of the SNAPI_Init function, or as a parameter to the WM_DEVICENOTIFICATION message when a new SNAPI compatible device is connected after the SNAPI SDK initialization.

Return Values

- SSICOMM_NOERROR – Returned if device is connected successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
HANDLE devicehandle[MAX_SCANNER]={0};
int NumScanners=0;
int status;
status = SNAPI_Init(m_hWnd, devicehandle, & NumScanners);
if ((status == SSI COM_NOERROR) && NumScanners)
{
// Assuming one scanner found during call to SNAPI_Init
status=SNAPI_Connect(devicehandle[0]);
}
```

SNAPI_Disconnect

Lifecycle Command

Terminates the communications link and releases all memory used by the library for the scanner being disconnected. No command is sent to the scanner during this API call.

Always call this function when the application is finished communicating with the decoder.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_Disconnect(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_Disconnect(devicehandle[0]);
```

SNAPI_EnterLowPwrMode

Host to Scanner Command

Sends a command to the scanner to enter low power mode.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_EnterLowPwrMode(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_EnterLowPwrMode(devicehandle[0]);
```

SNAPI_FlushMacroPdf

Host to Scanner Command

Sends a command to the scanner to abort the current MacroPDF scanning session and transmit stored MacroPDF data. This call must be preceded by a call to SNAPI_SetDecodeBuffer to provide a buffer for the data that will be returned from the scanner.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_FlushMacroPdf(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_FlushMacroPdf(devicehandle[0]);
```

SNAPI_GetSerialNumber

Data Request Command

This command returns the serial number of the connected scanner. No command is sent to scanner.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_GetSerialNumber(
    HANDLE DeviceHandle,
    unsigned char *SerialNo);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

SerialNo

Pointer to a buffer to receive serial number of connected scanner.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define SERIAL_NUM_LEN 255
// globally defined data storage
unsigned char SerialNo[SERIAL_NUM_LEN];
int status;
status=SNAPI_GetSerialNumber (devicehandle[0], SerialNo);
```

SNAPI_Init

Lifecycle Command

Initializing command that registers the window that will receive Windows messages from the library. It returns an array of the Symbol Image scanners that are connected to the host through USB. No command is sent to the scanner during this API call.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_Init(
    HWND hwnd,
    HANDLE *DeviceHandles,
    int *NumDevices);
```

Parameters

hwnd

The handle of the window whose procedure will receive Windows messages from the library.

DeviceHandles

A pointer to an array that will store handles for the devices that are found.

NumDevices

Pointer to the number of devices that were found.

Return Values

- SSICOMM_NOERROR – The device connected successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
HANDLE devicehandle[MAX_SCANNER]={0};
int NumScanners=0;
int status;
status = SNAPI_Init(m_hWnd, devicehandle, & NumScanners);
```

SNAPI_LEDOff

Host to Scanner Command

Sends a command to the scanner to turn off the LED on the scanner (SNAPI_LedOn turns it on). Which LED to turn off is specified as a bit in the nLEDselection parameter.

Syntax

SNAPIDLL_API int __stdcall SNAPI_LedOff(HANDLE DeviceHandle,unsigned char nLEDselection);

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

nLEDselection

The bitwise indicator for the LED to be turned off (scanner dependent).

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
unsigned char decode_led = 0x02; // assumes led is represented using
// bit 1
status=SNAPI_LedOff(devicehandle[0],decode_led);
```

SNAPI_LEDOn

Host to Scanner Command

Sends a command to the scanner to turn on the LED of the scanner (SNAPI_LedOff turns it off). Which LED to turn on is specified as a bit in the nLEDselection parameter.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_LedOn(
    HANDLE DeviceHandle,
    unsigned char nLEDselection);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

nLEDselection

The bitwise indicator for the LED to be turned on (scanner dependent).

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
unsigned char decode_led = 0x02; // assumes led is represented using bit 1
status=SNAPI_LedOn(devicehandle[0],decode_led);
```

SNAPI_PullTrigger

Host to Scanner Command

Sends a command to the scanner to perform a software trigger, causing the scanner to behave as if the trigger were pulled.

Some scanners require setting the trigger mode to host mode. To do this, call SNAPI_SetParameters before the SNAPI_PullTrigger function. See the documentation for your scanning device.

- If the scanner is in Decode Mode, the laser (or camera) turns off after a successful decode or a call to SNAPI_ReleaseTrigger.
- If the scanner is in Image Capture Mode and a call to Snapshot was made prior to the SNAPI_PullTrigger call, the camera captures an image and turns off (SNAPI_ReleaseTrigger call is not necessary).
- If the scanner is in Video Mode, the camera turns off after a call to SNAPI_ReleaseTrigger.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_PullTrigger(
   HANDLE DeviceHandle)
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_PullTrigger(devicehandle[0]);
```

SNAPI_ReleaseTrigger

Host to Scanner Command

Sends a command to the scanner to release the software trigger. Call SNAPI_ReleaseTrigger to abort a decode attempt or video transmission.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_ReleaseTrigger(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_ReleaseTrigger(devicehandle[0]);
```

SNAPI_RequestParameters

Data Request Command

Requests the scanner to send parameter values specified in a given parameter string. This call must be preceded by a call to SNAPI_SetParameterBuffer to provide a buffer for the data that will be returned from the scanner. Seven parameter values may be requested in one call.

After the DLL receives the parameter data from the scanner, it sends the host application a Windows message indicating that the parameter data is available.

NOTE Valid parameter numbers applicable to a scanner can be found in the scanner *Product Reference Guide* or *Integration Guide*. Parameter numbers appear in SSI format and must be converted into the actual parameter number. See the description for *WM_PARAM on page 1-6* for conversion instructions.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_RequestParameters(
   WORD *Params,
   int nParamWords,
   HANDLE DeviceHandle);
```

SNAPI_RequestParameters (continued)

Parameters

Params

Buffer of WORD values which specifies the parameter numbers whose values are being requested from the scanner. Refer to the documentation for the scanning device for parameter numbers.

nParamWords

The number of WORDs in the Params buffer.

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define SWTRIG_PARAMNUM 0x008a
// software trigger parameter number
#define IMAGE_FILETYPE_PARAMNUM 0x0130
#define PARAM_RETURN_DATA_LEN 2000
//globally defined data storage
unsigned char ParamReturnData[PARAM_RETURN_DATA_LEN];int status;
WORD Params[2];
Params[0] = SWTRIG_PARAMNUM;
// get the software trigger setting
Params[1] = IMAGE_FILETYPE_PARAMNUM
// image filetype setting
// Give the dll a buffer to use when the scanner returns the parameter
// data using com 1
SNAPI_SetParameterBuffer(devicehandle[0], ParamReturnData,
PARAM_RETURN_DATA_LEN);
// Send the request which is stored in Params and is 2 words long
status = SNAPI_RequestParameters(Params, 2, devicehandle[0]);
// If status is good, the request was sent. Later, when the DLL receives the data back
// from the scanner, the data will be stored in the ParamReturnData buffer and the
// application will receive a windows message. The parameter number and value
// will be a WORD data. That means to return a parameter, we'll need two
// bytes for parameter number and two bytes for parameter value.
//One should allocate memory accordingly.
```

SNAPI_RequestScannerCapabilities

Data Request Command

Sends a request to the scanner to send its capabilities data, i.e., the commands it can perform. This call must be preceded by a call to SNAPI_SetCapabilitiesBuffer.

When the scanner responds to this command with its capabilities data, the DLL sends a Windows message to the host application indicating that capabilities data is stored.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_RequestScannerCapabilities (
    HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define CAPABILITIES_RETURN_DATA_LEN 255
// globally defined data storage
unsigned char CapabilitesReturnData[CAPABILITIES_RETURN_DATA_LEN];
int status;
// Give the dll a buffer to use when the scanner returns the data
SNAPI_SetCapabilitiesBuffer(devicehandle[0],
CapabilitesReturnData,CAPABILITIES_RETURN_DATA_LEN);
status =SNAPI_RequestScannerCapabilities(devicehandle[0]);
// If status is good, the request was sent. Later, when the DLL receives
// the data back from the scanner, the data will be stored in the
// CapabilitiesReturnData buffer and the application will receive a
// windows message.
```

SNAPI_ReturnDLLVersion

Metadata Command

Returns the major and minor version levels of the DLL. No command is sent to the scanner during this API call.

Syntax

SNAPIDLL_API int __stdcall SNAPI_ReturnDLLVersion();

Parameters

No parameters are passed.

Return Values

- The minor version level is returned in the lower byte, and the major version level is returned in the next higher order byte.

Example

```
unsigned int version, major, minor;
CString msg;
version = SNAPI_ReturnDLLVersion();
major = (version & 0x0000ff00) >> 8;
minor = version & 0x000000ff;
msg.Format("Library Version %d.%d", major, minor);
```

SNAPI_ScanDisable

Host to Scanner Command

Sends a command to the scanner to disable the scanner (SNAPI_ScanEnable enables scanning). When scanning is disabled, the scanner does not respond to a physical or software trigger pull.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_ScanDisable(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_ScanDisable(devicehandle[0]);
```

SNAPI_ScanEnable

Host to Scanner Command

Sends a command to the scanner to enable the scanner (SNAPI_ScanDisable disables scanning).

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_ScanEnable(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_ScanEnable(devicehandle[0]);
```

SNAPI_SetCapabilitiesBuffer

Storage Command

Allows the application to specify the address and the length of a buffer for the DLL to use to store the capabilities data from the scanner. Set the capabilities data buffer immediately before a call to SNAPI_RequestScannerCapabilities.

No command is sent to the scanner during this API call. The amount of capabilities data is variable; a buffer of length 256 should be sufficient.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetCapabilitiesBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,
    long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for capabilities data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define CAPABILITIES_DATA_LEN 255
// globally defined data storage
unsigned char CapabilitiesReturnData[CAPABILITIES_DATA_LEN];
int status;
status = SNAPI_SetCapabilitiesBuffer(devicehandel[0],
CapabilitiesReturnData, MAX_DATA_LEN);
```

SNAPI_SetDecodeBuffer

Storage Command

Sets the decode data buffer and its length for the DLL to use to store decode data from the scanner. No command is sent to the scanner during this API call. The length of decode data depends on the type of bar code scanned; if MacroPDF is buffered, large amounts of data are possible.

When the DLL has decode data from the scanner, this buffer is filled and a Windows message is sent to the application to indicate that decode data is stored. The host application must then call SNAPI_SetDecodeBuffer again in order to receive additional decode data from the scanner.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetDecodeBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,
    long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for decode data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

NOTE Large amounts of data, such as MacroPDF and Signature Capture require a larger buffer.

```
#define DECODE_DATA_LEN 255
// globally defined data storage
unsigned char DecodeData[DECODE_DATA_LEN];
int status;
status = SNAPI_SetDecodeBuffer(devicehandle[0], DecodeData,
DECODE_DATA_LEN);
```

SNAPI_SetImageBuffer

Storage Command

Sets the image data buffer and its length for the DLL to use to store image data from the scanner. No command is sent to the scanner during this API call. By handling the image transfer status Windows messages sent by the DLL which specify total size of the image in bytes, the host application can create a buffer of the necessary size once an image transfer is initiated.

When the DLL has the entire image data from the scanner, the destination buffer is filled and a Windows message is sent to the application indicating that the image data is available.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetImageBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,
    long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for image data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define IMAGE_DATA_LEN 640 *480
// globally defined data storage
unsigned char ImageData[IMAGE_DATA_LEN];
int status;
status = SNAPI_SetImageBuffer(devicehandle[0], ImageData, IMAGE_DATA_LEN);
```

SNAPI_SetParameterBuffer

Storage Command

Sets the user's destination buffer and its length for the DLL to use to store parameter data from the scanner. No command is sent to the scanner during this API call.

Set the parameter data buffer immediately before calling SNAPI_RequestParameters. A call for a single parameter only requires a small buffer: 10 bytes is sufficient.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetParameterBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for parameter data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define PARAM_RETURN_DATA_LEN 2000
// globally defined data storage
unsigned char ParamReturnData[PARAM_RETURN_DATA_LEN];
int status;
status = SNAPI_SetParamterBuffer(devicehandle[0], ParamReturnData,
PARAM_RETURN_DATA_LEN);
```

SNAPI_SetParameterDefaults

Host to Scanner Command

Sends a command to the scanner to set all the parameters to default values.

NOTE This function results in permanent parameter changes which involves a write to non-volatile memory (NVM). There is a large but finite number of write cycles available to the NVM. Be careful not to use this function often.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_ SetParameterDefaults (
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_SetParameterDefaults (devicehandle[0]);
```

SNAPI_SetParameters

Host to Scanner Command

Sends a command to the scanner to change one or more of the scanner's parameter values. The format of the parameter number and its associated value is: <param_num><value> param_num and value are word values. Seven parameter values may be changed in one call.

NOTE Valid parameter numbers applicable to a scanner can be found in the scanner *Product Reference Guide* or *Integration Guide*. Parameter numbers appear in SSI format and must be converted into the actual parameter number. See the description for *WM_PARAM on page 1-6* for conversion instructions.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetParameters(
    WORD *Params,
    int ParamWords,
    HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Params

A pointer to the param_num/value data.

ParamWords

The size in words of the data stored in Params.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define IMAGE_FILETYPE_PARAMNUM 0x0130
#define BITMAP_TYPE 0x0003
int status;
unsigned WORD Params[2];
Params[0] = IMAGE_FILETYPE_PARAMNUM; // this is the parameter number
// image file type
Params[1] = BITMAP_TYPE; // this is the parameter value
// 2 WORDS were stored and we are
// using devicehandle[0]
status = SNAPI_SetParamters(Params, 2, devicehandle[0]);
```

SNAPI_SetParamPersistance

Host to Scanner Command

Sets the persistence quality for any parameter changes requested. Parameters may be changed permanently or temporarily. No command is sent to the scanner during this API call; during any subsequent call to SNAPI_SetParameters, the persistence quality given here is used. By default, parameter changes are temporary.

NOTE Permanent parameter changes involve writes to non-volatile memory (NVM). There is a large but finite number of write cycles available to the NVM. Be careful not to use the permanent parameter change method on parameters that change often.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetParamPersistance(
    HANDLE DeviceHandle,
    int bPersist);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

bPersist

Set to TRUE if persistence is desired, FALSE if not.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status = SNAPI_SetParamPersistance(devicehandle[0],FALSE);
```

SNAPI_SetVersionBuffer

Storage Command

Sets the user's destination buffer and its length for the DLL to use to store version data from the scanner. No command is sent to the scanner during this API call.

Set the version data buffer immediately before calling SNAPI_TransmitVersion. The amount of data returned is variable; a buffer of length 256 should be sufficient.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetVersionBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,
    long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for version data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define VERSION_DATA_LEN 255
// globally defined data storage
unsigned char VersionData[DECODE_DATA_LEN];
int status;
status = SNAPI_SetVersionBuffer(devicehandle[0], VersionData,
VERSION_DATA_LEN);
```

SNAPI_SetVideoBuffer

Storage Command

Sets the user's destination buffer and its length for the DLL to use to store video data from the scanner. No command is sent to the scanner during this API call.

Video data is sent continuously when the scanner is in video mode, so when the application receives the Windows message notifying it that Video data has been stored, the host program should process the stored video frame then call SNAPI_SetVideoBuffer again to set the destination for the next frame.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SetVersionBuffer(
    HANDLE DeviceHandle,
    unsigned char *pData,
    long max_length);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

pData

A pointer to the destination buffer for video data returned from the scanner.

max_length

The size in bytes of the destination buffer.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define VIDEO_DATA_LEN 5000
// globally defined data storage
unsigned char VideoData[VIDEO_DATA_LEN];
int status;
status = SNAPI_SetVideoBuffer(devicehandle[0], VideoData, VIDEO_DATA_LEN);
```

SNAPI_SnapShot

Host to Scanner Command

If the scanner supports imaging, this sends a command to the scanner to enter Image Capture Mode. The scanner remains in Image Capture Mode until the trigger is pulled (physically or with a call to SNAPI_PullTrigger) and an image is captured, or until the timeout for a trigger pull expires. The scanner then returns to Decode Mode.

If the trigger is pulled, the image data is sent to the DLL. WM_XFERSTATUS is sent to the host application with information about the size of the image.

When the first transfer status message is received, the host application should provide a destination buffer for the image by calling SNAPI_SetImageBuffer. After the entire image is transferred from the scanner to the DLL, the application receives a Windows message indicating that the image data was stored.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SnapShot(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_SnapShot(devicehandle[0]);
```

SNAPI_SoundBeeper

Host to Scanner Command

Sends a command to the scanner to turn the beeper on. (See *Table 1-5 on page 1-10* for beep codes.)

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_SoundBeeper(
    HANDLE DeviceHandle,
    unsigned char nBeepCode);
```

Parameters

nBeepCode

Specifies the tone and duration for the beep.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
unsigned char nBeepCode = TWOSHORTHI;
status=SNAPI_SoundBeeper(devicehandle[0], nBeepCode);
```

SNAPI_TransmitVersion

Data Request Command

Sends a request to the scanner to send its software release name. Call SNAPI_SetVersionBuffer before this call to provide a destination buffer for the version data when it is sent by the scanner.

After the scanner responds with the version data, the DLL sends the host application a Windows message indicating that the version data is available.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_TransmitVersion(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
#define VERSION_DATA_LEN 255
// globally defined data storage
unsigned char VersionData[VERSION_DATA_LEN];
int status;
status = SetVersionBuffer(devicehandle[0], VersionData, VERSION_DATA_LEN);
status=SNAPI_TransmitVersion(devicehandle[0]);
```

SNAPI_TransmitVideo

Host to Scanner Command

Sends a command to the imager to enter Video Mode. After the trigger is pulled (physically or with a call to SNAPI_PullTrigger), the decoder produces a continuous video stream until the trigger is released (physically or with a call to SNAPI_ReleaseTrigger). The destination buffer for each video frame must be set with a call to SNAPI_SetVideoBuffer.

Syntax

```
SNAPIDLL_API int __stdcall SNAPI_TransmitVideo(
   HANDLE DeviceHandle);
```

Parameters

DeviceHandle

Handle of the device used in the call to SNAPI_Connect.

Return Values

- SSICOMM_NOERROR – Returned if the command is sent successfully.
- An error code if an error occurred. (See *Table 1-2 on page 1-8*.)

Example

```
int status;
status=SNAPI_TransmitVideo(devicehandle[0]);
```

Appendix A

Bar Code Types & Code Examples

Bar Code Types, Code Types

Table A-1 *Codes and Bar Code Types*

SSI ID	Bar Code Type
1	Code 39
2	Codabar
3	Code 128
4	Discrete (Standard) 2 of 5
5	IATA
6	Interleaved 2 of 5
7	Code 93
8	UPC-A
9	UPC-E0
10	EAN-8
11	EAN-13
12	Code 11
13	Code 49
14	MSI
15	EAN-128
16	UPC-E1
17	PDF-417
18	Code 16K
19	Code 39 Full ASCII
20	UPC-D
21	Code 39 Trioptic

Bar Code Types, Code Types

Table A-1 *Codes and Bar Code Types (Continued)*

SSI ID	Bar Code Type
22	Bookland
23	Coupon Code
24	NW-7
25	ISBT-128
26	Micro PDF
27	DataMatrix
28	QR Code
29	Micro PDF CCA
30	PostNet US
31	Planet Code
32	Code 32
33	ISBT-128 Con
34	Japan Postal
35	Australian Postal
36	Dutch Postal
37	MaxiCode
38	Canadian Postal
39	UK Postal
40	Macro PDF
48	RSS-14
49	RSS Limited
50	RSS Expanded
55	Scanlet
72	UPC-A + 2 Supplemental
73	UPC-E0 + 2 Supplemental
74	EAN-8 + 2 Supplemental
75	EAN-13 + 2 Supplemental
80	UPC-E1 + 2 Supplemental

Bar Code Types, Code Types

Table A-1 *Codes and Bar Code Types (Continued)*

SSI ID	Bar Code Type
81	CCA EAN-128
82	CCA EAN-13
83	CCA EAN-8
84	CCA RSS Expanded
85	CCA RSS Limited
86	CCA RSS-14
87	CCA UPC-A
88	CCA UPC-E
89	CCC EAN-128
90	TLC-39
97	CCB EAN-128
98	CCB EAN-13
99	CCB EAN-8
100	CCB RSS Expanded
101	CCB RSS Limited
102	CCB RSS-14
103	CCB UPC-A
104	CCB UPC-E
105	Signature Capture
113	Matrix 2 of 5
114	Chinese 2 of 5
136	UPC-A + 5 supplemental
137	UPC-E0 + 5 supplemental
138	EAN-8 + 5 supplemental
139	EAN-13 + 5 supplemental
144	UPC-E1 + 5 Supplemental
154	Macro Micro PDF

Code Examples

WM_DECODE

Example

Registering for the decode data event and obtaining bar code data.

```
// Define a message handler member function inside the class declaration
afx_msg LRESULT OnBarcodeData(WPARAM wParam, LPARAM lParam);

// Include message handler for WM_DECODE message in message map
BEGIN_MESSAGE_MAP(CMyWnd, CMyParentWndClass)
    ON_MESSAGE(WM_DECODE, OnBarcodeData)
END_MESSAGE_MAP()
// Register buffer with SDK for receiving decode data
// (See description for SNAPI_SetDecodeBuffer API on page 2-20)
// Once bar code is scanner, the bar code message handler will be invoked
// Bar code data message handler
afx_msg LRESULT CMyWnd::OnBarcodeData(WPARAM w, LPARAM l)
{

// Low order word in wparam contains the status bits for the data,
// High order word in wparam contains length of data obtained

// lparam is the handle of the device from which decode happened
unsigned char LabelData[DECODE_DATA_LEN] = {0};
LabelType = (WORD)DecodeData; // label type is a word value stored as Little Endian
DWPARAM *wParam =(DWPARAM *) w;
DWPARAM *lParam =(DWPARAM *) l;
HANDLE deviceHandle = (HANDLE)lParam;

// Check status and copy the contents of dll's decode data buffer to our own.
if( LODWORD(wParam) & BUFFERSIZE_MASK == BUFFERSIZE_GOOD )
{
   DataLen = HIDWORD(wParam);
  // Assuming DecodeData is the global buffer registered
  // for receiving decoded data
  // First word in the buffer represent bar code type

  LabelType = DecodeData[0];

  // Copy the bar code data
  memcpy(LabelData, DecodeData+2, DataLen-2);
  // Do whatever processing required with LabelType and LabelData
}
else if( LODWORD(wParam) & BUFFERSIZE_MASK == BUFFERSIZE_ERROR )
{
  // Error: Size of data scanned is larger than
  // the size of the registered decode buffer
}
else
{
  // Error: No decode buffer was registered!
}
// Register back the decode data buffer for future bar code data
SNAPI_SetDecodeBuffer( devicehandle, DecodeData, DECODE_DATA_LEN );
return 0;
}
```

WM_PARAM

Example

Registering for the parameter data event and interpreting incoming parameter data.

```
// Define a message handler member function inside the class declaration
afx_msg LRESULT OnParameterData(WPARAM wParam, LPARAM lParam);

// Include message handler for WM_PARAM message in message map
BEGIN_MESSAGE_MAP(CMyWnd, CMyParentWndClass)
    ON_MESSAGE(WM_PARAM, OnParameterData)
END_MESSAGE_MAP()
// Register buffer with SDK for receiving parameter data
// (See description for SNAPI_SetParameterBuffer API on page 2-22)
// Request parameter data
// (See description for SNAPI_RequestParameters API on page 2-14)

// Parameter data message handler
afx_msg LRESULT CMyWnd::OnParameterData(WPARAM w, LPARAM l)
{
// Low order word in wparam contains the status bits for the data,
// High order word in wparam contains length of data as number of words
// lparam is the handle of the device from which the parameter is issued
DWPARAM *wParam =(DWPARAM *) w;
DWPARAM *lParam =(DWPARAM *) l;
HANDLE deviceHandle = (HANDLE)lParam;
int NumOfParams;
// Check status and extract parameter information
if( LODWORD(wParam) & BUFFERSIZE_MASK == BUFFERSIZE_GOOD )
{
  NumOfParams = HIDWORD(wParam)/ 2;
  // Assuming ParamReturnData is the global buffer registered with
  // SDK using SNAPI_SetParameterBuffer API
  WORD *ParamBuf = (WORD *)ParamReturnData;
  for(int count=0' cpimt <NumOfParams; count++
  {
     WORD NextParNum = ParamBuf[0];
     WORD NextParVal = ParamBuf[1];
     // Do whatever operations required with Parameter number and data ...
     // Display (NextParNum ,NextParVal);
     ParamBuf += 2;

  }
}
else if( LODWORD(wParam) & BUFFERSIZE_MASK == BUFFERSIZE_ERROR )
{
  // Error: Size of data scanned is larger than
  // the size of the registered decode buffer
}
else
{
  // Error: No parameter buffer was registered!
}
// Register back the parameter data buffer for future parameter messages
SNAPI_SetParameterBuffer( devicehandle, ParamReturnData
                          sizeof(ParamReturnData));
return 0;
}
```

Index

G

H

I

L

M

N

S

U

V

W

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

72E-71370-01
Revision A - March 2006