Simple Serial Interface

Developer's Guide

Simple Serial Interface (SSI) Developer's Guide

72E-50705-01
Revision C
June 2008

Motorola, Inc.
One Motorola Plaza
Holtsville, New York 11742-1300
http://www.motorola.com/enterprise

Revision History

Changes to the original manual are listed below:

Change	Date	Description
-01 Rev. A	1/2001	Initial Release.
-01 Rev. B	4/2001	Engineering updates.
-01 Rev. C	6/2008	Motorola re-branding.

Table of Contents

Chapter 6: OS API

Chapter 7: Port API

About This Guide

Introduction

The *Simple Serial Interface (SSI) Developer's Guide* provides system requirements and programming information about Symbol Technologies' Simple Serial Interface, which enables Symbol's decoders (e.g., SE 2223 scan engine, hand-held scanners, 2D scanners, etc.) to communicate with a serial host. It provides examples of how to implement the host side of the SSI protocol.

Chapter Descriptions

Topics covered in this guide are as follows:

- *Chapter 1, Using the SSI SDK* describes how to build the SDK for both DOS and Windows, and how to use the demo application to set SSI commands.
- *Chapter 2, Introduction to SSI* provides an overview of SSI, including signal lines, protocol, and packeting information. Protocol layers are described in a "bottom-up" manner, from the hardware layer up through software handshaking.
- *Chapter 3, Decoder API* describes the Decoder API commands which enable communication between the host and the decoder.
- *Chapter 4, Communications API* detail the Communications API commands, which enable communication between the serial port and the SSI Kernel code.
- *Chapter 5, Kernel API* describes the Kernel API commands, which maintain the SSI between the host and SSI-compliant decoder.
- *Chapter 6, OS API*describes the Operating System functions, used for the target operating system and the platform-independent SSI Kernel code.
- *Chapter 7, Port API*describes the Port API functions, the hardware-specific software implemented for the target operating system and the platform-independent SSI Kernel code.
- *Appendix A, SSI Commands* describes each command supported by SSI.

Notational Conventions

The following conventions are used in this document:

- "User" refers to anyone using an SSI product.
- "You" refers to the End User, System Administrator or Programmer using this manual as a reference for SSI.
- *Italics* are used to highlight the following:
  - Chapters and sections in this and related documents
  - Dialog box, window and screen names
  - Drop-down list and list box names
  - Check box and radio button names
  - Icons on a screen.
- **Bold** text is used to highlight the following:
  - Key names on a keypad
  - Button names on a screen or window.
- 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.

NOTE This symbol indicates something of special interest or importance to the reader. Failure to read the note will not result in physical harm to the reader, equipment or data.

CAUTION This symbol indicates that if this information is ignored, the possibility of data or material damage may occur.

WARNING! **This symbol indicates that if this information is ignored the possibility that serious personal injury may occur.**

Related Documents

Refer to the *Product Reference Guide* for your product for product-specific information on SSI.

For the latest version of this guide and all guides, go to: www.motorola.com/enterprise/manuals.

Service Information

If you have a problem with your equipment, contact Motorola Enterprise Mobility support for your region. Contact information is available at: http://www.motorola.com/enterprise/contactsupport. When contacting Enterprise Mobility support, please have the following information available:

- Serial number of the unit
- Model number or product name
- Software type and version number

Motorola responds to calls by e-mail, telephone or fax within the time limits set forth in service agreements. If your problem cannot be solved by Motorola Enterprise Mobility Support, you may need to return your equipment for servicing and will be given specific directions. Motorola 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 you purchased your Enterprise Mobility business product from a Motorola business partner, please contact that business partner for support.

Chapter 1 Using the SSI SDK

Introduction

This chapter describes how to build and use Symbol's SSI SDK DOS Demo Application and the SSI SDK 32-bit Windows version.

Building the SSI SDK DOS Demo Application (MSDOS 6.22 version)

1. Using the build tool MSVC++ version 1.52, open the project ssidos16.mak from the **\lib** directory on your SSI SDK CD.

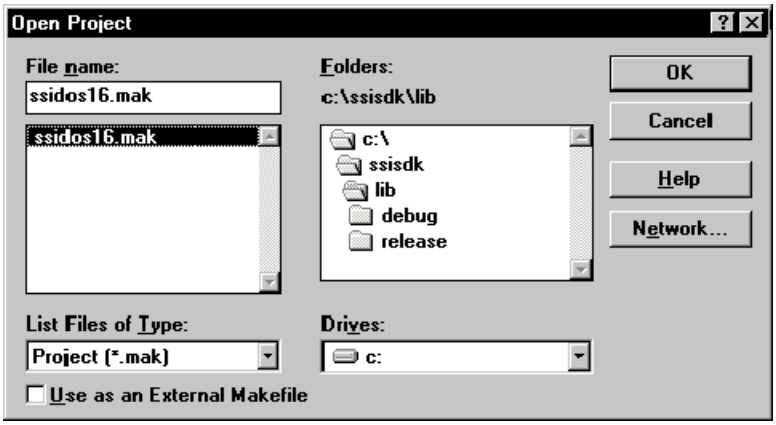

Figure 1-1 *Opening ssidos16.mak Project*

2. Select Project -> Rebuild All SSIDOS16.LIB to create the **ssidos16.lib** file. The following screen displays.

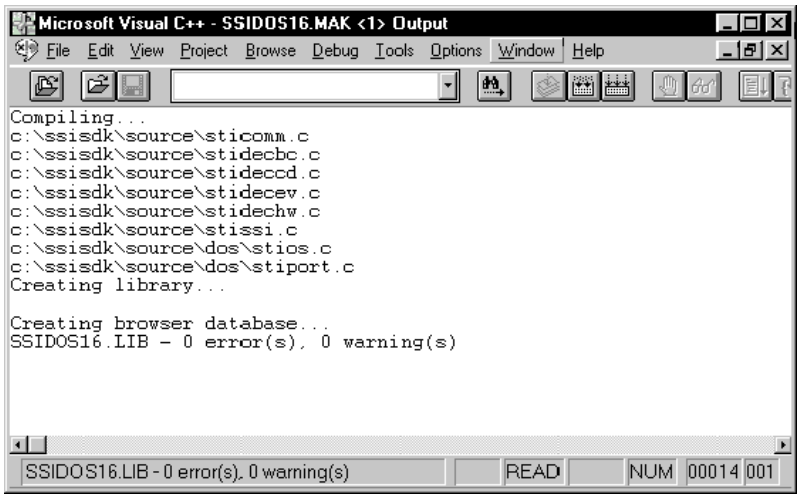

Figure 1-2 *Building ssidos16.lib Project*

3. Open the project **Dosdemo.mak** from the **\dos16test** directory.

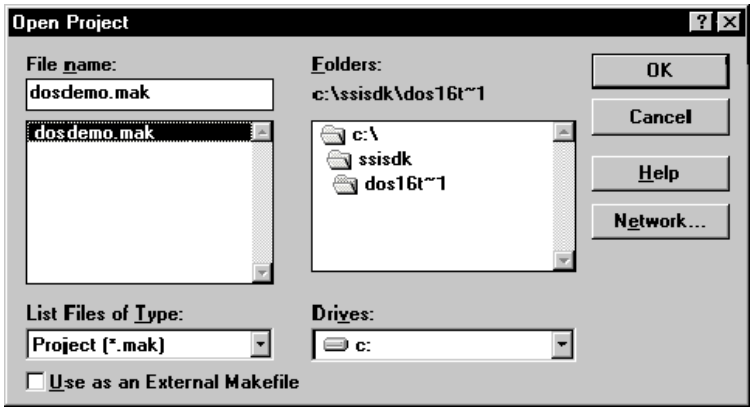

Figure 1-3 *Opening Dosdemo.mak*

4. Selecting Project -> Rebuild All to create **DOSDEMO.EXE**.
5. Connect your device to COM1.
6. Execute DOSDEMO.EXE.

Using the SSI SDK DOS Demo Application

Executing DOSDEMO.EXE opens the SSI SDK DOS Demo application window:

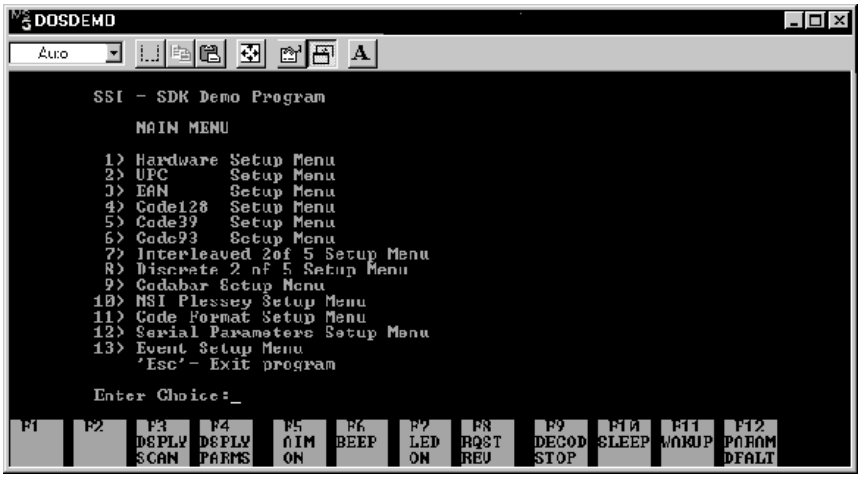

Figure 1-4 *SSI SDI DOS Demo Application*

The Function keys in the DOS Demo Program activate the following functions:

- F3 DSPLY SCAN: Displays scanned data.
- F4 DSPLY PARMS: Displays parameter data.
- F5 AIM ON: Activates the aim pattern.
- F6 BEEP: Sounds the beeper.
- F7 LED ON: Activates the Decode LED output.
- F8 RQST REV: Replies with the decoder's software configuration revision.
- F9 DECOD STOP: Tells decoder to abort a decode event.
- F10 SLEEP: Requests to place the decoder into low power.
- F11 WAKUP: Wakes the decoder after it's been powered down.
- F12 PARAM DFALT: Sets parameter default values.

The DOS Demo Program main menu displays the following Setup Menus: Hardware, UPC, EAN, Code128, Code39, Code93, Interleaved 2 of 5, Discrete 2 of 5, Codabar, MSI Plessey, Code Format, Serial Parameters, and Event Setup.

Select the number preceding each option to set up that option. Following are the screens that display. On each setup screen, enter the number preceding the option you'd like to change, then select the desired value.

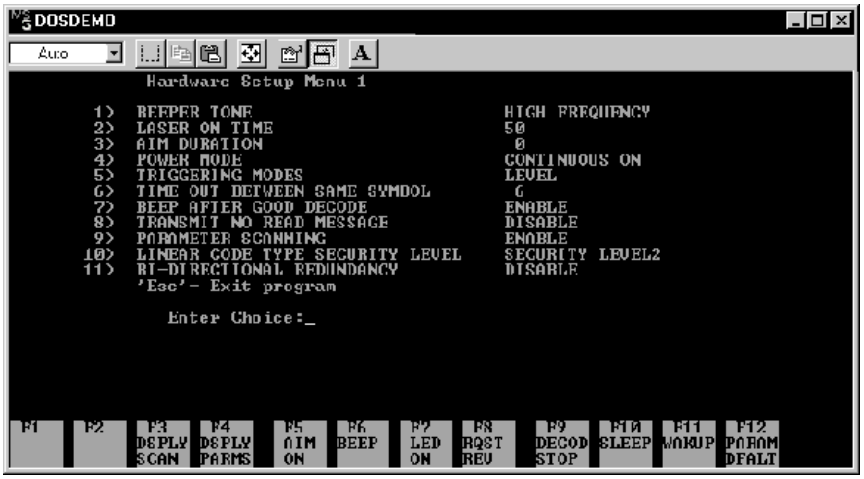

Figure 1-5 *Hardware Setup Menu*

Figure 1-6 *UPC Setup Menu*

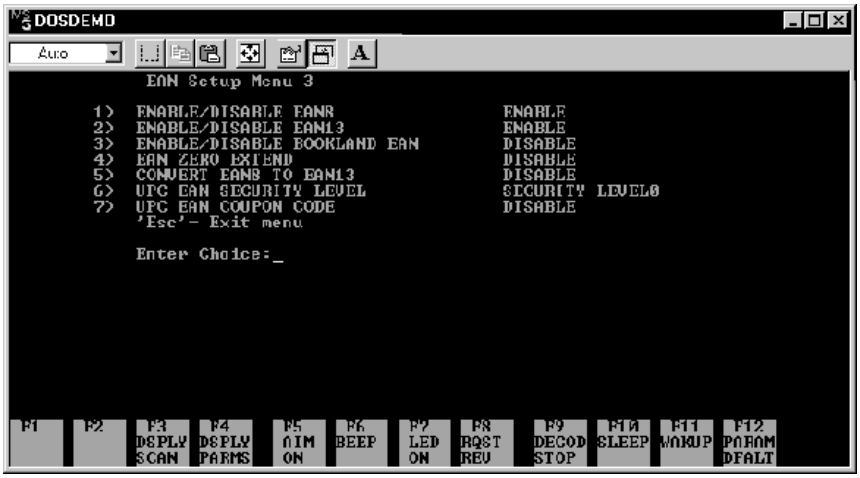

Figure 1-7 *EAN Setup Menu*

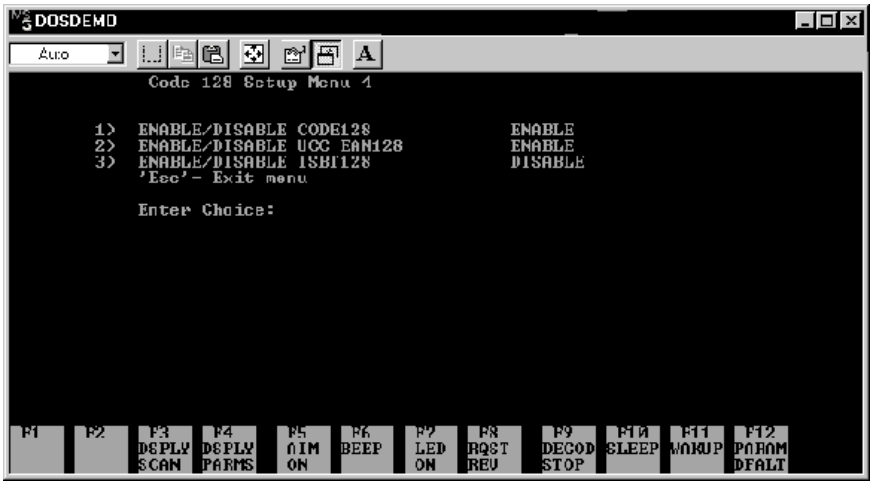

Figure 1-8 *Code 128 Setup Menu*

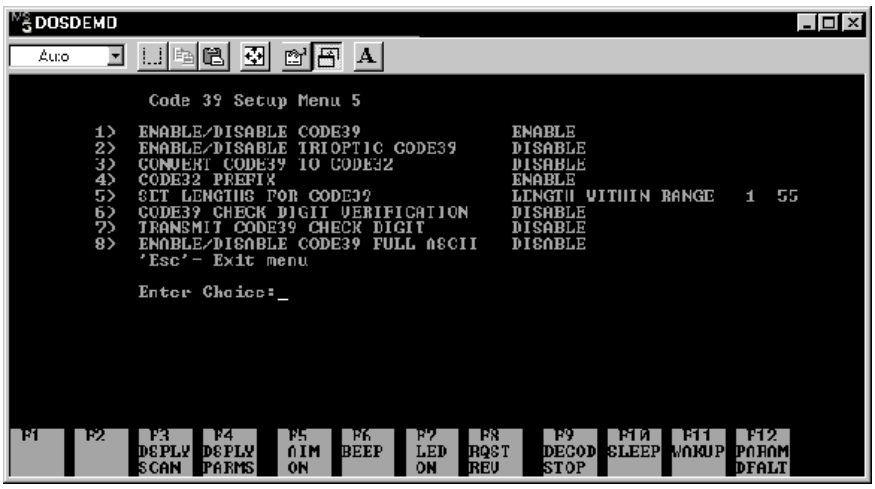

Figure 1-9 *Code 39 Setup Menu*

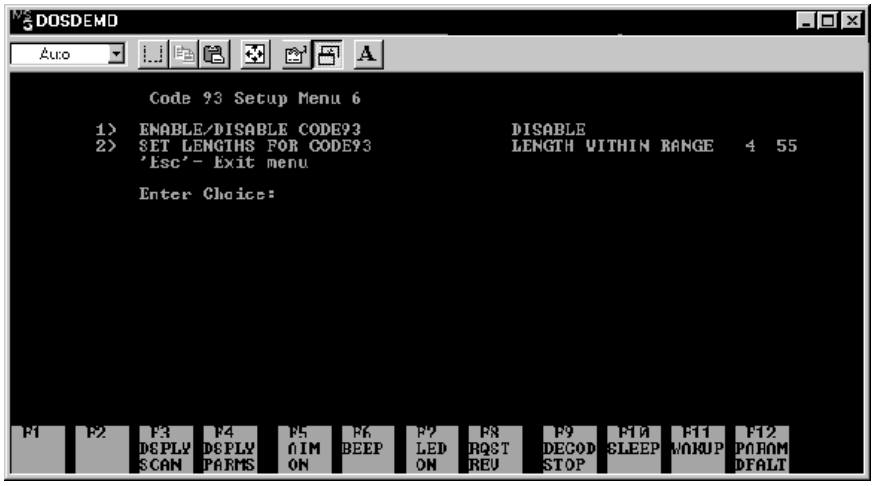

Figure 1-10 *Code 93 Setup Menu*

Figure 1-11 *Interleaved 2 of 5 Setup Menu*

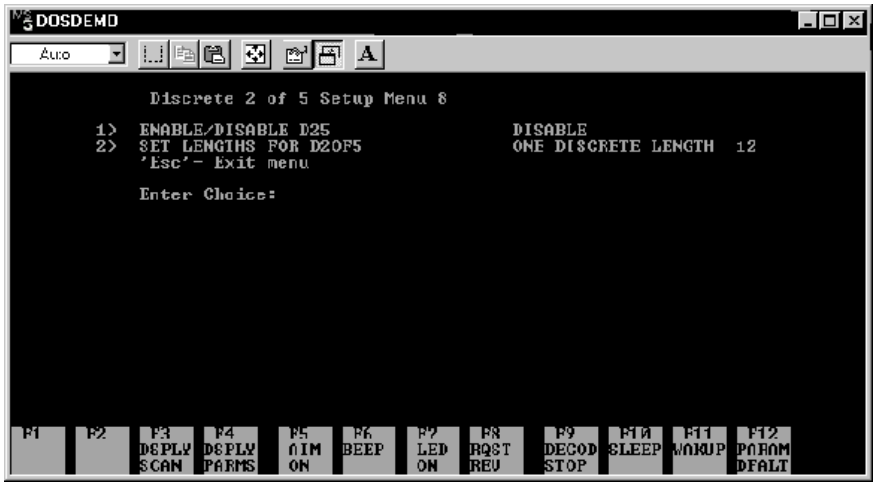

Figure 1-12 *Discrete 2 of 5 Setup Menu*

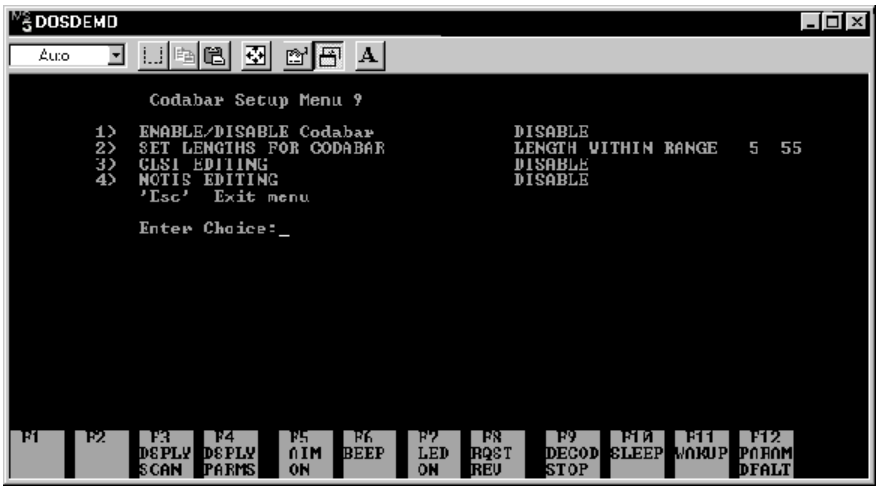

Figure 1-13 *Codabar Setup Menu*

Figure 1-14 *MSI Plessey Setup Menu*

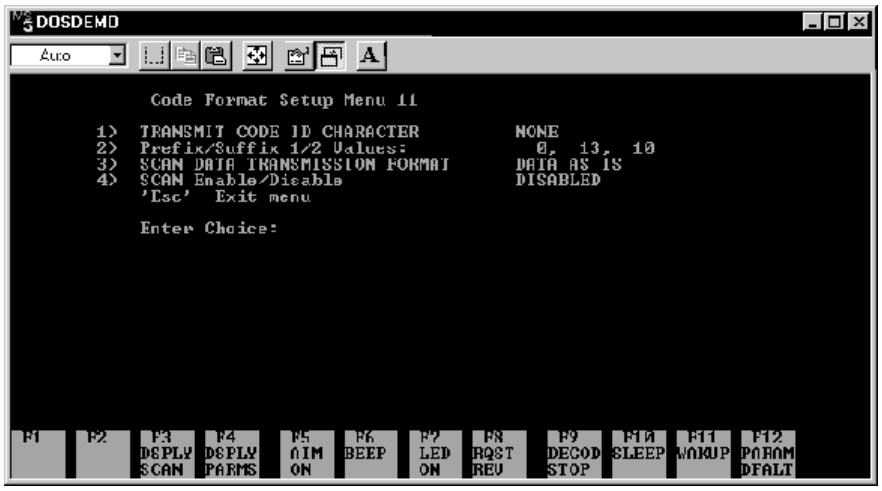

Figure 1-15 *Code Format Setup Menu*

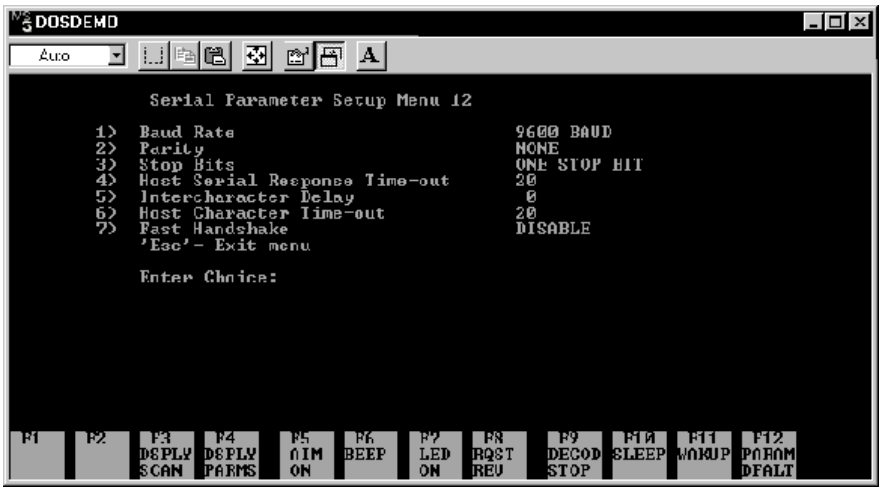

Figure 1-16 *Serial Parameter Setup Menu*

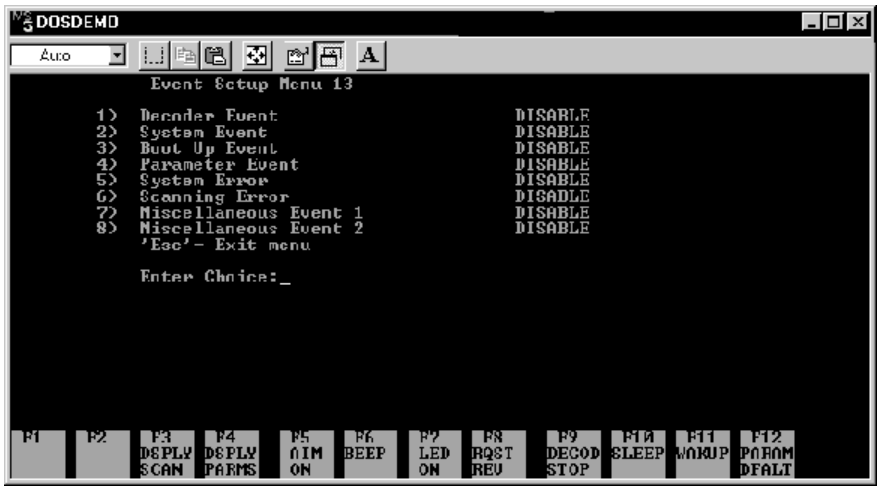

Figure 1-17 *Event Setup Menu*

Building the SSI SDK (32-bit Windows Version)

Using the build tool Microsoft Visual Studio 6.0, open the project **ssiwin32.dsp** from the **\lib** directory on your SSI SDK CD.

1. In the **File** menu, select **Open.**
2. Navigate to your CD drive.
3. Select the **\Lib** directory.
4. In Files of type, select **Projects(.dsp)** from the pull down menu.

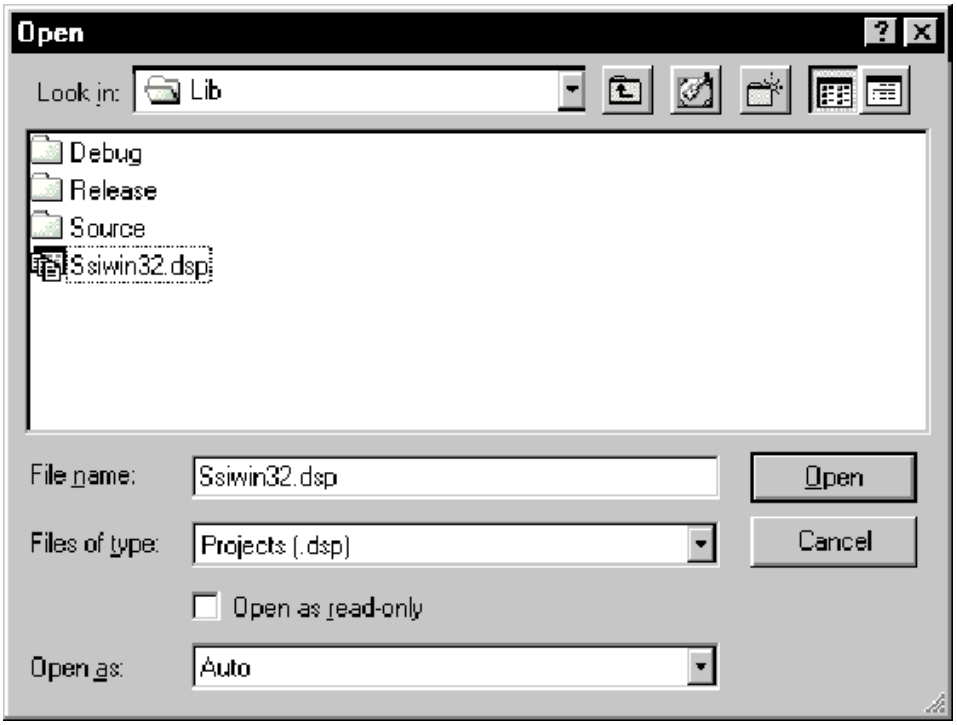

Figure 1-18 *Opening Ssiwin32 Project*

5. To build the project, select **Build**, then **Rebuild All.** This creates the **Ssiwin32.lib**. After building the project, the Build tab of the output window displays:

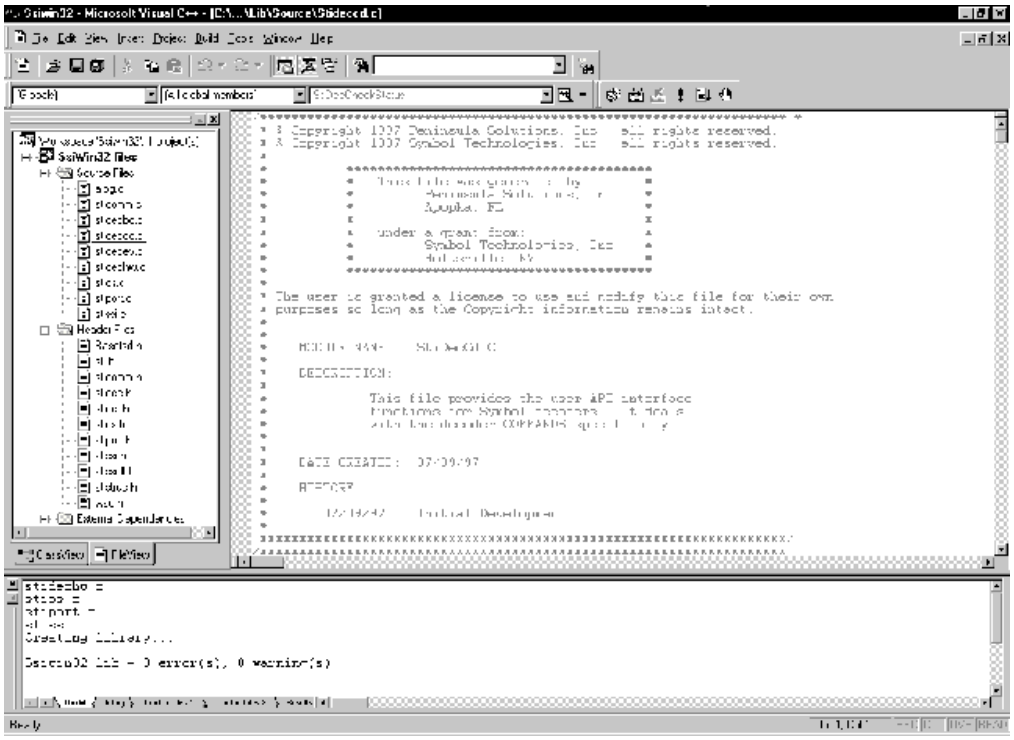

Figure 1-19 *Ssiwin32.lib Build Tab*

The **win32test** project can now be built.

Building the Win32test Project

Open the project **Win32Test.dsp** from the **\ssiwin32** directory on your SDK CD.

1. In the **File** menu, select **Open.**
2. Navigate to your CD drive.
3. Select the **\Win32Test** directory.
4. In Files of type, select **Projects(.dsp)** from the pull down menu.

5. Select **Open**.

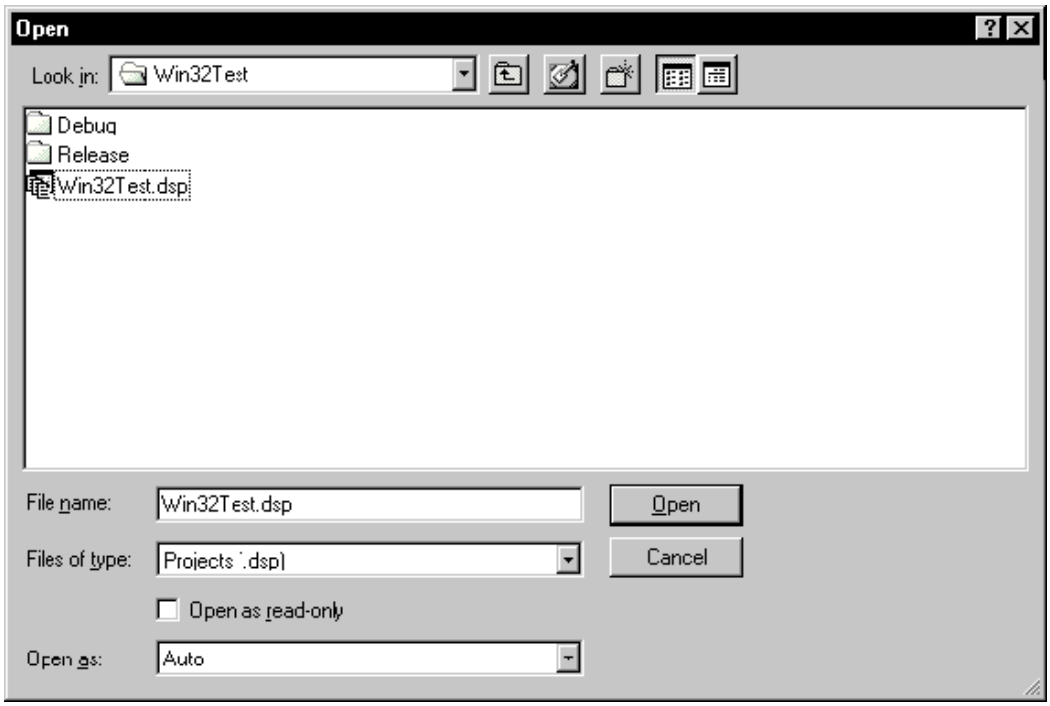

Figure 1-20 *Opening Win32Test Project*

6. To build the project, select **Build**, then **Rebuild All.** This creates the **Win32Test.exe**. After building the project, the output window displays:

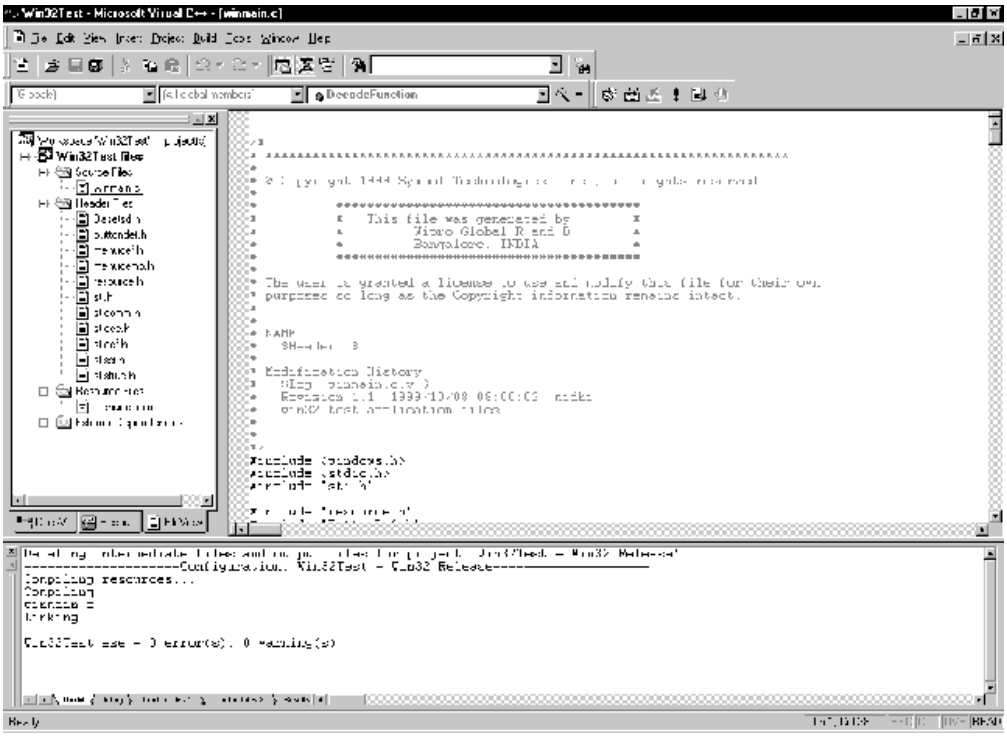

Figure 1-21 *Win32Test.exe Build Tab*

Using the SSI SDK Demo Application (32-bit Windows Version)

Connect the device to Com2. From the **Win32Test** directory, select **Win32Test.exe**.

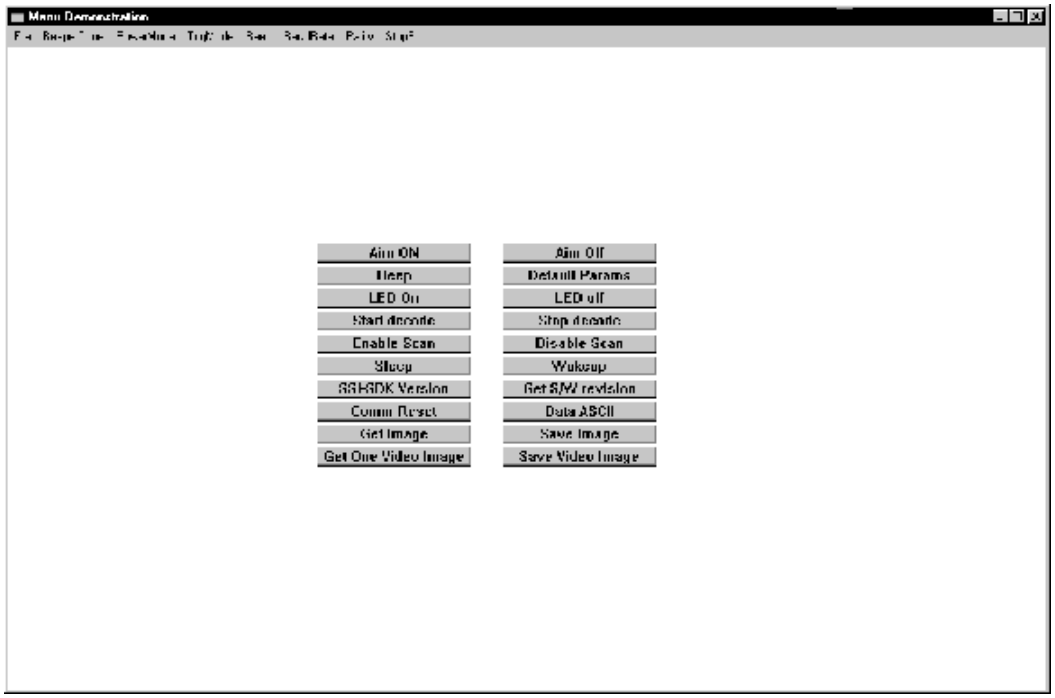

Figure 1-22 *Demo Application Window*

The demo application supports the following SSI Commands:

- Aim On: Activates the aim pattern.
- Beep: Sounds the beeper.
- LED On: Activates the Decode LED output.
- Start decode: Tells the decoder to attempt to decode a bar code.
- Enable scan: Permits barcode scanning.
- Sleep: Requests to place the decoder into low power.
- SSI-SDK Version: Replies with the SDK version.
- Comm Reset: Resets communication settings.
- Get Image: Captures an image.
- Get one Video Image: Captures one video image.
- Aim Off: Deactivates aim pattern.
- Default Params: Sets parameter default values.
- LED Off: Turns off the Decode LED.
- Stop Decode: Tells decoder to abort a decode event.
- Disable Scan: Prevents the user from scanning bar codes.

- Wakeup: Wakes the decoder after it's been powered down.
- Get S/W revision: Replies with the decoder's software configuration.
- Data ASCII: Displays ASCII bar code value.
- Save Image: Saves a captured image.
- Save Video Image: Saves a video image.

The demo application also provides the following menu options:

- Beeper Tone: Sets beeper tone.
- Power Mode: Sets the power mode to low power or continuous on.
- Trig Mode: Sets the trigger mode to host, level, continuous, blink, or pulse.
- Beep: Enable/Disables the beeper.
- Baudrate: Sets the baud rate.
- Parity: Sets the parity.
- StopBit: Sets the number of stop bits.

Chapter 2 Introduction to SSI

Introduction

This chapter describes the system requirements of the Simple Serial Interface (SSI), which provides a communications link between Symbol Technologies decoders and a serial host. Information is provided from the perspective of both the host and the decoder.

The following must be understood before using SSI:

- The SSI interface provides a means for the host to control the decoder.
- SSI is a half-duplex communication protocol.
- SSI is transaction-based, that is, the host commands and the decoder responds. For example, the host commands “beep the beeper” and the decoder both beeps and “ACKs” as a response. Acknowledgements are vital for maintaining synchronization.

The following sections describe the basic hardware layer (signals and handshaking) first, followed by software protocol, and finishing with a description of message packets.

Serial Parameter Settings

For communication to occur serial parameters must match between the host and the decoder. These parameters can be set using information in the Product Reference Guide supplied with your decoder.

The default parameters are:

- Baud Rate:9600 Baud
- Data Bits:8 bits
- Number of parity bits:1 bit
- Parity: None
- Stop Bits:1
- Hardware Handshaking:Always
- Software Handshaking:On
- Inter-Packet Delay:0 milliseconds
- Multi-Packet Option:Option 1

Settings for other parameters related to image capture, video capture and other decoder performances should also be reviewed before using SSI.

Hardware Signals

The hardware layer of SSI consists of four signals: Transmit Data (TXD), Receive Data (RXD), Request to Send (RTS) and Clear to Send (CTS).

From the decoder's perspective:

- TXD:Serial data transmit output. Drives the serial data receive input of the host.
- RXD:Serial data receive input. Driven by the serial data transmit output of the host.
- RTS:Drives host CTS, Decoder Output. Acknowledges host demand to transmit.
- CTS:Driven by HOST RTS. Decoder Input. Host Demand / Interrupt to receive.

From the host's perspective:

- HOST RXD:Serial data receive input. Driven by the serial data transmit output of the decoder.
- HOST TXD:Serial data transmit output. Drives the serial data receive input of the decoder.
- HOST CTS:Driven by decoder RTS. Host Input. Decoder ACK of host demand to transmit.
- HOST RTS:Host demand / interrupt to transmit. Must be honored by decoder.

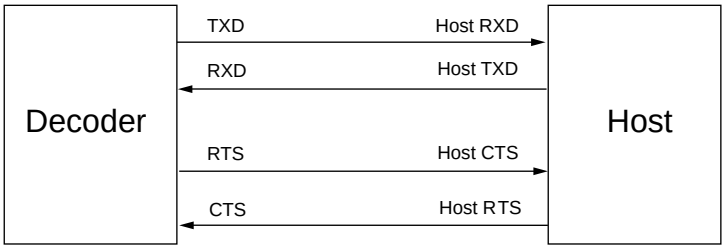

Figure 2-1 *Host/Decoder Interconnection*

Hardware Handshaking

The four hardware signals are used to perform host transmission to decoder, and decoder transmission to host. The host is the Bus Master and controls the actions of SSI. In cases of collision or arbitration, the decoder always defers to the actions of the host. The host programmer should adhere to these specifications closely.

Host Transmission to Decoder

When the host wants to transmit data to the decoder:

1. The host changes the state of the HOST RTS line from inactive to active.
2. The decoder replies by changing the state of the HOST CTS line from inactive to active, within the time frame set by the response timeout parameter.
3. Upon detecting the active HOST CTS, the host transmits the message on the HOST TXD line, verifying that HOST CTS remains active on a character-by-character basis.
4. After sending all characters in the message, the host must change the state of the HOST RTS line from active to inactive.
5. The decoder responds by changing the state of the HOST CTS line from active to inactive.

If the decoder does not respond with HOST CTS when it detects HOST RTS within the programmable (via parameters) response timeout, the host may retry the message. Only after multiple failure attempts spread over some period of time should the host declare the interface 'non-viable'. The decoder may be performing a time-intensive function which precludes talking to the host, resulting in the failure of the HOST CTS to go active.

Only one message per HOST RTS/CTS handshake is recognized, so if the host sends two (or more) messages without changing the state of the HOST RTS line, the decoder ignores the second (and subsequent) messages. Once the number of characters indicated by the host is received (plus the two bytes required for a checksum), further characters are ignored. The host may briefly toggle the state of the RTS line during this transaction, as long as the inter-character timeout is not exceeded.

While the host is not required to continuously stream characters, the maximum character-to-character delay cannot exceed the host Intercharacter Timeout parameter. If this timeout is exceeded, the decoder waits for the host to de-assert HOST RTS. Once the HOST RTS is in-active, the decoder issues a NAK message, which causes the host to re-try the entire message.

Host Transmission Sample Code

```
boolean host_transmit()
    request permission to send  [HOST RTS active]
    WHILE (the last character has not been sent) DO
        set up serial response time out
        WHILE (permission has not been granted [HOST CTS is inactive]) DO
            IF (serial response time out expired) THEN
                remove request to send   /* transmit failed */
                /* calling function may retry transmit */
                RETURN (FALSE)
            END
        END
        transmit a character
    END
    remove request to send [HOST RTS inactive]
    RETURN (TRUE)   /* transmit successful */
```

Decoder Reception of Host Transmission

The decoder constantly monitors the CTS line for activity:

1. When CTS is made active by the host, the decoder responds by making the RTS line active, even if the decoder was attempting to transmit, deferring to host action.
2. The decoder monitors the RXD line and receives the characters comprising the message.
3. When all characters are received (length of message plus the two byte checksum) the decoder acts upon the message/command and ignores further characters.
4. When the decoder detects that CTS is inactive, it makes RTS inactive and transmits the response.

If the host exceeds the inter-character timeout delay, the decoder waits for de-assertion of the CTS line to send a NAK message to the host.

There are two cases where the host may make the CTS line inactive before the message is complete. If the host did not send any characters, there is no message, so the decoder does not reply. If, however, the first character is sent, an incomplete message results in a NAK.

Decoder Reception Sample Code

This pseudo code assumes that the receiving is enabled.

```
void decoder_receive()
    IF (host is requesting to send [CTS active]) THEN
        give host permission to send [RTS active]
        WHILE (no characters received) DO
            IF (host not requesting to send [CTS Inactive]) THEN
                remove host's permission to send
                RETURN   /* NULL xmit - do not NAK */
            END
        END
        set up host character time out
        WHILE (not timed out AND not the last character) DO
            IF (a character was received) THEN
                reset host character time out
            END
        END
        WHILE (host is requesting to send [CTS active]) DO
            wait   /* for host to end handshake */
        END
        remove host's permission to send [RTS inactive]
        process received message and prepare response
    END
    RETURN
```

Decoder Transmission to Host

The decoder is sometimes required to send data to the host. This transaction is more complicated than the host transmission.

The decoder first ensures that the host is not attempting to send by examining the state of CTS. If the host is attempting to send, the decoder defers transmission and permits the host to send as described previously. The response to the host transmission is given by the decoder before the transmission pending. For example, if the decoder is sending decode data (as a result of scanning a bar code) and the host interrupts with a beep the beeper message, the beeper message is acted upon and acknowledged prior to transmitting the decode data message.

The host may temporarily hold off the decoder transmission by making the HOST RTS line active until it is ready to receive. If the host does not send any characters, the decoder resumes transmission when the HOST RTS line becomes inactive. If, however, the host sends one or more characters, the decoder resends the entire packet from the start when the HOST RTS line becomes inactive.

If the host is not trying to send, the decoder sends one character of the transmission on the TXD line. The decoder again checks the state of the CTS line, so the host may interrupt on a character-by-character basis. While the host should try to avoid interrupting the decoder transmission, bus collisions are possible and must defer to the host.

Decoder Transmission Sample Code

```
boolean decoder_xmit()
    /* insure that the host is not trying to send something */
    IF (host is requesting to send [CTS active]) THEN
        /* host attempt to send…grant permission */
        enable receiving
        give host permission to send [RTS active]
        set up serial response time out
        WHILE (host is still requesting to send [CTS active]) DO
            IF (character was received OR timed out) THEN
                /* abort transmission…try again later */
                RETURN (FALSE)
            END
        END
        disable receiving
        remove host's permission to send [RTS inactive]
    END
    WHILE (there are characters to send) DO
        IF (host is not requesting to send [CTS inactive]) THEN
            send next character
        ELSE
            /* the host is either holding us off or beginning  */
            /* a transmission to the decoder                   */
            enable receiving
            give host permission to send [RTS active]
            WHILE (host is still requesting to send [CTS active]) DO
                IF (character was received) THEN
                    /* beginning of a host transmission   */
                    /* abort transmission…try again later */
                    RETURN (FALSE)
                END
            END
            /* by virtue of code flow to here the host */
            /* was merely holding off the decoder      */
            disable receiving
            remove host's permission to send [RTS inactive]
        END   /* resume transmit */
    END
    RETURN (TRUE)
```

Host Reception of Decoder Transmission

The host must be ready to receive data from the decoder anytime the host is not transmitting. The host can temporarily hold off the decoder transmission by keeping the HOST RTS line asserted. The host can also interrupt an incoming message by asserting HOST RTS.

Since the decoder does not change the state of the HOST CTS line when transmitting, the host is aware of a decoder transmission only when it receives the first character. For each subsequent character, the host sets up an intercharacter timeout. If this timeout has not expired and the host has not received the last character of the transmission, the host receives the next character.

Host Reception Sample Code

```
void host_receive()
    IF (a character has been received) THEN
        set up intercharacter time out
        WHILE (not timed out AND not the last character) DO
            IF (host can receive right now) THEN
                deassert HOST RTS   /* in case host was holding off decoder */
                IF (a character was received) THEN
                    reset intercharacter time out
                END
            ELSE
                IF (host wants to send to decoder) THEN
                    RETURN   /* so host can transmit */
                ELSE
                    /* host does not want to send but needs some time */
                    assert request to send [HOST RTS active]   /*hold off
                    decoder */
                    set up new intercharacter time-out
                END
            END
        END
        process received message and prepare response
        RETURN
    END
    RETURN
```

Software Handshaking

Software handshaking provides an ACK/NAK response for commands that do not have a natural response. For example, the command “tell me your parameters” is followed by the response “my parameters are X”. A “start a decode session” command, however, has no natural response, so software handshaking provides an ACK/NAK response.

ACK/NAK handshaking may be enabled (default) or disabled. If enabled, all packeted messages must have a response in the form of an ACK, or a NAK of various types. We recommend this handshaking remain enabled to provide feedback to the host.

Raw decode data and the WAKEUP command do not use ACK/NAK handshaking since they are not packeted data.

Transfer of Decode Data

The Decode Data Packet Format parameter controls how decode data is sent to the host. When this parameter is enabled, the data is sent in a DECODE_DATA packet. When disabled, data is transmitted as raw ASCII data.

NOTE When decode data is transmitted as raw ASCII data, ACK/NAK handshaking does not apply even if it is enabled.

ACK/NAK Enabled and Packeted Data

The decoder sends a DECODE_DATA message after a successful decode. The decoder waits for a programmable time-out for a CMD_ACK response. If it does not receive the response, the decoder tries to send two more times before issuing a host transmission error. If the decoder receives a CMD_NAK from the host, it may attempt a retry depending on the cause field of the CMD_NAK message.

ACK/NAK Enabled and Unpacketed ASCII Data

The decoder sends a RAW DATA message after a successful decode. No ACK/NAK handshaking occurs even though it is enabled because data is unpacketed.

ACK/NAK Disabled and Decode Data of Any Type

The decoder sends the decode data. No response is expected from the host since ACK/NAK handshaking is disabled. The security of this transaction is not guaranteed.

Unsolicited ACK/NAK

An unsolicited ACK or NAK is an unexpected message, and is ignored since the decoder can not interpret the message.

CMD_NAK Cancel is a special case of transmission by the host, so is considered a solicited message. This message halts (and discards) an unwanted transmission by the decoder. For example, a large image sent by the decoder to the host is typically multi-packeted, and consists of a large transmission. The host may cancel this by transmitting a CMD_NAK Cancel.

Expected Responses

The following tables list allowable decoder and host responses.

Table 2-1. Decoder Responses to Host Transmission

Host Transmission	Allowable Decoder Responses
AIM_OFF	CMD_ACK / CMD_NAK
AIM_ON	CMD_ACK / CMD_NAK
BATCH_REQUEST	BATCH_DATA / CMD_NAK
BEEP	CMD_ACK / CMD_NAK
CAPABILITIES_REQUEST	CAPABILITIES_REPLY
CMD_ACK	None
CMD_NAK	None
FLUSH_QUEUE	CMD_ACK / CMD_NAK
IMAGER_MODE	CMD_ACK / CMD_NAK
LED_OFF	CMD_ACK / CMD_NAK
LED_ON	CMD_ACK / CMD_NAK
PARAM_DEFAULTS	CMD_ACK / CMD_NAK
PARAM_REQUEST	PARAM_SEND
PARAM_SEND	CMD_ACK / CMD_NAK

Table 2-1. Decoder Responses to Host Transmission (Continued)

Host Transmission	Allowable Decoder Responses
REQUEST_REVISION	REPLY_REVISION
SCAN_DISABLE	CMD_ACK / CMD_NAK
SCAN_ENABLE	CMD_ACK / CMD_NAK
SLEEP	CMD_ACK / CMD_NAK
START_SESSION	CMD_ACK / CMD_NAK Note that once the decoder gathers the appropriate data, it sends this data unsolicited.
STOP_SESSION	CMD_ACK / CMD_NAK
WAKEUP	None

Table 2-2. Host Responses to Decoder Transmission

Decoder Transmission	Allowable Host Responses
CAPABILITIES_REPLY	None
CMD_ACK	None
CMD_NAK	None
DECODE_DATA	CMD_ACK / CMD_NAK *
EVENT	CMD_ACK / CMD_NAK *
IMAGE_DATA	CMD_ACK / CMD_NAK *
PARAM_SEND	None
REPLY_REVISION	None
VIDEO_DATA	CMD_ACK / CMD_NAK
*Multipacketed data; the host may ACK/NAK only the last packet of a multi-packeted message. Intermediate packets get no response. Intermediate packets always have the continuation bit set (1). The last packet has the continuation bit cleared (0). See <Blue><Italic>Multipacketing on page -10 for multi-packeting options.	

Message Packets

All communications between the host and the decoder are exchanged in the form of packets. A packet is a collection of bytes framed by the proper SSI protocol formatting bytes. The maximum length of a packet is 257 bytes, consisting of a checksum (two bytes), a header (four bytes), and up to 251 characters of data. Note that the length field in the header does NOT include the length of the checksum, but DOES include the length of the header itself.

Multipacketing

SSI supports multiple packets for one message for cases when size is insufficient to transfer a complete message. Bit1 of the status byte in the message header is set to one for all packets except the last to indicate another packet is to follow. In the last packet, this bit is set to zero. The host must re-assemble these packets into one message. The decoder sends each packet in order.

Multipacketing, Option 1

The host ACK/NAKs each packet in a strict transaction-based method. If a CMD_NAK checksum message occurs, the decoder retransmits the packet that was NAK'd.

Multipacketing, Option 2

The decoder sends data packets continuously, with no ACK/NAK handshaking to pace the transmission. If the host is overrun, it can use hardware handshaking to temporarily hold off the decoder.

At the end of transmission, the decoder waits for a CMD_ACK or CMD_NAK. The host acknowledges the transmission, or requests the entire multi-packet message be resent from the first packet.

The host may stop the transmission from the decoder at any time by asserting hardware handshaking. The host then transmits either CMD_NAK, resend to instruct the decoder to resend the entire message, or CMD_NAK, cancel to cancel the transmission completely. Note that because these NAKs are unexpected, interruption of transmission must occur first.

Multipacketing, Option 3

Option 3 is similar to Option 2, except there is a programmable inter-packet delay. The decoder waits a programmed period after sending each packet. This may be faster than Option 1 because the host receives data on a periodic basis without attempting to send the ACK/NAK, but slower than Option 2 since the inter-packet delay transpires on each packet. However, it helps prevent host receiver overrun.

Packet Format

The general packet format for SSI messages is as follows:

Length	Opcode	Message Source	Status	Data...	Checksum

Table 2-3. Field Descriptions

Field Name	Format	Sub-Field	Meaning
Length	1 Byte	Length	Length of message not including the check sum bytes. Maximum value is 0xFF.
Opcode	1 Byte	See <Blue><Italic>SSI Command Lists on page A-1.	Identifies the type of packet data sent.
Message Source	1 Byte	0 = Decoder, 04 = Host	Identifies where the message is coming from.
Status	Bit 0	Retransmit	0 = First time packet is sent 1 = Subsequent transmission attempts
	Bit 1	Continuation Bit	0 = Last frame of a multipacket message 1 =Intermediate packet of a multipacket message
	Bit 2	Reserved	Always set to zero
	Bit 3	Change Type (applies to parameters)	0 = Temporary change 1 = Permanent change
	Bits 4 - 7		Unused bits must be set to 0.
Data...	Variable number of bytes	See individual sections for details.	
Checksum	2 Bytes	2's complement sum of message contents excluding checksum.	Checksum of message formatted as HIGH BYTE LOW BYTE
Note: The checksum is a 2 byte checksum and must be sent as HIGH BYTE followed by LOW BYTE.			

Chapter 3 Decoder API

Introduction

The host application uses the Host API Function Set to direct the decoder's operation and receive information from the decoder. All SSI interface functions are available to the host via the functions in the Decoder API.

To promote an event-driven architecture for the SDK, use callback function registration, even in the single-threaded DOS implementation, to enable interrupt-driven background processing of the communication tasks and to return control to the host application between interface events.

The following decoder capabilities, as a minimum, are defined in terms of callback registration:

- Decode Event
- System Event
- Boot Up Event
- Parameter Event
- System Error
- Scanning Error
- Miscellaneous Event 1
- Miscellaneous Event 2

Return Value Definitions

The functions in this chapter may return the following standard status codes:

- Any non-negative value (0 to 32767): parameter value.
- STATUS_OK: the function parameters were verified. If a function must wait for an ACK from the decoder, STATUS_OK indicates the ACK was received.
- MSG_PENDING: a message is pending in the transmit buffer. No further messages can be placed in the buffer in this condition; the program must wait until there is room for a new message in the transmit buffer.
- NOT_SUPPORTED: the last packet received from the decoder was either a NAK_DENIED or NAK_BAD_CONTEXT. When retrieving parameters from the decoder, this indicates that the specified parameter is not supported by this decoder, or that it was unable to comply with the request.
- COMMUNICATIONS_ERROR: either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not acknowledged, and therefore is questionable.
- BAD_PARAM: the user-supplied parameter(s) in the function call was not in the expected range.
- BATCH_ERROR: the limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with **StiDecSet**_ are responsible for generating a batch command to establish decoder parameters. The parameters are not sent to the decoder until the **StiDecSendParams()** function is called, when a new batch is started.
- ERROR_UNDEFINED: an error condition not specifically associated with the decoder or its communications.

Sti DecCheckStatus

Description

Returns the status of the current message.

Syntax

int far StiDecCheckStatus(*void*)

Return Values

STATUS_OK
MSG_PENDING
NAK_DENIED
NOT_SUPPORTED
COMM_ERROR

StiDecCmdAimOff

Description

Turns off the aiming pattern.

Syntax

int far StiDecCmdAimOff(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdAimOn

Description

Turns on the aiming pattern.

Syntax

int far StiDecCmdAimOn(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdBeep

Description

Sounds the beeper.

Syntax

int far StiDecCmdBeep(*int beep, int func_wait*)

where:

beep is one of the following values:

ONE_SHORT_HIGH
TWO_SHORT_HIGH
THREE_SHORT_HIGH
FOUR_SHORT_HIGH
FIVE_SHORT_HIGH

ONE_SHORT_LOW
TWO_SHORT_LOW
THREE_SHORT_LOW
FOUR_SHORT_LOW
FIVE_SHORT_LOW

ONE_LONG_HIGH
TWO_LONG_HIGH
THREE_LONG_HIGH
FOUR_LONG_HIGH
FIVE_LONG_HIGH

ONE_LONG_LOW
TWO_LONG_LOW
THREE_LONG_LOW
FOUR_LONG_LOW
FIVE_LONG_LOW

FAST_WARBLE
SLOW_WARBLE
MIX1
MIX2
MIX3
MIX4

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

StiDecCmdDecodeData

Description

Fills in a structure with type, decoded data, length, and checksum information. Decode data in SSI packet format.

Syntax

int far StiDecCmdDecodeData(*DECODE_DATA_STRUCT *ptr, int func_wait*)

where:

ptr is a pointer to the user allocated DECODE_DATA_STRUCT where the decoded data is to be placed.

ptr->length indicates the number of characters in the decoded data string

ptr->data contains the decoded data

ptr->data[ptr->length] is the start of the packet checksum

ptr->type indicates what type of barcode was decoded:

DECODE_DATA
BCTYPE_NOT_APPLICABLE
BCTYPE_CODE39
BCTYPE_CODABAR
BCTYPE_CODE128
BCTYPE_D2OF5
BCTYPE_IATA2OF5
BCTYPE_I2OF5
BCTYPE_CODE93
BCTYPE_UPCA
BCTYPE_UPCA_2SUPPLEMENTALS
BCTYPE_UPCA_5SUPPLEMENTALS
BCTYPE_UPCE0
BCTYPE_UPCE0_2SUPPLEMENTALS
BCTYPE_UPCE0_5SUPPLEMENTALS
BCTYPE_EAN8
BCTYPE_EAN8_2SUPPLEMENTALS
BCTYPE_EAN13_5SUPPLEMENTALS
BCTYPE_EAN8_5SUPPLEMENTALS
BCTYPE_EAN13
BCTYPE_EAN13_2SUPPLEMENTALS
BCTYPE_MSI_PLESSEY
BCTYPE_EAN128
BCTYPE_UPCE1
BCTYPE_UPCE1_2SUPPLEMENTALS
BCTYPE_UPCE1_5SUPPLEMENTALS

BCTYPE_CODE39_FULL_ASCII

BCTYPE_TRIOPTIC_CODE39

BCTYPE_BOOKLAND_EAN

BCTYPE_COUPON_CODE

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

StiDecCmdGetImage

Description

Tells the decoder to enter image capture mode.

Syntax

int CDECL STIFAR StiDecCmdGetImage(*int func_wait*);

where:

func_wait is one of the following values:

WAIT: waits for the function to complete before returning to the caller.
NO_WAIT: does not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdGetVideo

Description

Tells the decoder to enter Video capture mode.

Syntax

int CDECL STIFAR StiDecCmdGetVideo(*int func_wait*);

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

StiDecCmdLedOff

Description

De-activates LED output.

Syntax

int far StiDecCmdLedOff(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdLedOn

Description

Activates LED output.

Syntax

int far StiDecCmdLedOn(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdParamDefaults

Description

Sets all parameters to the factory defaults.

Syntax

int far StiDecCmdParamDefaults(*void*)

Return Values

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecCmdRequestRevision

Description

Requests the software revision string from the decoder.

Syntax

int far StiDecCmdRequestRevision(*void, int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdScanDisable

Description

Prevents the decoder from scanning bar codes.

Syntax

int far StiDecCmdScanDisable(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdScanEnable

Description

Permits the decoder to scan bar codes.

Syntax

int far StiDecCmdScanEnable(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdSleep

Description

Request to place the decoder into low power mode.

Syntax

int far StiDecCmdSleep(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdStartDecode

Description

Tells decoder to attempt to decode a bar code.

Syntax

int far StiDecCmdStartDecode(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdStopDecode

Description

Tells decoder to abort a decode attempt.

Syntax

int far StiDecCmdStopDecode(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.
NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING
BAD_PARAM

StiDecCmdWakeUp

Description

Wakes up decoder after it's been put into low power operation. The WAKEUP character has no effect if sent when the decoder is awake. If the host is unsure of the decoder state, it should send the wakeup character anytime it wants to communicate with the decoder.

Syntax

int far StiDecCmdWakeUp(*int func_wait*)

where:

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

StiDecEventCount

Description

Returns the current count for the requested event (listed below) and resets the count for that event.

NOTE System event codes from the decoder.

Syntax

int far StiDecEventCount(*int event*)

where:

event is one of the following values:

DECODE_EVENT
SYSTEM_EVENT
BOOT_UP_EVENT
PARAMETER_EVENT
SYSTEM_ERROR
SCANNING_ERROR
MISCELLANEOUS_EVENT1
MISCELLANEOUS_EVENT2

StiDecGetAimDuration

Description

Gets the duration the aiming pattern is seen before a scan attempt begins for long-range (LR) or high-visibility (HV) scanners triggered either by a trigger pull or a START_DECODE command.

Syntax

int far StiDecGetAimDuration(*void*)

Return Values

Returns an integer value in the range [0..99] , representing a time period from 0.0 to 9.9 seconds in 0.1 second increments. If an error occurs, the function returns one of the following values:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetBeepAfterGoodDecode

Description

Returns whether beeping after a good decode is enabled. If disabled, the beeper still operates during parameter menu scanning and indicates error conditions.

Syntax

int far StiDecGetBeepAfterGoodDecode(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetBeeperTone

Description

Gets the decode beep frequency (tone).

Syntax

int far StiDecGetBeeperTone(*void*)

Return Values

LOW_FREQUENCY
MEDIUM_FREQUENCY
HIGH_FREQUENCY
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetBidirectionalRedundancy

Description

Returns whether a bar code must be successfully scanned in both directions (forward and reverse) before being decoded.

Syntax

int far StiDecGetBidirectionalRedundancy(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetBooklandEan

Description

Returns whether Bookland EAN is enabled or disabled.

Syntax

int far StiDecGetBooklandEan(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetBootUpEvent

Description

Returns whether the Boot Up Event option is enabled or disabled.

Syntax

int StiDecGetBootUpEvent(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetClsiEditing

Description

Returns whether CLSI Editing is enabled or disabled.

Syntax

int far StiDecGetClsiEditing(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetCodabar

Description

Returns whether Codabar is enabled or disabled.

Syntax

int far StiDecGetCodabar(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetCode128

Description

Returns whether Code 128 is enabled or disabled.

Syntax

int far StiDecGetCode128(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetCode32Prefix

Description

Appends the character 'A' to the start of the decode data if enabled.

Syntax

int far StiDecGetCode32Prefix(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetCode39

Description

Returns whether Code 39 is enabled or disabled.

Syntax

int far StiDecGetCode39(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetCode39CheckDigit

Description

Returns whether Code 39 check digit is enabled or disabled.

Syntax

int far StiDecGetCode39CheckDigit(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetCode39FullAscii

Description

Returns whether Code 39 Full ASCII is enabled or disabled.

Syntax

int far StiDecGetCode39FullAscii(*void*)

Returns one of the following values:

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetCode93

Description

Returns whether Code 93 is enabled or disabled.

Syntax

int far StiDecGetCode93(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetConvertCode39toCode32

Description

Returns whether Code 39 is converted to Code 32.

Syntax

int far StiDecGetConvertCode39toCode32(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetConvertEan8toEan13

Description

Returns whether an extended symbol is labeled as either an EAN-13 bar code or an EAN-8 bar code.

Syntax

int far StiDecGetConvertEan8toEan13(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetConvertI2of5toEan13

Description

Returns whether a 14-character I 2 of 5 code is converted to EAN-13 and transmitted to the host as EAN-13.

Syntax

int far StiDecGetConvertI2of5toEan13(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetConvertUpcE1toUpcA

Description

Returns whether Convert UPC-E1 (zero suppressed) to UPC-A format before transmission is enabled.

Syntax

int far StiDecGetConvertUpcE1toUpcA(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetConvertUpcEtoUpcA

Description

Returns whether Convert UPC-E (zero suppressed) to UPC-A format before transmission is enabled.

Syntax

int far StiDecGetConvertUpcEtoUpcA(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetD2of5

Description

Returns whether Discrete 2 of 5 is enabled or disabled.

Syntax

int far StiDecGetD2of5(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetDecodeEvent

Description

Returns whether the Decode Event option is enabled or disabled.

Syntax

int StiDecGetDecodeEvent(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetDecodeUpcEanRedundancy

Description

Returns the number of times a symbol without supplementals is decoded before transmission, from 2 to 20 times.

Syntax

int far StiDecGetDecodeUpcEanRedundancy(*void*)

Return Values

Returns a value in the range [0..20]. If an error occurs, the function returns one of the following values:

NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetDecodeUpcEanSupplementals

Description

Returns the Decode UPC/EAN Supplementals option setting. Supplementals are additionally appended characters (2 or 5) according to specific code format conventions (e.g., UPC A+2, UPC E+2, EAN 8+2). Three options are available.

Syntax

int far StiDecGetDecodeUpcEanSupplementals(*void*)

Return Values

DECODE_SUPPLEMENTALS

IGNORE_SUPPLEMENTALS

AUTODISCRIMINATE_SUPPLEMENTALS

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetEan13

Description

Returns whether EAN-13 is enabled or disabled.

Syntax

int far StiDecGetEan13(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetEan8

Description

Returns whether EAN-8 is enabled or disabled.

Syntax

int far StiDecGetEan8(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetEanZeroExtend

Description

Returns whether five leading zeros are added to decoded EAN-8 symbols to make them compatible in format to EAN-13 symbols.

Syntax

int far StiDecGetEanZeroExtend(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetHostCharacterTimeOut

Description

Returns the maximum time the decoder waits between characters transmitted by the host before discarding the received data and declaring an error.

Syntax

int far StiDecGetHostCharacterTimeOut(*void*)

Return Values

Returns a value in the range [0..99] , representing a time period from 0.00 to 0.99 seconds in 0.01 increments. If an error occurs, one of the following status codes is returned:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetHostSerialResponseTimeOut

Description

Returns the Host Serial Response T imeout, which specifies how long the decoder waits for an ACK, NAK or CTS before determining that a transmission error has occurred.

Syntax

int far StiDecGetHostSerialResponseTimeOut(*void*)

Return Values

Returns a value in the range [0..99] , representing a time period from 0.0 to 9.9 seconds in 0.1 increments. If an error occurs, one of the following status codes is returned:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetI2of5

Description

Returns whether Interleaved 2 of 5 is enabled or disabled.

Syntax

int far StiDecGetI2of5(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetI2of5CheckDigitVerification

Description

Checks the integrity of an I 2 of 5 symbol to ensure it complies with specified algorithms, either USS (Uniform Symbology Specification), or OPCC (Optical Product Code Council).

Syntax

int far StiDecGetI2of5CheckDigitVerification(*void*)

Return Values

DISABLE

USS_CHECK_DIGIT

OPCC_CHECK_DIGIT

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetIntercharacterDelay

Description

Returns the intercharacter delay period, which gives the host system time to service its receiver and perform other tasks between characters.

Syntax

int far StiDecGetIntercharacterDelay(*void*)

Return Values

Returns a value in the range [0..99] , representing a time period from 0 to 99 msec in 1 msec increments. If an error occurs, one of the following status codes is returned:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetIsbt128

Description

Returns whether ISBT 128 is enabled or disabled.

Syntax

int far StiDecGetIsbt128(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetLaserOnTime

Description

Gets the maximum time decode processing continues during a scan attempt.

Syntax

int far StiDecGetLaserOnTime(*void*)

Return Values

Returns an integer value in the range [5..99], representing a time period from 0.5 to 9.9 seconds in 0.1 second increments. If an error occurs, the function returns one of the following values:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsCodabar

Description

Returns the setting for Codabar lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsCodabar(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsCode39

Description

Returns the setting for Code 39 lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsCode39(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsCode93

Description

Returns the setting for Code 93 lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsCode93(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsD2of5

Description

Returns the setting for D 2 of 5 lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsD2of5(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsI2of5

Description

Returns the setting for I 2 of 5 lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsI2of5(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetLengthsMsiPlessey

Description

Returns the setting for MSI Plessey lengths. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

Syntax

int far StiDecGetLengthsMsiPlessey(*int far *ptr*)

ptr[0] returns one of the following values:

ONE_DISCRETE_LENGTH

TWO_DISCRETE_LENGTHS

LENGTH_WITHIN_RANGE

ANY_LENGTH

ptr[1] returns length1

ptr[2] returns length2

Return Values

STATUS_OK

BAD_PARAM

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetLinearCodeTypeSecurityLevel

Description

Returns the level of decode security for linear code types (e.g. Code 39, Interleaved 2 of 5) set. Higher security levels are set for low levels of bar code quality. As security levels increase, the decoder's aggressiveness decreases.

Syntax

int far StiDecGetLinearCodeTypeSecurityLevel(*void*)

Return Values

SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3
SECURITY_LEVEL4
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetMiscellaneousEvent1

Description

Returns whether the Miscellaneous Event 1 option is enabled or disabled.

Syntax

int StiDecGetMiscellaneousEvent1(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetMiscellaneousEvent2

Description

Returns whether the Miscellaneous Event 2 option is enabled or disabled.

Syntax

int StiDecGetMiscellaneousEvent2(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetMsiPlessey

Description

Returns whether MSI Plessey is enabled or disabled.

Syntax

int far StiDecGetMsiPlessey(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetMsiPlesseyCheckDigitAlgorithm

Description

Returns whether MOD10/MOD11 or MOD10/MOD10 algorithm is selected.

Syntax

int far StiDecGetMsiPlesseyCheckDigitAlgorithm(*void*)

Return Values

MOD10_MOD11

MOD10_MOD10

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetMsiPlesseyCheckDigits

Description

Returns the number of check digits at the end of the bar code that verify the integrity of the data.

Syntax

int far StiDecGetMsiPlesseyCheckDigits(*void*)

Return Values

ONE_CHECK_DIGIT
TWO_CHECK_DIGITS
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetNotisEditing

Description

Returns whether NOTIS Editing is enabled or disabled.

Syntax

int far StiDecGetNotisEditing(*void*)

Returns one of the following values:

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetParameterEvent

Description

Returns whether the Parameter Event option is enabled or disabled.

Syntax

int StiDecGetParameterEvent(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetParameterScanning

Description

Returns whether decoding of parameter bar codes is enabled. Note that the Set Defaults and Enable Parameter Scanning parameter bar codes can still be decoded.

Syntax

int far StiDecGetParameterScanning(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetPowerMode

Description

Gets the power mode setting, whether power remains on when the decoder is not busy. In low power mode, the decoder enters a low power consumption mode whenever possible. In continuous power mode, the decoder does not automatically enter the low power consumption mode.

Syntax

int far StiDecGetPowerMode(*void*)

Return Values

CONTINUOUS_ON

LOW_POWER

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetPrefixSuffixValues

Description

Returns whether a prefix and/or one or two suffixes is appended to scan data for use in data editing.

Syntax

int far StiDecGetPrefixSuffixValues(*int far *ptr*)

where:

ptr[0] returns prefix

ptr[1] returns suffix_1

ptr[2] returns suffix_2

Return Values

STATUS_OK
BAD_PARAM
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetScanDataTransmissionFormat

Description

Returns the setting of the Scan Data Transmission Format.

Syntax

int far StiDecGetScanDataTransmissionFormat(*void*)

Return Values

DATA_AS_IS
DATA_SUFFIX 1
DATA_SUFFIX 2
DATA_SUFFIX 1_SUFFIX 2
PREFIX_DATA
PREFIX_DATA_SUFFIX 1
PREFIX_DATA_SUFFIX 2
PREFIX_DATA_SUFFIX 1_SUFFIX 2
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetScanningError

Description

Returns whether the Scanning Error option is enabled or disabled.

Syntax

int StiDecGetScanningError(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetSystemError

Description

Returns whether the System Error option is enabled or disabled.

Syntax

int StiDecGetSystemError(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetSystemEvent

Description

Returns whether the System Event option is enabled or disabled.

Syntax

int StiDecGetSystemEvent(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetTimeOutBetweenSameSymbol

Description

In Continuous triggering mode, gets the minimum time that must elapse before the decoder decodes a second bar code which is identical to the first decoded. This reduces the risk of accidentally scanning the same symbol twice.

Syntax

int far StiDecGetTimeOutBetweenSameSymbol(*void*)

Return Values

Returns a value in the range [0..99] , representing a time period from 0.0 to 9.9 seconds in 0.1 increments. If an error occurs, the function returns one of the following values:

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetTransmitCode39CheckDigit

Description

Returns whether Code 39 Check Digit is enabled or disabled.

Syntax

int far StiDecGetTransmitCode39CheckDigit(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

SticDecGetTransmitCodeIdCharacter

Description

Returns whether the Code ID character is transmitted with the data.

Syntax

int far SticDecGetTransmitCodeIdCharacter(*void*)

Return Values

SYMBOL_CODE_ID_CHARACTER

AIM_CODE_ID_CHARACTER

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetTransmitI2of5CheckDigit

Description

Returns whether I 2 of 5 check digit is enabled or disabled.

Syntax

int far StiDecGetTransmitI2of5CheckDigit(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetTransmitMsiPlesseyCheckDigit

Description

Returns whether MSI Plessey Check Digit is transmitted with the data.

Syntax

int far StiDecGetTransmitMsiPlesseyCheckDigit(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetTransmitNoReadMessage

Description

Returns whether the decoder transmits "NR" if a symbol does not decode. Any prefix or suffixes enabled are appended around this message.

Syntax

int far StiDecGetTransmitNoReadMessage(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetTransmitUpcACheckDigit

Description

Returns whether a symbol is transmitted with or without the UPC-A check digit.

Syntax

int far StiDecGetTransmitUpcACheckDigit(*void*)

Return Values

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetTransmitUpcE1CheckDigit

Description

Returns whether a symbol is transmitted with or without the UPC-E1 check digit.

Syntax

int far StiDecGetTransmitUpcE1CheckDigit(*void*)

Return Values

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetTransmitUpcECheckDigit

Description

Returns whether a symbol is transmitted with or without the UPC-E check digit.

Syntax

int far StiDecGetTransmitUpcECheckDigit(*void*)

Return Values

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetTriggeringModes

Description

Gets the triggering mode of the decoder.

Syntax

int far StiDecGetTriggeringModes(*void*)

Return Values

LEVEL
PULSE
CONTINUOUS
BLINKING
HOST
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Sti DecGetTriopticCode39

Description

Returns whether Trioptic Code 39 is enabled or disabled.

Syntax

int far StiDecGetTriopticCode39(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetUccEan128

Description

Returns whether UCC/EAN-128 is enabled or disabled.

Syntax

int far StiDecGetUccEan128(*void*)

Returns one of the following values:

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetUpcA

Description

Returns whether UPC-A is enabled or disabled.

Syntax

int far StiDecGetUpcA(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetUpcAPreamble

Description

Returns the selected UPC-A Preamble option: transmit system character only, transmit system character and country code ("0" for USA), or no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecGetUpcAPreamble(*void*)

Return Values

NO_PREAMBLE

SYSTEM_CHARACTER

SYSTEM_CHARACTER_COUNTRY_CODE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetUpcE

Description

Returns whether UPC-E is enabled or disabled.

Syntax

int far StiDecGetUpcE(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetUpcE1

Description

Returns whether UPC-E1 is enabled or disabled.

Syntax

int far StiDecGetUpcE1(*void*)

Return Values

ENABLE
DISABLE
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetUpcEanCouponCode

Description

Returns whether UPC-A, UPC-A with 2 supplemental characters, UPC-A with 5 supplemental characters, and UPC-A/EAN128 bar codes are decoded.

Syntax

int far StiDecGetUpcEanCouponCode(*void*)

Return Values

ENABLE

DISABLE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecGetUpcEanSecurityLevel

Description

Returns the UPC/EAN Security Level selected.

Syntax

int far StiDecGetUpcEanSecurityLevel(*void*)

Return Values

SECURITY_LEVEL0
SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3
NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecGetUpcE1Preamble

Description

Returns the selected UPC-E1 Preamble option: transmit system character only, transmit system character and country code (“0” for USA), or no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecGetUpcE1Preamble(*void*)

Return Values

NO_PREAMBLE

SYSTEM_CHARACTER

SYSTEM_CHARACTER_COUNTRY_CODE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

Sti DecGetUpcEPreamble

Description

Returns the selected UPC-E Preamble option: transmit system character only, transmit system character and country code ("0" for USA), or no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecGetUpcEPreamble(*void*)

Return Values

NO_PREAMBLE

SYSTEM_CHARACTER

SYSTEM_CHARACTER_COUNTRY_CODE

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiDecRspReplyRevision

Description

Returns the revision string obtained through the REQUEST_REVISION command.

Syntax

int far StiDecRspReplyRevision(*char *ptr, int max_length*)

Copies the ASCII revision string to a user specified location.

where:

**ptr* is a pointer to a user allocated char array. This function places the revision into the array, null terminated.

max_length is a maximum number of characters to be copied to *ptr[ ]*.

Return Values

Length of the revision string if no errors are encountered, otherwise,

NOT_SUPPORTED
COMMUNICATIONS_ERROR

StiDecSendParams

Description

Sends batched parameters. Parameters set with the following functions (**StiDecSet**.....) are not sent to the decoder until **StiDecSendParams** is executed.

Syntax

int far StiDecSendParams(*int change, int func_wait*)

where:

change is one of the following values:

TEMPORARY
PERMANENT

func_wait is one of the following values:

WAIT	Wait for the function to complete before returning to the caller.
NO_WAIT	Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* = WAIT):

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

if (*func_wait* = NO_WAIT):

MSG_PENDING
BAD_PARAM

StiDecSetAimDuration

Description

When a long-range (LR) or high-visibility (HV) scanner is triggered either by a trigger pull or a START_DECODE command, this parameter sets the duration the aiming pattern is seen before a scan attempt begins. It does not apply to the aim signal or the AIM_ON command. It is programmable in 0.1 second increments from 0.0 to 9.9 seconds. No aim pattern is visible when the value is 0.0.

Syntax

int far StiDecSetAimDuration(*int aim_duration*)

where:

aim_duration is an integer value in the range [0..99], representing a time period from 0.0 to 9.9 seconds in 0.1 second increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetBeepAfterGoodDecode

Description

Enables or disables beeping after a good decode. The beeper still operates during parameter menu scanning and indicates error conditions.

Syntax

int far StiDecSetBeepAfterGoodDecode(*int beep*)

where:

beep is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetBeeperTone

Description

Sets the decode beep frequency (tone).

Syntax

int far StiDecSetBeeperTone(*int beeper_tone*)

where:

beeper_tone is one of the following values:

LOW_FREQUENCY
MEDIUM_FREQUENCY
HIGH_FREQUENCY

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetBidirectionalRedundancy

Description

This parameter is only valid when a Linear Code Type Security Level is enabled. When enabled, a bar code must be successfully scanned in both directions (forward and reverse) before being decoded.

Syntax

int far StiDecSetBidirectionalRedundancy(*int redundancy*)

where:

redundancy is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetBooklandEan

Description

Enables or disables Bookland EAN.

Syntax

int far StiDecSetBooklandEan(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetBootUpEvent

Description

Enables Boot Up Event option.

Syntax

int StiDecSetBootUpEvent(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Boot Up Event is enabled when the Boot Up Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetClsiEditing

Description

Strips the start and stop characters and inserts a space after the first, fifth, and tenth characters of a 14-character Codabar symbol.

NOTE Symbol length does not include start and stop characters.

Syntax

int far StiDecSetClsiEditing(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCodabar

Description

Enables or disables Codabar.

Syntax

int far StiDecSetCodabar(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCode128

Description

Enables or disables Code 128.

Syntax

int far StiDecSetCode128(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCode32Prefix

Description

Appends the character 'A' to the start of the decode data if enabled.

Syntax

int far StiDecSetCode32Prefix(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCode39

Description

Enables or disables Code 39.

Syntax

int far StiDecSetCode39(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCode39CheckDigit

Description

Checks the integrity of a Code 39 symbol to ensure it complies with specified algorithms. Only Code 39 symbols which include a modulo 43 check digit are decoded.

Syntax

int far StiDecSetCode39CheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetCode39FullAscii

Description

Enables or disables Code 39 Full ASCII.

When enabled, the ASCII character set assigns a code to letters, punctuation marks, numerals, and most control keystrokes on the keyboard.

The first 32 codes are non-printable and are assigned to keyboard control characters such as BACKSPACE and RETURN. The other 96 are called printable codes because all but SPACE and DELETE produce visible characters.

Code 39 Full ASCII interprets the bar code special character ($ + % /) preceding a Code 39 character and assigns an ASCII character value. For example, when Code 39 Full ASCII is enabled and a +B is scanned, it is interpreted as b, %J as 7, and $H emulates the keystroke BACKSPACE. Scanning ABC$M outputs the keystroke equivalent of ABC ENTER.

Do not enable Code 39 Full ASCII and Trioptic Code 39 simultaneously. The decoder does not autodiscriminate between Code 39 and Code 39 Full ASCII.

Syntax

int far StiDecSetCode39FullAscii(*int full_ascii*)

where:

full_ascii is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetCode93

Description

Enables or disables Code 93.

Syntax

int far StiDecSetCode93(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetConvertCode39toCode32

Description

Converts Code 39 to Code 32.

Syntax

int far StiDecSetConvertCode39toCode32(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetConvertEan8toEan13

Description

When EAN Zero Extend is enabled, labels the extended symbol as either an EAN-13 bar code, or an EAN-8 bar code.

When EAN Zero Extend is disabled, this parameter has no effect on bar code data.

Syntax

int far StiDecSetConvertEan8toEan13(*int enable*)

where:

enable is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetConvertI2of5toEan13

Description

Converts a 14-character I 2 of 5 code to EAN-13, and transmits to the host as EAN-13. To accomplish this, the I 2 of 5 code must be enabled, one length must be set to 14, and the code must have a leading zero and a valid EAN-13 check digit.

Syntax

int far StiDecSetConvertI2of5toEan13(*int enable*)

where:

enable is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetConvertUpcE1toUpcA

Description

Converts UPC-E1 (zero suppressed) decoded data to UPC-A format before transmission. After conversion, data follows UPC-A format and is affected by UPC-A programming selections (e.g., Preamble, Check Digit).

Selecting DISABLE allows you to transmit UPC-E1 (zero suppressed) decoded data

Syntax

int far StiDecSetConvertUpcE1toUpcA(*int enable*)

where:

enable is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetConvertUpcEtoUpcA

Description

Converts UPC-E (zero suppressed) decoded data to UPC-A format before transmission. After conversion, data follows UPC-A format and is affected by UPC-A programming selections (e.g., Preamble, Check Digit).

Selecting DISABLE allows you to transmit UPC-E (zero suppressed) decoded data.

Syntax

int far StiDecSetConvertUpcEtoUpcA(*int enable*)

where:

enable is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetD2of5

Description

Enables or disables Discrete 2 of 5.

Syntax

int far StiDecSetD2of5(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetDecodeEvent

Description

Enables Decode Event option.

Syntax

int StiDecSetDecodeEvent(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Decode Event is enabled when the Decode Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetDecodeUpcEanRedundancy

Description

When Autodiscriminate UPC/EAN Supplementals selected, adjusts the number of times a symbol without supplementals is decoded before transmission. The range is from 2 to 20 times. Five or above is recommended when decoding a mix of UPC/EAN symbols with and without supplementals, and the autodiscriminate option is selected.

Syntax

int far StiDecSetDecodeUpcEanRedundancy(*int supplemental_redundancy*)

where:

supplemental_redundancy is an integer value in the range [2..20].

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetDecodeUpcEanSupplementals

Description

Sets Decode UPC/EAN Supplementals option. Supplementals are additionally appended characters (2 or 5) according to specific code format conventions (e.g., UPC A+2, UPC E+2, EAN 8+2). Three options are available.

- If UPC/EAN with supplemental characters is selected, UPC/EAN symbols without supplemental characters are not decoded.
- If UPC/EAN without supplemental characters is selected, and the decoder is presented with a UPC/EAN plus supplemental symbol, the UPC/EAN is decoded and the supplemental characters ignored.
- An autodiscriminate option is also available.

Syntax

int far StiDecSetDecodeUpcEanSupplementals(*int supplementals*)

where:

supplementals is one of the following values:

DECODE_SUPPLEMENTALS

IGNORE_SUPPLEMENTALS

AUTODISCRIMINATE_SUPPLEMENTALS

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetEan13

Description

Enables or disables EAN-13.

Syntax

int far StiDecSetEan13(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetEan8

Description

Enables or disables EAN-8.

Syntax

int far StiDecSetEan8(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetEanZeroExtend

Description

When enabled, five leading zeros are added to decoded EAN-8 symbols to make them compatible in format to EAN-13 symbols.

Disabling this returns EAN-8 symbols to their normal format.

Syntax

int far StiDecSetEanZeroExtend(*int enable*)

where:

enable is one of the following values:

ENABLE

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetHostCharacterTimeOut

Description

Determines the maximum time the decoder waits between characters transmitted by the host before discarding the received data and declaring an error. The time-out is set in 0.01 second increments from 0.01 to 0.99 seconds.

Syntax

int far StiDecSetHostCharacterTimeOut(*int time_out*)

where:

time_out is an integer value in the range [1..99], representing a time period from 0.01 to 0.99 seconds in 0.01 second increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetHostSerialResponseTimeOut

Description

Specifies how long the decoder waits for an ACK, NAK or CTS before determining that a transmission error has occurred. The delay period can range from 0.0 to 9.9 seconds in .1 second increments.

Syntax

int far StiDecSetHostSerialResponseTimeOut(*int time_out*)

where:

time_out is an integer value in the range [0..99], representing a time period from 0.0 to 9.9 seconds in 0.1 increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetI2of5

Description

Enables or disables Interleaved 2 of 5.

Syntax

int far StiDecSetI2of5(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetI2of5CheckDigitVerification

Description

Checks the integrity of an I 2 of 5 symbol to ensure it complies with a specified algorithm, either USS (Uniform Symbology Specification), or OPCC (Optical Product Code Council).

Syntax

int far StiDecSetI2of5CheckDigitVerification(*int check_digit*)

where:

check_digit is one of the following values:

DISABLE
USS_CHECK_DIGIT
OPCC_CHECK_DIGIT

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetIntercharacterDelay

Description

Sets the intercharacter delay option to match host requirements. The intercharacter delay gives the host system time to service its receiver and perform other tasks between characters. The delay period can range from no delay to 99 msec in 1 msec increments.

Syntax

int far StiDecSetIntercharacterDelay(*int intercharacter_delay*)

where:

intercharacter_delay is an integer value in the range [0..99], representing a time period from 0 to 99 msec in 1 msec increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetIsbt128

Description

Enables or disables ISBT 128.

Syntax

int far StiDecSetIsbt128(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLaserOnTime

Description

Sets the maximum time decode processing continues during a scan attempt. It is programmable in 0.1 second increments from 0.5 to 9.9 seconds.

Syntax

int far StiDecSetLaserOnTime(*int laser_on_time*)

where:

laser_on_time is an integer value in the range [5..99], representing a time period from 0.5 to 9.9 seconds in 0.1 second increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetLengthsCodabar

Description

Sets lengths for Codabar. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters) the code contains. It also includes any start or stop characters.

- One Discrete Length - Decodes only those codes containing a selected length.
- Two Discrete Lengths - Decodes only those codes containing two selected lengths.
- Length Within Range - Decodes a code type within a specified range.
- Any length - Decodes Codabar symbols containing any number of characters.

Syntax

int far StiDecSetLengthsCodabar(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLengthsCode39

Description

Sets lengths for Code 39. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains. If Code 39 Full ASCII is enabled, Length Within a Range or Any Length are the preferred options.

- One Discrete Length - decodes only those codes containing a selected length.
- Two Discrete Lengths - decodes only those codes containing two selected lengths.
- Length Within Range - decodes a code type within a specified range.
- Any Length - decodes Code 39 symbols containing any number of characters.

Syntax

int far StiDecSetLengthsCode39(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLengthsCode93

Description

Sets lengths for Code 93. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters), including check digit(s) the code contains.

- One Discrete Length - Decodes only those codes containing a selected length.
- Two Discrete Lengths - Decodes only those codes containing two selected lengths.
- Length Within Range - Decodes a code type within a specified range.
- Any Length - Decodes Code 93 symbols containing any number of characters.

Syntax

int far StiDecSetLengthsCode93(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLengthsD2of5

Description

Sets lengths for D 2 of 5. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters) the code contains, and includes check digits.

- One Discrete Length - Decodes only those codes containing a selected length.
- Two Discrete Lengths - Decodes only those codes containing two selected lengths.
- Length Within Range - Decodes a code type within a specified range.
- Any Length - Decodes D 2 of 5 symbols containing any number of characters.

NOTE Selecting this option may lead to misdecodes for D 2 of 5 codes.

Syntax

int far StiDecSetLengthsD2of5(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLengthsI2of5

Description

Sets lengths for I 2 of 5. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters) the code contains, and includes check digits.

- One Discrete Length - Decodes only those codes containing a selected length.
- Two Discrete Lengths - Decodes only those codes containing two selected lengths.
- Length Within Range - Decodes a code type within a specified range.
- Any Length - Decodes I 2 of 5 symbols containing any number of characters.

NOTE Selecting this option may lead to misdecodes for I 2 of 5 codes.

Syntax

int far StiDecSetLengthsI2of5(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLengthsMsiPlessey

Description

Sets lengths for MSI Plessey. Options are any length, one or two discrete lengths, or lengths within a specific range. The length of a code refers to the number of characters (i.e., human readable characters) the code contains, and includes check digits.

- One Discrete Length - Decodes only those codes containing a selected length.
- Two Discrete Lengths - Decodes only those codes containing two selected lengths.
- Length Within Range - Decodes a code type within a specified range.
- Any Length - Decodes MSI Plessey symbols containing any number of characters.

NOTE Selecting this option may lead to misdecodes for codes.

Syntax

int far StiDecSetLengthsMsiPlessey(*int length_type, int length1, int length2*)

where:

length_type is one of the following values:

ONE_DISCRETE_LENGTH
TWO_DISCRETE_LENGTHS
LENGTH_WITHIN_RANGE
ANY_LENGTH

length1 and *length2* describe the discrete length(s) or the specific range of values.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetLinearCodeTypeSecurityLevel

Description

Four levels of decode security for linear code types (e.g. Code 39, Interleaved 2 of 5) are supported. Select a higher security level for low levels of bar code quality. As security levels increase, the decoder's aggressiveness decreases.

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

Code Type	Length
Codabar	All
MSI Plessey	4 or less
D2of 5	8 or less
12 of 5	8 or less

- Linear Security Level 2: All code types must be successfully read twice before being decoded.
- Linear Security Level 3: Code types other than the following must be successfully read twice before being decoded. The following codes must be read three times:

Code Type	Length
MSI Plessey	4 or less
D2of 5	8 or less
I2of 5	8 or less

- Linear Security Level 4: All code types must be successfully read three times before being decoded.

Syntax

int far StiDecSetLinearCodeTypeSecurityLevel(*int security_level*)

where:

security_level is one of the following values:

SECURITY_LEVEL1

SECURITY_LEVEL2

SECURITY_LEVEL3

SECURITY_LEVEL4

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetMiscellaneousEvent1

Description

Enables Miscellaneous Event 1 option.

Syntax

int StiDecSetMiscellaneousEvent1(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Miscellaneous Event 1 is enabled when the Miscellaneous Event 1 occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetMiscellaneousEvent2

Description

Enables Miscellaneous Event 2 option.

Syntax

int StiDecSetMiscellaneousEvent2(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Miscellaneous Event 2 is enabled when the Miscellaneous Event 2 occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetMsiPlessey

Description

Enables or disables MSI Plessey.

Syntax

int far StiDecSetMsiPlessey(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetMsiPlesseyCheckDigitAlgorithm

Description

When Two MSI Plessey check digits is selected, sets an additional verification is required to ensure integrity. Two following algorithms may be selected: MOD10/MOD11 or MOD10/MOD10.

Syntax

int far StiDecSetMsiPlesseyCheckDigitAlgorithm(*int algorithm*)

where:

algorithm is one of the following values:

MOD10_MOD11
MOD10_MOD10

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetMsiPlesseyCheckDigits

Description

Sets the number of check digits at the end of the bar code that verify the integrity of the data. At least one check digit is always required. Check digits are not automatically transmitted with the data.

Syntax

int far StiDecSetMsiPlesseyCheckDigits(*int check_digits*)

where:

check_digits is one of the following values:

ONE_CHECK_DIGIT
TWO_CHECK_DIGITS

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetNotisEditing

Description

Strips the start and stop characters from decoded Codabar symbol.

Syntax

int far StiDecSetNotisEditing(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetParameterEvent

Description

Enables Parameter Event option.

Syntax

int StiDecSetParameterEvent(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Parameter Event is enabled when the Parameter Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetParameterScanning

Description

Enables or disables decoding of parameter bar codes. Note that the Set Defaults and Enable Parameter Scanning parameter bar codes can still be decoded.

Syntax

int far StiDecSetParameterScanning(*int parameter_scanning*)

where:

parameter_scanning is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetPowerMode

Description

Determines whether power remains on when the decoder is not busy. In low power mode, the decoder enters a low power consumption mode whenever possible. In continuous power mode, the decoder does not automatically enter the low power consumption mode.

Syntax

int far StiDecSetPowerMode(*int power_mode*)

where:

power_mode is one of the following values:

CONTINUOUS_ON
LOW_POWER

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetPrefixSuffixValues

Description

Appends a prefix and/or one or two suffixes to scan data for use in data editing.

NOTE In order to use Prefix/Suffix values, the Scan Data Transmission Format must be set.

Syntax

int far StiDecSetPrefixSuffixValues(*int prefix, int suffix_1, int suffix_2*)

where:

prefix, suffix_1 and *suffix_2* are the ASCII values desired.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetScanDataTransmissionFormat

Description

Sets the Scan Data Transmission Format.

Syntax

int far StiDecSetScanDataTransmissionFormat(*int transmission_format*)

where:

transmission_format is one of the following values:

DATA_AS_IS
DATA_SUFFIX_1
DATA_SUFFIX _2
DATA_SUFFIX_1_SUFFIX_2
PREFIX_DATA
PREFIX_DATA_SUFFIX_1
PREFIX_DATA_SUFFIX_2
PREFIX_DATA_SUFFIX_1_SUFFIX_2

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetScanningError

Description

Enables Scanning Error option.

Syntax

int StiDecSetScanningError(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the Scanning Event is enabled when the Scanning Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetSystemError

Description

Enables System Error option.

Syntax

int StiDecSetSystemError(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the System Event is enabled when the System Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetSystemEvent

Description

Enables System Event option.

Syntax

int StiDecSetSystemEvent(*int enable, void far (*eventFunction)(void)*)

where:

enable is one of the following values:

ENABLE

DISABLE

eventFunction is a pointer to a function which will be called if the System Event is enabled when the System Event occurs.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetTimeOutBetweenSameSymbol

Description

In Continuous triggering mode, sets the minimum time that must elapse before the decoder decodes a second bar code which is identical to the first decoded. This reduces the risk of accidentally scanning the same symbol twice. It is programmable in .1 second increments from 0.0 to 9.9 seconds.

Syntax

int far StiDecSetTimeOutBetweenSameSymbol(*int time_out*)

where:

time_out is an integer value in the range [0..99], representing a time period from 0.0 to 9.9 seconds in 0.1 second increments.

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetTransmitCode39CheckDigit

Description

Enables or disables Code 39 Check Digit.

Syntax

int far StiDecSetTransmitCode39CheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitCodeIdCharacter

Description

Transmits Code ID character which identifies the code type of a scanned bar code. This may be useful when the decoder is decoding more than one code type. In addition to any single character prefix already selected, the code ID character is inserted between the prefix and the decoded symbol.

The user may select no code ID character, a Symbol Code ID character, or an AIM Code ID character. The Symbol Code ID characters are listed below.

A = UPC-A, UPC-E, UPC-E1, EAN-8, EAN-13

B = Code 39, Code 32

C = Codabar

D = Code 128, ISBT 128

E = Code 93

F = Interleaved 2 of 5

G = Discrete 2 of 5, or Discrete 2 of 5 IATA

J = MSI Plessey

K = UCC/EAN-128

L = Bookland EAN

M = Trioptic Code 39

N = Coupon Code

Syntax

int far StiDecSetTransmitCodeIdCharacter(*int code_id*)

where:

code_id is one of the following values:

SYMBOL_CODE_ID_CHARACTER

AIM_CODE_ID_CHARACTER

DISABLE

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetTransmitI2of5CheckDigit

Description

Enables or disables the I 2 of 5 check digit.

Syntax

int far StiDecSetTransmitI2of5CheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitMsiPlesseyCheckDigit

Description

Enables or disables transmission of MSI Plessey Check Digit.

Syntax

int far StiDecSetTransmitMsiPlesseyCheckDigit(*int check_digits*)

where:

check_digits is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitNoReadMessage

Description

When enabled, the decoder transmits "NR" if a symbol does not decode. Any prefix or suffixes enabled are appended around this message.

Syntax

int far StiDecSetTransmitNoReadMessage(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitUpcACheckDigit

Description

Transmits the symbol with or without the UPC-A check digit.

Syntax

int far StiDecSetTransmitUpcACheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitUpcE1CheckDigit

Description

Transmits the symbol with or without the UPC-E1 check digit.

Syntax

int far StiDecSetTransmitUpcE1CheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTransmitUpcECheckDigit

Description

Transmits the symbol with or without the UPC-E check digit.

Syntax

int far StiDecSetTransmitUpcECheckDigit(*int check_digit*)

where:

check_digit is one of the following values:

TRANSMIT_CHECK_DIGIT
DO_NOT_TRANSMIT_CHECK_DIGIT

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetTriggeringModes

Description

Triggers the decoder.

Syntax

int far StiDecSetTriggeringModes(*int triggering_mode*)

where:

triggering _mode is one of the following values:

LEVEL

PULSE

CONTINUOUS

BLINKING

HOST

Return Values

STATUS_OK

BAD_PARAM

BATCH_ERROR

StiDecSetTriopticCode39

Description

Enables or disables Trioptic Code 39. Trioptic Code 39 symbols always contain six characters. Do not enable Trioptic Code 39 and Code 39 Full ASCII simultaneously.

Syntax

int far StiDecSetTriopticCode39(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUccEan128

Description

Enables or disables UCC/EAN-128.

Syntax

int far StiDecSetUccEan128(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcA

Description

Enables or disables UPC-A.

Syntax

int far StiDecSetUpcA(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcAPreamble

Description

Sets the UPC-A Preamble option. Three options are given for lead-in characters for UPC-A symbols transmitted to the host device: transmit system character only, transmit system character and country code ("0" for USA), and no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecSetUpcAPreamble(*int preamble*)

where:

preamble is one of the following values:

NO_PREAMBLE
SYSTEM_CHARACTER
SYSTEM_CHARACTER_COUNTRY_CODE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcE

Description

Enables or disables UPC-E.

Syntax

int far StiDecSetUpcE(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcE1

Description

Enables or disables UPC-E1.

Syntax

int far StiDecSetUpcE1(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcE1Preamble

Description

Sets the UPC-E1 Preamble option. Three options are given for lead-in characters for UPC-E1 symbols transmitted to the host device: transmit system character only, transmit system character and country code (“0” for USA), and no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecSetUpcE1Preamble(*int preamble*)

where:

preamble is one of the following values:

NO_PREAMBLE
SYSTEM_CHARACTER
SYSTEM_CHARACTER_COUNTRY_CODE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcEanCouponCode

Description

When enabled, decodes UPC-A, UPC-A with 2 supplemental characters, UPC-A with 5 supplemental characters, and UPC-A/EAN128 bar codes. UPC-A with supplemental characters need not be enabled.

Syntax

int far StiDecSetUpcEanCouponCode(*int enable*)

where:

enable is one of the following values:

ENABLE
DISABLE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcEanSecurityLevel

Description

Sets the UPC/EAN Security Level. There are four levels of decode security for UPC/EAN bar codes. Select a higher level of security are provided for decreasing levels of bar code quality. There is an inverse relationship between security and decoder aggressiveness, so be sure to choose only that level of security necessary for any given application.

- UPC/EAN Security Level 0: This default setting allows the decoder to operate in its most aggressive state, while providing sufficient security in decoding “in-spec” UPC/EAN bar codes.
- UPC/EAN Security Level 1: As bar code quality levels diminish, certain characters become prone to misdecodes before others (i.e., 1, 2, 7, 8). If you are experiencing misdecodes of poorly printed bar codes, and the misdecodes are limited to these characters, select this security level.
- UPC/EAN Security Level 2: If you are experiencing misdecodes of poorly printed bar codes, and the misdecodes are not limited to characters 1, 2, 7, and 8, select this security level.
- UPCIEAN Security Level 3: If you have tried Security Level 2, and are still experiencing misdecodes, select this security level. Be advised, selecting this option is an extreme measure against mis-decoding severely out of spec bar codes. Selection of this level of security significantly impairs the decoding ability of the decoder. If this level of security is necessary you should try to improve the quality of your bar codes.

Syntax

int far StiDecSetUpcEanSecurityLevel(*int security_level*)

where:

security_level is one of the following values:

SECURITY_LEVEL0
SECURITY_LEVEL1
SECURITY_LEVEL2
SECURITY_LEVEL3

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiDecSetUpcEPreamble

Description

Sets the UPC-E Preamble option. Three options are given for lead-in characters for UPC-E symbols transmitted to the host device: transmit system character only, transmit system character and country code ("0" for USA), and no preamble transmitted. The lead-in characters are considered part of the symbol.

Syntax

int far StiDecSetUpcEPreamble(*int preamble*)

where:

preamble is one of the following values:

NO_PREAMBLE
SYSTEM_CHARACTER
SYSTEM_CHARACTER_COUNTRY_CODE

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

Chapter 4 Communications API

Introduction

The SSI-SDK Serial Comm Driver is platform-specific software that communicates between the serial port on the host, or the serial port driver code in the target operating system, and the platform-independent SSI Kernel code. This driver's functions use the **StiComm_** prefix.

This code contains the functions required to:

- Initialize the serial port; read and control the appropriate modem control lines (i.e., RTS and CTS)
- Send a buffer with completion notification and error checking
- Receive a buffer with completion notification and error checking
- Provide low-level hardware and software handshaking support.

The Serial Comm driver also provides the necessary OS-specific Timer functions required to time communications events in the SSI Kernel.

Return Value Definitions

The functions in this chapter may return the following standard status codes:

- Any non-negative value (0 to 32767): parameter value.
- STATUS_OK: the function parameters were verified. If a function must wait for an ACK from the decoder, STATUS_OK indicates the ACK was received.
- MSG_PENDING: a message is pending in the transmit buffer. No further messages can be placed in the buffer in this condition; the program must wait until there is room for a new message in the transmit buffer.
- NOT_SUPPORTED: the last packet received from the decoder was either a NAK_DENIED or NAK_BAD_CONTEXT. When retrieving parameters from the decoder, this indicates that the specified parameter is not supported by this decoder, or that it was unable to comply with the request.
- COMMUNICATIONS_ERROR: either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not acknowledged, and therefore is questionable.
- BAD_PARAM: the user-supplied parameter(s) in the function call was not in the expected range.
- BATCH_ERROR: the limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with **StiDecSet**_ are responsible for generating a batch command to establish decoder parameters. The parameters are not sent to the decoder until the **StiDecSendParams()** function is called, when a new batch is started.
- ERROR_UNDEFINED: an error condition not specifically associated with the decoder or its communications.

Initialize Communications Port

The SSI-SDK uses a commercially available, royalty-free serial communications library for Personal Computers. Functions referred to by the **Sio**_ prefix are provided by the Serial
I/O Library. The library of functions is identical under DOS and Windows and provides direct access to the serial communications hardware.

SioBaud

Description

Sets the baud rate without resetting the port. It is used to change the baud rate after calling **SioReset**.

Syntax

int SioBaud(*int Port,int BaudCode*)

Code	Rate	Name	Code	Rate	Name
0	300	Baud300	5	9600	Baud9600
1	600	Baud600	6	19200	Baud19200
2	1200	Baud1200	7	38400	Baud38400
3	2400	Baud2400	8	57600	Baud57600
4	4800	Baud4800	9	115200	Baud115200

where:

Port: Port selected (COM1 - COM20)

BaudCode: Baud code

Return Values

-4 : No such port. Expect 0 to MaxPort.

-11 : Bad baud rate code.

SioBrkSig

Description

Controls the BREAK bit in the line status register. The legal commands are:

ASSERT_BREAK ('A') to assert BREAK

CANCEL_BREAK ('C') to cancel BREAK

DETECT_BREAK ('D') to detect BREAK

Syntax

int SioBrkSig(*int Port,char Cmd*)

Port: Port selected (COM1 thru COM20)

char Cmd: ASSERT, CANCEL, or DETECT

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

-6 : Illegal command. Expected 'A', 'C', or 'D'.

>0 : BREAK signal detected (DETECT command only)

Notes

- The Serial Comm driver also provides the necessary OS-specific Timer functions required to time communications events in the SSI Kernel.
- The periodic tic timer (18.6 Hz) is used for all timing. All required time functions are implemented within the Serial Comm driver functions.

SioCTS

Description

Reads the Clear to Send (CTS) modem status bit.

The CTS line is used by some error correcting modems to implement hardware flow control. The modem drops CTS to signal the computer not to send data and raises CTS to signal the computer to continue.

Syntax

int SioCTS(*int Port*)

where:

int Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

0 : CTS is clear.

>0 : CTS is set.

Note

- Send a buffer with completion notification and error checking.

SioDone

Description

Terminates further serial processing. You MUST call **SioDone** before exiting your application so that interrupts can be restored to their original state. Failure to do this can crash the operating system. If you do not call **SioDone** before exiting, re-boot your computer. You may call **SioDone** even if **SioReset** has not been called.

SioDone dereferences the transmit and receive buffers set up by **SioRxQue** and **SioTxQue**.

Syntax

int SioDone(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

SioFIFO

Description

Enables both transmit and receive FIFOs for 16550 UARTS, and sets the trigger level at which receive interrupts are generated.

For example, if the FIFO level is set to 8, the 16550 UART does not generate an interrupt until 8 bytes are received. This reduces the number of interrupts generated and allows faster processing with slower machines or when running simultaneous ports.

To test if your port is a 16550 UART, call **SioFIFO** with a LevelCode of other than FIFO_OFF. **SioFIFO** can be called for the 8250 and 16450 UART without negative effect.

Code Name	Trigger Level
-1: FIFO_OFF	Disable FIFO
0: LEVEL_1	1 byte
1: LEVEL_4	4 bytes
2: LEVEL_8	8 bytes
3: LEVEL_14	14 bytes

Syntax

int SioFIFO(*int Port,int LevelCode*)

where:

Port: Port selected (COM1 thru COM20)

LevelCode: FIFO level code (see below)

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

>0 : FIFO level set.

0 : FIFO level not set (not 16550).

SioFlow

Description

Enables or disables hardware flow control, which uses RTS and CTS to control data flow between the modem and the computer.

Tics is the number of timer tics before **SioPutc** returns timeout on transmission because CTS is down. To disable flow control, call **SioFlow** with Tics = -1.

For hardware flow control to work correctly, your modem must also be configured to work with hardware flow control, and your computer-to-modem cable must have RTS and CTS wired straight through. If you enable hardware flow control, do not modify the RTS line (by calling **SioRTS**) unless you are an experienced user.

Flow control is disabled after resetting a port. To explicitly disable hardware flow control, call **SioFlow** with Flag=FALSE.

Syntax

int SioFlow(*int Port,int Tics*)

where:

Port: Port selected (COM1 thru COM20)

Tics: # tics before TX timeout

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

=0 : Flow control disabled.

>0 : Flow control enabled.

SioGetc

Description

Reads a byte from the selected serial port and waits for the number of system tics given by the Tics argument before returning “timed out”. There are 18.2 tics per second. Try to limit Tics since no other Windows task can run while waiting for input.

Syntax

int SioGetc(*int Port, int Tics*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

-1 : If timed out.

>0 : Character read.

SioGetDiv

Description

Reads the baud rate divisor. The baud rate can then be determined by the divisor.

Syntax

int SioGetDiv(*int Port*)

Baud	Divisor	Baud	Divisor	Baud	Divisor
300	0180	4800	0018	38400	0003
1200	0060	9600	000C	57600	0002
2400	0030	19200	0006	115200	0001

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

>0 : Baud rate divisor.

SioGets

Description

Reads bytes from the selected serial port. This function reads either the number of requested bytes, or fewer if no character is immediately available. **SioGets** cannot return more characters than are in the receive buffer at the time it is called. The number of characters read is returned.

Syntax

int SioGets(*int Port,char *Buffer, int Length*)

where:

Port: Port selected (COM1 thru COM20)

Buffer: Character buffer

Length: Length of Buffer (# of requested bytes)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

-1 : If timed out.

>0 : Number of characters read into Buffer.

SioLine

Description

NOTE This function has not been used in the provided code as the transmitter empty bit does not appear to set correctly. Instead, the hardware is read directly using SioRead (int Port, int Reg) where Reg. is 5 (the Line Status Register). The bit masks are identical to those below.

Reads the line status register. The individual bit masks are:

- 0x40 = Transmitter empty.
- 0x20 = Transmitter buffer empty.
- 0x10 = Break detected.
- 0x08 = Framing error.
- 0x04 = Parity error.
- 0x02 = Overrun error.
- 0x01 = Data ready.

Syntax

int SioLine(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

>0 : Line status (rightmost byte of word).

SioModem

Description

Reads the modem register. The bit definitions for the function mask are as follows:

Bit	Name	Function
7	DCD	Data Carrier Detect
6	RI	Ring Indicator
5	DSR	Data Set Ready
4	CTS	Clear To Send
3	DeltaDCD	DCD has changed
2	DeltaRI	RI has changed
1	DeltaDSR	DSR has changed
0	DeltaCTS	CTS has changed

Bits 4 through 7 represent the absolute state of their respective RS-232 inputs. Bits 0 through 3 represent a change in the state of their respective RS-232 inputs since last read. Once UART register 3 is read, the Delta bits are cleared, so do not depend on the Delta bits.

Syntax

int SioModem(*int Port, int Mask*)

where:

Port: Port selected (COM1 thru COM20)

Mask: Modem function mask

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

>0 : Modem status (rightmost byte of word).

SioParms

Description

Sets the parity, stop bits, and word length. If the default parity (none), stop bits (1), or word length (8) is not acceptable, change them by calling SioParms, either before or after calling SioReset.

	Value	Description	Name
ParityCode	*0	no parity	NoParity
	1	odd parity	OddParity
	3	even parity	EvenParity
StopBitsCode	*0	1 stop bit	OneStopBit
	1	2 stop bits	TwoStopBits
WordLengthCode	0	5 data bits	WordLength5
	1	6 data bits	WordLength6
	2	7 data bits	WordLength7
	*3	8 data bits	WordLength8

Syntax

int SioParms(*int Port,int Parity,int StopBits, int DataBits*)

where:

Port: Port selected (COM1 - COM20)

Parity: Parity code [0,1,2]

StopBits: Stop bits code [0,1]

DataBits: Word length code [0,1,2,3]

Return Values

-4 : No such port. Expect 0 to MaxPort.

-7 : Bad parity code selected. Expecting 0 to 2.

-8 : Bad stop bits code. Expecting 0 or 1.

-9 : Bad word length code. Expecting 0 to 3.

SioPorts

Description

Call the **SioPorts** function before ANY other serial functions. **SioPorts** function sets the total number of ports, the first DigiBoard (or BOCA board) port and the DigiBoard (or BOCA board) status register address. Once **SioPorts** is called, all COM ports starting with “FirstPort” are treated as DigiBoard (or BOCA board) ports. The default setup is 4 standard PC ports and no DigiBoard or BOCA ports [SioPorts(4,4,0,PC_PORTS)].

Syntax

int SioPorts(*int NbrPorts,int FirstPort,int StatusReg, int Code*)

where:

NbrPorts: Total number of ports

FirstPort: First DigiBoard port

StatusReg: DigiBoard Status Register

Code: PC_PORTS, DIGIBOARD, or BOCABOARD

Return Values

-4 : No such port. Expect 0 to 9.

0 : No error (sets MaxPort to NumberPorts-1).

SioPutc

Description

Transmits one character over the selected serial line.

If flow control is enabled, **SioPutc** may return a -1 (time out) if the number of tics specified in the **SioFlow** function was exceeded waiting for the modem to raise CTS.

If transmitter interrupts are enabled (there are separate versions of the library for transmitter interrupts enabled), the byte is placed in the transmit buffer and waits for transmission by the interrupt service routine.

Syntax

int SioPutc(*int Port,char Ch*)

where:

Port: Port selected (COM1 thru COM20)

Ch: Character to send

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

-1 : Timed out waiting for CTS (flow control enabled)

SioPuts

Description

Transmits each buffer character over the selected serial line, and returns the number of characters transmitted. This function can take a long time for large buffers if transmitter interrupts are not enabled!

If flow control is enabled, **SioPuts** may transmit fewer characters than requested if the number of tics specified in the **SioFlow** function was exceeded waiting for the modem to raise CTS.

If transmitter interrupts are enabled, **SioPuts** transmits fewer than the number of characters requested if the transmit buffer would overflow. The actual number of characters transmitted is returned.

Syntax

int SioPuts(*int Port,char *Buffer,int Length*)

where:

Port: Port selected (COM1 thru COM20)

Buffer: Character buffer

Length: Length of Buffer

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

>=0: Number of characters actually transmitted.

SioRead

Description

Reads the contents of any of the 7 UART registers. This function is useful when debugging application programs and as a method for verifying UART contents.

Syntax

int SioRead(*int Port,int Reg*)

where:

Port: Port selected (COM1 thru COM20)

Reg: UART register (0 to 7)

Return Values

-3 : No buffer available. Call **SioRxBuf** first.

-4 : No such port. Expect 0 to MaxPort.

SioReset

Description

Initializes the selected serial port. Call **SioReset** after calling **SioParm** and **SioRxBuf** but before making any other calls to the serial library. **SioReset** uses the parity, stop bits, and word length value previously set if **SioParm** was called, otherwise the default values (see **SioParm**) are used.

COM1 and COM3 share the same interrupt vector, so cannot operate simultaneously. Similarly, COM2 and COM4 cannot operate simultaneously. Any other combination of two ports can be used.

By specifying NORESET (-1) for the baud rate code, the port will NOT be reset. This is used to “take over” a port from a host communications program that allows a “DOS gateway”. External protocols can be implemented this way. See **SioBaud** for a list of the baud rate codes.

Syntax

int SioReset(*int Port,int BaudCode*)

where:

Port: Port selected (COM1 thru COM20)

BaudCode: Baud code or -1

Return Values

-3 : No buffer available. Call **SioRxBuf** first.

-4 : No such port. Expect 0 to MaxPort.

-11 : Bad baud rate code selected. Expecting 0 to 9.

-13 : UART undefined. **SioUART**(Port,0) was called previously.

-14 : Bad or missing UART. You may not have hardware present.

-15 : Port already enabled. **SioReset** has already been called.

-16 : Interrupt already in use.

SioRTS

Description

Controls the Request to Send (RTS) bit in the modem control register.

The RTS line is used by some error correcting modems to implement hardware flow control. The host drops RTS to signal the modem not to send data, and raise RTS to signal the modem to continue. Set RTS when communicating with a modem, unless you are using Flow Control.

SET_LINE ('S') - set RTS (ON)

CLEAR_LINE ('C') | clear RTS (OFF)

READ_LINE ('R') - read RTS

Syntax

int SioRTS(*Port, char Cmd*)

where:

Port: Port selected (COM1 thru COM20)

Cmd: RTS command (SET, CLEAR, or READ)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

-5 : Command is not one of 'S', 'C', or 'R'.

0 : RTS is OFF (READ_LINE Command).

>0 : RTS is ON (READ_LINE Command).

SioRxBuf

Description

Sets the address selector and size of the receive buffer. A receive buffer is required for each port in simultaneous operation since the receive function is interrupt driven. It must be called before any incoming characters can be received. **SioRxBuf** must be called before **SioReset**.

Size Code	Buffer Size	Name
4	128 bytes	Size128
5	256 bytes	Size256
6	512 bytes	Size512
7	1024 bytes	Size1024 or Size1K
8	2048 bytes	Size2048 or Size2K
9	4096 bytes	Size4096 or Size4K
10	8192 bytes	Size8192 or Size8K
11	16384 bytes	Size16384 or Size16K
12	32768 bytes	Size32768 or Size32K

Syntax

int SioRxBuf(*int Port,int Selector,SizeCode*)

where:

Port: Port selected (COM1 thru COM20)

Selector: Receive buffer selector

SizeCode: Buffer size code

Return Values

-4 : No such port. Expect 0 to MaxPort.

-10 : Bad buffer size code. Expecting 0 to 11.

SioRxClear

Description

Deletes any characters in the receive buffer for the specified port. After execution, the receive buffer is empty. Call **SioRxBuf** after resetting a port to delete any spurious characters.

Syntax

int SioRxClear(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

SioRxQue

Description

Returns the number of characters in the receive queue at the time of the call. It can be used to implement XON/XOFF flow control.

Syntax

int SioRxQue(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

SioTxBuf

Description

Assigns the address selector and size of the transmit buffer for each port in simultaneous operation. **SioTxBuf**() must be called before **SioReset**.

Size Code	Buffer Size	Name
4	128 bytes	Size128
5	256 bytes	Size256
6	512 bytes	Size512
7	1024 bytes	Size1024 or Size1K
8	2048 bytes	Size2048 or Size2K
9	4096 bytes	Size4096 or Size4K
10	8192 bytes	Size8192 or Size8K
11	16384 bytes	Size16384 or Size16K
12	32768 bytes	Size32768 or Size32K

This function is not used unless the transmitter interrupts are enabled.

Syntax

int SioTxBuf(*int Port,int Selector,int SizeCode*)

where:

Port: Port selected (COM1 thru COM20)

Selector: Receive buffer selector

SizeCode: Buffer size code

Return Values

-4 : No such port. Expect 0 to MaxPort.

-10 : Bad buffer size code. Expecting 0 to 11.

SioTxClear

Description

Deletes any characters in the transmit buffer for the specified port. After execution, the transmit buffer is empty. Once this function is called, any character in the transmit buffer (put there by **SioPutc**) is lost and not transmitted. This function is not used unless the transmitter interrupts are enabled.

Syntax

int SioTxClear(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

SioTxFlush

Description

Re-enables transmitter interrupts if they were disabled by CTS dropping. This forces all data out of the transmitter queue unless CTS drops again.

Once the transmitter interrupt is disabled (by CTS dropping), you may restart it by calling one of the serial I/O functions (**SioPutc**, **SioPuts**, **SioGetc**, or **SioGets**), **SioTxQue**, or **SioTxFlush**.

The main purpose of **SioTxFlush** is to provide an explicit means of restarting transmitter interrupts.

Syntax

int SioTxFlush(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call SioReset first.

-4 : No such port. Expect 0 to MaxPort.

SioTxQue

Description

Returns the number of characters in the transmit queue at the time of the call. This function is only used if the transmitter interrupts are enabled.

Syntax

int SioTxQue(*int Port*)

where:

Port: Port selected (COM1 thru COM20)

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

SioUnGetc

Description

Returns ("pushes") the character back into the serial input buffer. The character pushed is the next character returned by **SioGetc**. Only one character can be pushed back. This function works like the "ungetc" function in C language.

Syntax

int SioUnGetc(*int Port, char Ch*)

where:

Port: Port selected (COM1 thru COM20)

Ch: Character to unget

Return Values

-2 : Port not enabled. Call **SioReset** first.

-4 : No such port. Expect 0 to MaxPort.

StiCommChangeDecoderSettings

Description

Commands the decoder to change its baud rate, parity, and stop bits to a new setting. After the command is acknowledged by the decoder, this function changes the host communications settings to match those of the decoder. If the command is not acknowledged, no changes are made to the host communication settings.

Syntax

int far StiCommChangeDecoderSettings(*COMM_STRUCT comm, int change_type*)

where:

comm is a structure specifying all communications parameters for connection with the decoder.

change_type is:

PERMANENT
TEMPORARY

Return Values

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR
BAD_PARAM

StiCommInitPort

Description

Sets up serial communications parameters such as baud rate, parity, buffering, and handshaking on the host. This function does not change the communications settings for the decoder.

Syntax

int far StiCommInitPort(*COMM_STRUCT comm*)

where:

comm is a structure specifying all of the communications parameters for connection with the decoder.

Return Values

STATUS_OK

BAD_PARAM

NOT_SUPPORTED

SticommRxInterrupt

Description

Receive Interrupt Service Routine. Will be implemented in the serial I/O library functionality.

Syntax

void interrupt StiCommRxInterrupt( )

Note

Provides low-level hardware and software handshaking support.

StiCommTxInterrupt

Description

Transmit Interrupt Service Routine. Will be implemented in the serial I/O library functionality.

Syntax

void interrupt StiCommTxInterrupt()

Note

- Receive a buffer with completion notification and error checking.

Chapter 5 Kernel API

Introduction

The SSI-SDK Kernel (functions denoted by the StiSsi_ prefix) performs all functions required to maintain the SSI between the host and SSI-compliant decoder, including:

- Creation of message packets destined for the decoder
- Interpretation of message packets from the decoder
- Assembly of multi-packet messages from the decoder
- High-level implementation of interface handshaking modes
- Appropriate communications to the Host application, through the Host API function set, of interface events and data.

The Kernel also can detect and recover from interface error conditions, such as bad packets (invalid checksum, bad format, unexpected format, out of context messages, communications event timeouts, and protocol violations).

The Kernel uses the periodic tic timer interrupt (18.2 Hz) that typically runs on a PC. After every timer tic, the Kernel executes code dealing with the following aspects:

- It determines if a complete message was received and is in the queue.
- It deals with any handshaking that must be performed. If the RTS line was shut down because the receive queue was full, it monitors the receive queue in an attempt to restore the RTS line to its operational state.
- If a message packet is received which indicates that a specific event has occurred, the Kernel sees if the event is associated with a “callback” function. If the callback function is defined, it executes the function. Otherwise, it keeps a count of how many times the event has occurred.
- If a NAK is received, the Kernel retries the operation as defined in Chapter 2, Introduction to SSI. If an invalid packet is received, the Kernel retries the message responsible for the error.
- If an ACK is received unexpectedly, the ACK packet is ignored (discarded). Unexpected ACK packets are not considered errors.

Return Values Definitions

The functions in this chapter may return the following standard status codes:

- Any non-negative value (0 to 32767): parameter value.
- STATUS_OK: the function parameters were verified. If a function must wait for an ACK from the decoder, STATUS_OK indicates the ACK was received.
- MSG_PENDING: a message is pending in the transmit buffer. No further messages can be placed in the buffer in this condition; the program must wait until there is room for a new message in the transmit buffer.
- NOT_SUPPORTED: the last packet received from the decoder was either a NAK_DENIED or NAK_BAD_CONTEXT. When retrieving parameters from the decoder, this indicates that the specified parameter is not supported by this decoder, or that it was unable to comply with the request.
- COMMUNICATIONS_ERROR: either a timeout condition or the maximum number of retries (or both) occurred. The previous transmit message was not acknowledged, and therefore is questionable.
- BAD_PARAM: the user-supplied parameter(s) in the function call was not in the expected range.
- BATCH_ERROR: the limits of a batch function have been exceeded. Unless otherwise indicated, functions that start with **StiDecSet**_ are responsible for generating a batch command to establish decoder parameters. The parameters are not sent to the decoder until the **StiDecSendParams()** function is called, when a new batch is started.
- ERROR_UNDEFINED: an error condition not specifically associated with the decoder or its communications.

PARAM_REQUEST

Description

Returns the current decoder value for the requested parameter.

Syntax

int far StiSsiParamRequest(*int param*)

where:

param specifies a single, specific parameter stored in non-volatile memory in the decoder.

Return Values

Returns the current decoder value for the requested parameter. If an error occurs, the returned error status is one of the following:

NOT_SUPPORTED

COMMUNICATION_ERROR

BAD_PARAM

PARAM_REQUEST MULTIPLE

Description

Returns the current decoder value for the requested parameters.

Syntax

int far StiSsiParamRequestMultiple(*int far param[ ], int length*)

where:

param[ ] is an array of specific parameters which are stored in non-volatile memory on the decoder.

length specifies how many individual parameters are contained within param[].

Return Values

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

NOT_SUPPORTED
COMMUNICATION_ERROR
BAD_PARAM

PARAM_SEND

Description

Changes the specified parameter value. Changes can be either PERMANENT or TEMPORARY. PERMANENT changes are stored in the non-volatile memory and retain their settings after cycling power to the decoder. TEMPORARY changes lose effect when power is cycled on the decoder or if they are overwritten by another **StiSsiParamSend**() to the same parameter.

Syntax

int far StiSsiSendParams(*int change_type, int beep, int func_wait*)

where:

change_type is one of the following:

PERMANENT
TEMPORARY

beep is one of the following values:

ONE_SHORT_HIGH
TWO_SHORT_HIGH
THREE_SHORT_HIGH
FOUR_SHORT_HIGH
FIVE_SHORT_HIGH

ONE_SHORT_LOW
TWO_SHORT_LOW
THREE_SHORT_LOW
FOUR_SHORT_LOW
FIVE_SHORT_LOW

ONE_LONG_HIGH
TWO_LONG_HIGH
THREE_LONG_HIGH
FOUR_LONG_HIGH
FIVE_LONG_HIGH

ONE_LONG_LOW
TWO_LONG_LOW
THREE_LONG_LOW
FOUR_LONG_LOW
FIVE_LONG_LOW

FAST_WARBLE
SLOW_WARBLE

MIX1

MIX2

MIX3

MIX4

If *beep* is not one of these values, no beep is generated. This does not cause an error condition.

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

StiSsiCmdAck

Description

Interprets and verifies the CMD_ACK Opcode format. It marks the previous packeted command as sent, allowing the next message format to be queued for transmission.

Syntax

int far StiSsiCmdAck(*char far rx_cmd[ ]*)

where:

char rx_cmd[ ] is the entire buffered command including the length, Ack Opcode, and checksum characters.

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiCmdNak

Description

Interprets and verifies the CMD_NAK Opcode format.

- If the NAK Type is NAK_RESEND, the previous packeted command is resent in accordance with the retry provisions of the Simple Serial Interface.
- If the NAK Type is NAK_DENIED, the previous packeted command is marked as sent and returns an ERROR_UNDEFINED.
- If the NAK Type is NAK_BAD_CONTEXT, the previous packeted command is marked as sent and returns an ERROR_UNDEFINED.

Syntax

int far StiSsiCmdNak(*char far rx_cmd*[ ])

where:

char rx_cmd[ ] is the entire buffered command including the length, Nak Opcode, and checksum characters.

Return Values

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiSsiCommandSend

Description

Queues up a command to be sent to the decoder. This function appends checksum information to the command string.

Syntax

int far StiSsiCommandSend(*char far *ptr, int length, int func_wait*)

where:

ptr is a pointer to the command structure that must be sent to the decoder WITHOUT checksum information.

length specifies the length, in bytes, of this command sequence without checksum digits.

func_wait is one of the following values:

WAIT: Wait for the function to complete before returning to the caller.

NO_WAIT: Do not wait for the function to complete before returning to the caller.

Return Values

if (*func_wait* == WAIT)

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

if (*func_wait* == NO_WAIT)

MSG_PENDING

BAD_PARAM

Note

Interpretation of message packets from the decoder.

StiSsiDecodeData

Description

Interprets and verifies the DECODE_DATA Opcode format. It copies the decoded information into a DECODE_DATA_STRUCT. If a callback function is defined, the callback is executed. If no callback function is defined, a flag is set indicating the availability of data.

Syntax

int far StiSsiDecodeData(*char far rx_cmd[ ]*)

where:

char rx_cmd[ ] is the entire buffered command including the length, Decode_Data Opcode, and checksum characters.

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiEvent

Description

Interprets the Event Opcode format and services the decoder event represented by the Event Code. If a callback function is defined for a specific Event Code, the callback is executed. If no callback function is defined, a counter is incremented to reflect the event.

Syntax

int far StiSsiEvent(*char far rx_cmd[ ]*)

where:

char rx_cmd[ ] is the entire buffered command including the length, Event Opcode, and checksum characters.

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiGetAllParams

Description

Retrieves all current decoder parameter values. The parameters are stored starting the location *ptr*[0]. Up to maxlength characters of parameter data may be stored in *ptr*[ ]. The parameters are returned as data pairs consisting of (parameter_number, parameter_value). The user must parse through the data pairs and associate each parameter_number with a specific decoder capability.

If *maxlength* is less than the number of decoder parameters retrieved, the remaining parameters are discarded. It is recommended that *ptr* be at least 256 bytes long.

Syntax

int far StiSsiGetAllParams(*unsigned char far *ptr, int maxlength*)

where:

ptr[ ] is an array of maxlength bytes where the decoder's parameter information will be deposited.

Return Values

the number of bytes copied into *ptr*[ ]. If an error occurs, the returned error status is one of the following:

NOT_SUPPORTED
COMMUNICATION_ERROR

StiSsiParamPacket

Description

Verifies the specified parameter against its low_limit and hi_limit values. If the parameter is correct, it attempts to put the opcode and parameter into the ParamBatch.data[ ] array. The contents of the array are processed by **StiSsiParamSend**.

Syntax

int far StiSsiParamPacket(*int opcode, int param, int low_limit, int hi_limit*)

where:

opcode specifies the decoder parameter to be modified.

param is the new setting for the decoder parameter.

low_limit is the lowest allowable setting for the parameter.

hi_limit is the highest allowable setting for the parameter.

Return Values

STATUS_OK
BAD_PARAM
BATCH_ERROR

StiSsiParamSend

Description

Interprets and verifies the PARAM_SEND Opcode format. It verifies the message and uses the parameter data to respond to the PARAM_REQUEST command.

Syntax

int far StiSsiParamSend(*char far rx_cmd[ ]*)

where:

char rx_cmd[ ] is the entire buffered command including the length, Param_Send Opcode, and checksum characters.

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiRecovery

Description

Performs retry operations on Host API Messages, ACK, or NAK messages. It retries the last transmission with the resend status bit set. If the maximum number of retries for a given message is exhausted, the function returns a COMMUNICATIONS_ERROR.

Syntax

int far StiSsiRecovery(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiReplyRevision

Description

Interprets and verifies the REPLY REVISION Opcode format. It copies the revision information into a REVISION_DATA_STRUCT.

Syntax

int far StiSsiReplyRevision(*char far rx_cmd[ ]*)

where:

char rx_cmd[ ] is the entire buffered command including the length, Reply_Revision Opcode, and checksum characters.

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

Notes

- High-level implementation of interface handshaking modes.
- Appropriate communications to the Host application, through the Host API function set, of interface events and data.
- This also includes the capability to detect and recover from interface error conditions such as bad packets, examples: invalid checksum, bad format, unexpected format, out of context messages, communications event timeouts, and protocol violations.

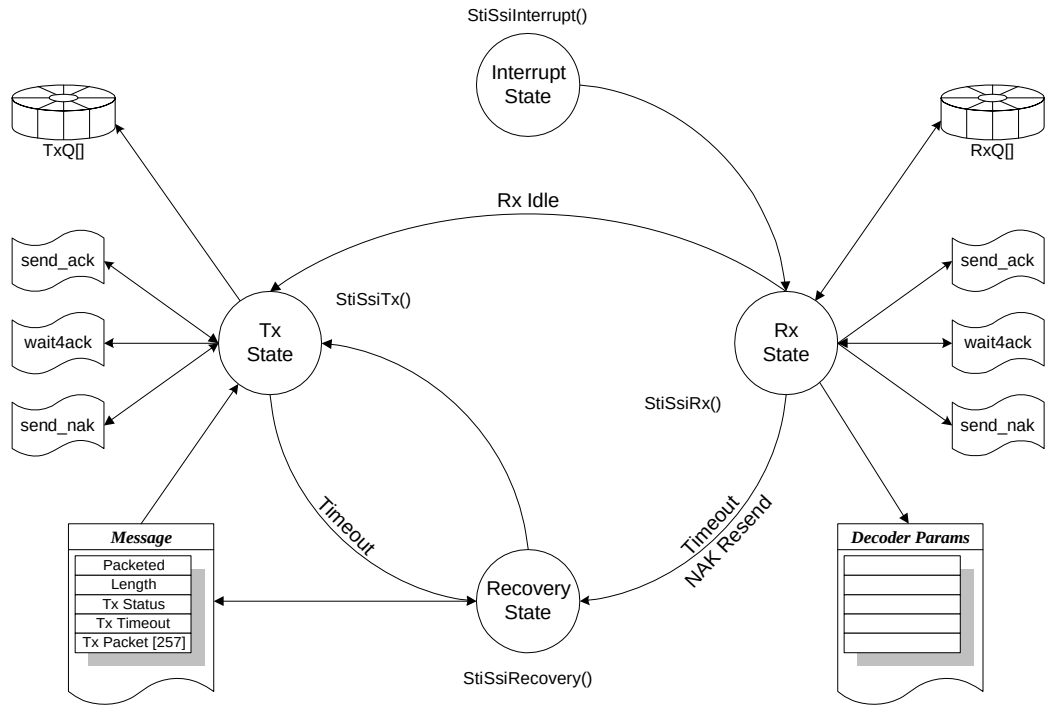

Figure 5-1 *Kernel State Machine*

These requirements are shown in the State Diagrams. The top level state diagram, <Blue>Figure 5-1, is the Kernel State Machine. Enter the Kernel State Machine via the **StsSsiInterrupt**() function. On even ticks, the TX State of the state machine is serviced. On odd ticks, the RX State of the state machine is serviced. This reduces the amount of overhead that the Kernel imposes on the host machine. Since serial communications is relatively slow, and the communications is running half-duplex, the Kernel does not need to run the entire machine during each interrupt.

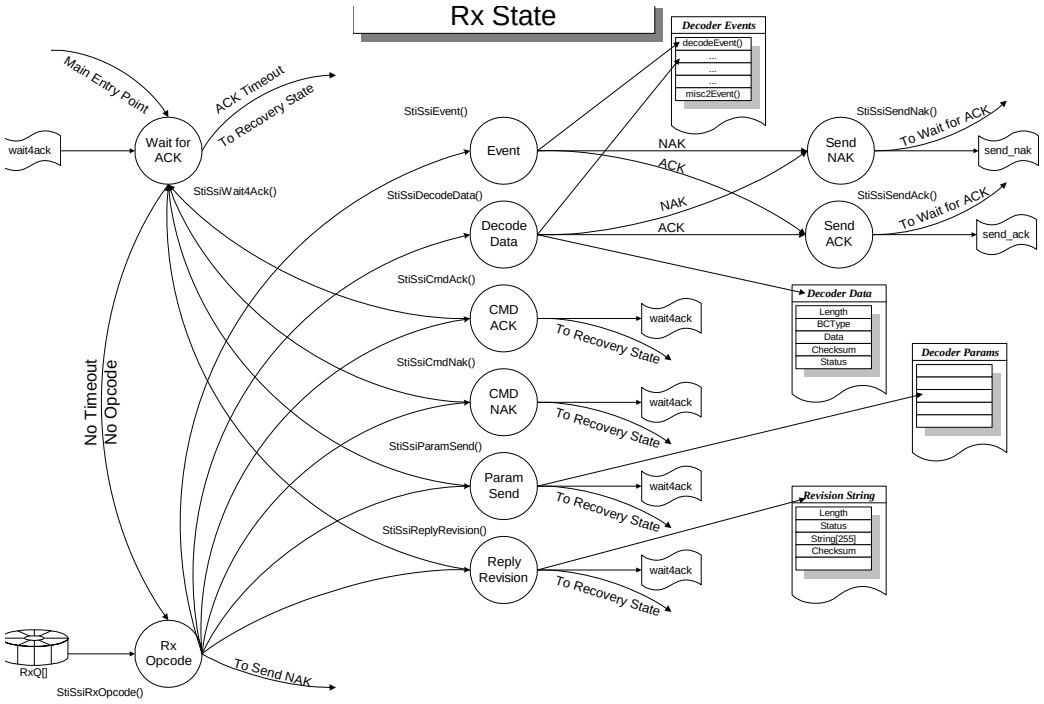

Figure 5-1. RX State Machine Details

<Blue>Figure 5-1 shows the inner details of the RX State machine, including the processing required for ACK/NAK timeouts as well as received command processing. Data structures show how received data is stored for the host application. Callback functions, where implemented, allow the Host API to hook into the data and event processing. High-level communications handshaking is represented by the various flags **wait4ack**, **send_nak**, and **send_ack**. Hooks to the Recovery State permit Host API Messages to be retried as defined in the Simple Serial Interface protocol.

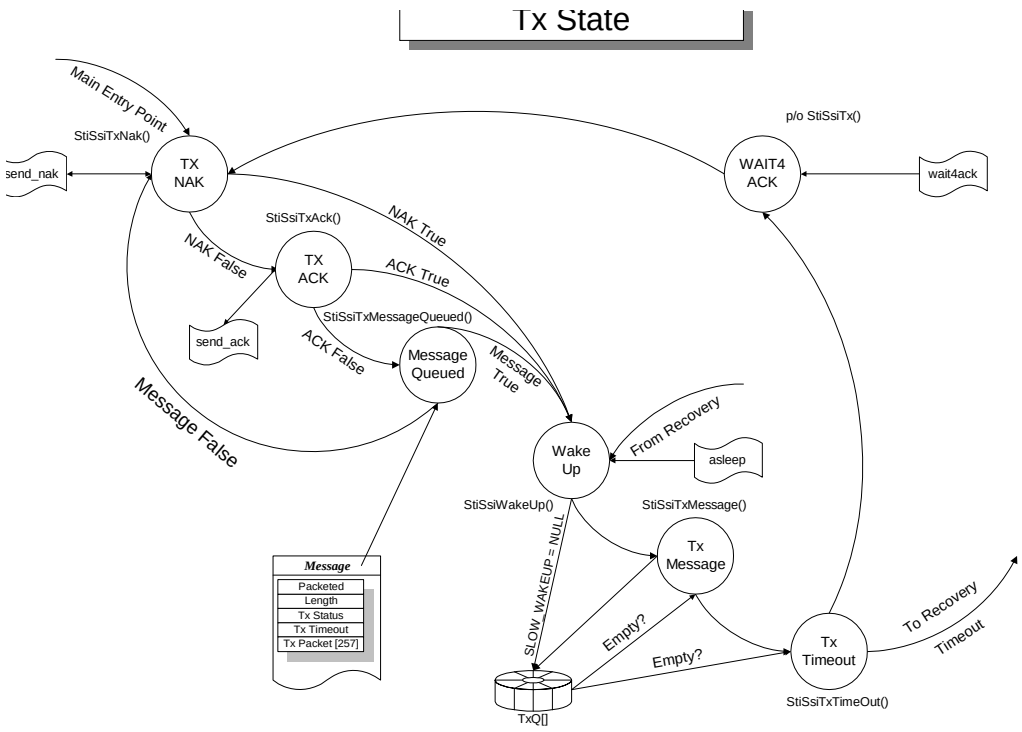

Figure 5-2. TX State Machine Details

<Blue>Figure 5-2 shows the inner details of the TX State machine, including the processing of high-level ACK/NAK handshaking and Host API Message formats. It is able to wake the decoder if the sleep option is enabled. The TX state machine checks the timeout status of messages and handshaking events. Hooks to the Recovery State permit Host API Messages and handshaking to be retried as defined in the Simple Serial Interface protocol.

StiSsiRx

Description

Monitors the received message queue for commands and timing out ACK/NAK/PARAM_SEND commands from the previous message transmission.

Syntax

int far StiSsiRx(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiRxOpcode

Description

Gets messages from the UART receive queue and interprets the opcode field. When an entire message is received, it calls the appropriate message handling routine to verify the contents of the message and perform any actions that are required. If no Opcode matches the message contents, a NAK is sent to the decoder.

Syntax

int far StiSsiRxOpcode(*void*)

Return Values

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiSsiSendAck

Description

Causes an ACK to be transmitted during the next available transmission slot.

Syntax

int far StiSsiSendAck(*void*)

Return Values

STATUS_OK

StiSsiSendNak

Description

Causes a NAK to be transmitted during the next available transmission slot.

Syntax

int far StiSsiSendNak(*int problem*)

where:

problem is the cause of the NAK message. Valid values for problem are:

INVALID_CHECKSUM
BAD_FORMAT
UNEXPECTED_FORMAT
OUT_OF_CONTEXT_MESSAGE
COMMUNICATION_TIMEOUT
PROTOCOL_VIOLATION

Return Values

STATUS_OK

StiSsiTx

Description

Wakes the decoder (if required), queuing ACK, NAK, or Host API Messages to the decoder, and checking the timeout state of transmissions.

Syntax

int far StiSsiTx(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiTxAck

Description

Generates an ACK message to be sent to the decoder when a **send_ack** is requested by the receive state processing. ACK is generated before any pending Host API Messages can be queued. When an ACK is required, it passes the message of to the **StiSsiWakeUp**() function for further processing.

Syntax

int far StiSsiTxAck(*void*)

Return Values

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiSsiTxMessage

Description

“Kick starts” the transmit interrupt service routine and sends the contents of the transmit queue to the decoder.

Syntax

int far StiSsiTxMessage(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiTxMessageQueued

Description

Determines if a Host API Message should to be queued to the decoder. When a message is available, it passes the message to the **StiSsiWakeUp**() function for further processing. Only one Host API Message can be queued up at a time.

Syntax

int far StiSsiTxMessageQueued(*void*)

Return Values

MSG_PENDING

NOT_SUPPORTED

COMMUNICATIONS_ERROR

StiSsiTxNak

Description

Generates a NAK message to be sent to the decoder whenever a **send_nak** is requested by the receive state processing. NAK is generated before any pending Host API Messages can be queued. When a NAK is required, it passes the message of to the **StiSsiWakeUp**() function for further processing.

Syntax

int far StiSsiTxNak(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiTxTimeOut

Description

Counts down the transmit timeout event once per Kernel loop. If the timeout expires before the transmit queue is empty, this function initiates the recovery operation to retry the message in accordance with the Simple Serial Interface protocol. This function halts serial transmissions and flushes the transmit queue in the event of a timeout. If the queue transmits the entire packeted message within the allotted time, the **wait4ack** flag is set to alert the receive processing to expect either an ACK, a NAK, or a Param Send Command (if the message was a Param Request). Unpacketed messages and ACK/NAK commands do not set the **wait4ack** flag.

Syntax

int far StiSsiTxTimeOut(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiWait4Ack

Description

Determines if a Host API Message is acknowledged by the decoder within the specified timeout period. If the timeout expires without an acknowledge, the Kernel performs a retry operation in accordance with the Simple Serial Interface protocol.

Syntax

int far StiSsiWait4Ack(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

NOT_SUPPORTED

StiSsiWakeUp

Description

Wakes the decoder if the low power mode is used or if the decoder was told to Sleep. If required, it generates a wake up command sequence in accordance with the Simple Serial Interface protocol before any messages are sent out.

Syntax

int far StiSsiWakeUp(*void*)

Return Values

STATUS_OK

COMMUNICATIONS_ERROR

StiSsiWakeUpMethod

Description

Determines which method of waking up the decoder is used. The default method is SLOW. When the Kernel needs to wake up the decoder, this setting determines how the wakeup is performed.

- If the SLOW_WAKEUP method is used, the Kernel asserts RTS, transmits a NULL character, waits at least 10 milliseconds, then waits for the CTS signal before transmitting messages to the decoder.
- If the FAST_WAKEUP method is used, the Kernel asserts RTS, then waits for the CTS signal before transmitting messages to the decoder.

Syntax

int far StiSsiWakeUpMethod(*int type*)

where:

type specifies one of the following conditions:

SLOW_WAKEUP

FAST_WAKEUP

Return Values

STATUS_OK

BAD_PARAM

Decoder Setup

The functions in this section are used to set up and initialize the decoder. They are made Kernel functions to hide their operation from the user. They are not called when servicing the Kernel interrupt.

StiDecInitDecoder

Description

This single call initializes the decoder and all of the software structures for the SSI-SDK interface. The initialization consists of:

1. Initialize all memory structures, queues, and pointers.
2. Establish a serial communications connection with the decoder using the user specified port and communication settings.
3. Initialize all interrupt processing related to the serial interface.
4. Enable ACK/NAK handshaking on the decoder.
5. Enable packeted decode data responses.
6. Query the decoder for all of its current programmable settings and allowable configuration parameters.

If any errors are detected, the serial communications are restored to its original form.

Syntax

DOS Only:

int far StiDecInitDecoder(*COMM_STRUCT comm*)

Windows Only:

int far StiDecInitDecoder(*struct COMM_STRUCT comm, HINSTANCE hInstance*)

where:

comm is a structure containing the information about the communications link, such as: Port, Baud Rate, Parity and Stop Bits.

hInstance is a 16-bit handle to an instance of a module or application (Windows Only).

Return Values

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

StiDecRestoreDecoder

Description

This single call restores the comm port and all of the software structures for the the SSI-SDK interface

Syntax

int far StiDecRestoreDecoder(*COMM_STRUCT comm*)

where:

comm is a structure containing the information about the communications link, such as: Port, Baud Rate, Parity and Stop Bits.

Return Values

STATUS_OK

NOT_SUPPORTED

COMMUNICATIONS_ERROR

BAD_PARAM

StiSsiEstablishComm

Description

Establishes a serial communications connection with the decoder using the user-specified port and communication settings. If a connection cannot be established, an error condition is reported.

This function enables ACK/NAK software handshaking with the decoder on the current serial port settings. To show the user Symbol's recommended approach to serial communications, ACK/NAK handshaking is always enabled.

Syntax

int far StiSsiEstablishComm(*COMM_STRUCT comm*)

where:

comm is a structure containing the information about the communications link, such as Port, Baud Rate, Parity and Stop Bits.

Return Values

STATUS_OK

BAD_PARAM

ERROR_UNDEFINED

StiSsiInitStructs

Description

Initializes structures, buffers and pointers.

Syntax

int far StiSsiInitStructs(*void*)

Return Values

STATUS_OK

BAD_PARAM

ERROR_UNDEFINED

StiSsiPacketedDecodeDataFormat

Description

Forces decoded data to be transmitted with the packet format as defined by the serial protocol.

Syntax

int far StiSsiPacketedDecodeDataFormat(*void*)

Return Values

STATUS_OK
NOT_SUPPORTED
COMMUNICATIONS_ERROR

Additional Functions

The following functions are added by Wipro Ltd.,

StiSsiInterruptProcess

Description

Performs the Kernel State processing. It is responsible for directing transmit and receive processing. Additionally, it performs communications checking for unexpected formats, out of context messages, and protocol violations.

Syntax

void StiSsiInterruptProcess(*void*)

StiSsiReset

Description

Resets all the communication-related queues and the structure fields, variables.

Syntax

int StiSsiReset(*void*)

Return Values

STATUS_OK

COMMUNICATION_ERROR

StiSsiVersion

Description

Returns the version of the SSI-SDK. It returns a 16 bit value, where the higher byte is major version and the lower byte is minor version

Syntax

unsigned short StiSsiVersion(*void*)

Return Values

Higher byte - major version number

Higher byte - minor version number

Chapter 6 OS API

Introduction

The SSI-SDK Operating System-specific functions (denoted by the **StiOS**_ prefix) are implemented for the target operating system, and the platform-independent SSI Kernel code. The OS APIs provide the OS-specific services for the external functions. The API provides the following functionalities:

- Enable/disable of interrupts
- Memory Copy and Memory Set functions
- Initialization and restoration of the timer interrupt service routines.

Return Value Definitions

The standard returned status codes are as follows:

- STATUS_OK: the function parameters were verified or the function completed processing successfully.
- MSG_PENDING: a message is pending (Windows).
- NOT_USED: there is no functionality for this OS platform.
- ERROR_UNDEFINED: an error condition not specifically associated with the decoder or its communications.

There are values such as the destination pointer address, other implementation specific APIs return values which are also returned.

StiOSDisableInterrupt

Description

Disables the hardware interrupts.

Syntax

void StiOSDisableInterrupt(*void*)

StiOSEnableInterrupt

Description

Enables the hardware interrupts.

Syntax

void StiOSEnableInterrupt(*void*)

StiOSInitInterrupts

Description

Hooks the callback or timer interrupt function used for asynchronous event indication. If the interrupt was already hooked, it returns an error value. **StiOSRestoreInterrupt** is the complementary function. Every call of this function must be accompanied by its complementary function before return from the application, or the same function is called again.

Syntax

int StiOSInitInterrupts(*volatile int* pWaitTimer, void (*fp_InterruptProcess)(void*)

Where:

PWaitTimer is the volatile pointer to the timeout variable. The pointed value is decremented for every timer tick interrupt inside the interrupt service routine.

fp_InterruptProcess is the function pointer of return type void. This function is called for every timer tick interrupt.

Returned Status

STATUS_OK - if the timer interrupt gets hooked

NOT_SUPPORTED - if the timer interrupt fails to get hooked or if the interrupt is already hooked before calling this function.

StiOSMemoryCopy

Description

Copies the contents of source memory block to destination memory block.

Syntax

*void** StiOSMemoryCopy(*void *dest, const void *src, int count*)

Where:

dest is a pointer to the destination buffer

src is a constant pointer of the source buffer

count gives the number of bytes to be copied

Returned Status

Destination address

StiOSMemorySet

Description

Fills the buffer with the given character value.

Syntax

*void** StiOSMemorySet(*void *dest, int val, int count*)

Where:

dest is a pointer to the destination buffer

val is the value to be set or filled

count gives the number of byte locations to be filled

Returned Status

Destination address

StiOSNotifyError

Description

Displays the error message to the user. Used only in windows version.

Syntax

void StiOSNotifyError(*char *message, char* title*)

Where:

message & *title* are the character string pointers that are displayed

StiOSRestoreInterrupt

Description

Unhooks the callback or timer interrupt function if already hooked. After planting the timer callback using **StiOSInitInterrupts**, call this function before the application quits.

Syntax

int StiOSRestoreInterrupt(*void*)

Returned Status

STATUS_OK - successfully unhooked the interrupt if already hooked

NOT_SUPPORTED - unsuccessful attempt to unhook the timer interrupt

StiOSWait

Description

Yields control to the OS while the application waits for a response from the decoder. This is currently used only in the Windows version, where it looks for the message and receives it if a message is pending.

Syntax

int StiOSWait(*void*)

Returned Status

STATUS_OK - if there is no message

NOT_USED - returned by DOS version

MSG_PENDING - if the message pending is “quit application”

Chapter 7 Port API

Introduction

The SSI-SDK Port API (functions denoted by the **StiPort_** prefix) is hardware-specific software implemented for the target operating system and the platform-independent SSI Kernel code. The Port APIs provide the platform-independent interface for the external functions. The API performs following:

- Initialization of the port and set up of the transmit and receive buffers.
- Character I/O over the port, and block data I/O using the data buffers.
- Communication control and all other port control functionality.

NOTE This API provides a wrapper to the port-specific. OS-specific or vendor-specific code can be called inside this interface. The return data that follows is specific only to this wrapper interface; refer to the specific implementation documentation for other implementation-specific return values.

Return Value Definitions

The standard returned status codes are as follows:

- STATUS_OK: the function parameters were verified or the function completed processing successfully.
- BAD_PARAM: the user-supplied parameter(s) in the function call was not in the expected range.
- STI_PORT_ERROR: an error condition for all the negative values???
- DECODER_OK: the external device is ready for communication.
- DECODER_BUSY - if the external device is not ready for communication.

StiPortAllowDecoderTransmit

Description

Clears the RTS line and allows the host transmission.

Syntax

int StiPortAllowDecoderTransmit(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortClearReceiveBuf

Description

Resets the receive data buffer. This function deletes any characters in the receive buffer for the specified port. After execution, the receive buffer is empty.

Syntax

int StiPortClearReceiveBuf(*int port*)

Where:

port indicates the port number such as 1, 2 etc.,

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortClearTransmitBuf

Description

Resets the transmit data buffer to all zeroes. This function deletes any characters in the transmit buffer for the specified port. After execution, the transmit buffer is empty.

Syntax

int StiPortClearTransmitBuf(*int port*)

Where:

port indicates the port number such as 1, 2 etc.,

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortCommunicationDone

Description

Completes the ongoing serial processing. Inside this function it de-references the transmit and receive buffers already set up.

Syntax

int StiPortCommunicationDone(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortGetChar

Description

Gets a character byte from the port if the character is available within the programmed amount of time; otherwise it returns an error message.

Syntax

int StiPortGetChar(*int port, int pics*)

Where:

port indicates the port number such as 1, 2 etc.,

tics indicates the timeout in terms of number of timer ticks for the function to return the data or error

Returned Status

STI_PORT_ERROR- for all error conditions.

If no error it returns the data read from the port.

StiPortGetDecoderStatus

Description

Gets the status of the clear-to-send control line. The CTS line is used by some error correcting modems to implement hardware flow control. The modem drops CTS to signal the computer not to send data, and raises CTS to signal the computer to continue.

Syntax

int StiPortGetDecoderStatus(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for error conditions

DECODER_OK - if the external device is ready for communication

DECODER_BUSY - if the external device is not ready for communication

StiPortGetLineStatus

Description

Returns the line status register contents of the serial port

Syntax

int StiPortGetLineStatus(*int port)*

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortGetReceiveQueueLength

Description

Returns the receive buffer queue length

Syntax

int StiPortGetReceiveQueueLength(*int port*)

Where:

port indicates the port number such as 1, 2 etc.,

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortGetTransmitQueueLength

Description

Returns the transmit buffer queue length.

Syntax

int StiPortGetTransmitQueueLength(*int port*)

Where:

port indicates the port number such as 1, 2 etc.,

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortIndicateHostReception

Description

Sets the Request to Send control line of the serial port and indicates the host reception.

Syntax

int StiPortIndicateHostReception(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortIndicateHostTransmission

Description

Clears the RTS line and indicates the host transmission.

Syntax

int StiPortIndicateHostTransmission(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortInitPort

Description

Sets up and initializes the Transmit and Receive buffers.

Syntax

int StiPortInitPort(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortPreventDecoderTransmit

Description

Sets the RTS line and prevents host transmission.

Syntax

int StiPortPreventDecoderTransmit(*int port*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortPutChar

Description

Sends a character byte to the port which is transmitted.

Syntax

int StiPortPutChar(*int port, unsigned char dataByte*)

Where:

port indicates the port number such as 1, 2 etc.,

dataByte is the byte of data to be transmitted

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortResetBuffers

Description

Sets the transmit and receive buffers with the null value in the DOS version. The Windows version returns void.

Syntax

void StiPortResetBuffers(*void*)

StiPortSetBaudRate

Description

Sets the serial port's baud rate.

Syntax

int StiPortSetBaudRate(*int port, int BaudRate*)

Where:

port indicates the port number such as 1, 2.

Returned Status

STI_PORT_ERROR - for all error conditions.

StiPortSetCommStruct

Description

Copies the data (port parameters) from the source structure to the destination structure.

Syntax

int StiPortSetCommStruct(*struct COMM_STRUCT source, struct COMM_STRUCT* destination*)

Where:

source is the source communication structure

destination is the pointer to the destination communication structure

Returned Status

BAD_PARAM- if the input value is out of range or invalid

STATUS_OK- if the function completes processing successfully

StiPortSetParams

Description

Sets up the serial port parameters (parity, stopbits, data length).

Syntax

int StiPortSetParams(*int port, int parity, int stopBit, int dataBit*)

Where:

port indicates the port number such as 1, 2

parity indicates the parity (value is implementation-specific; should be #defined)

stopBit indicates the stop bit (value is implementation-specific; should be #defined)

dataBit indicates the data size (value is implementation-specific; should be #defined)

Returned Status

STI_PORT_ERROR - for all error conditions.

Appendix A SSI Commands

This chapter describes each available SSI command, including field descriptions and host and decoder requirements.

SSI Command Lists

The following table lists the available SSI commands alphabetically.

Table A-1. SSI Commands

Name	Type	Opcode	Description	Page
ABORT_MACRO_PDF	H	0x11	Terminates MacroPDF sequence and discards segments.	A-4
AIM_OFF	H	0xC4	Deactivates aim pattern.	A-5
AIM_ON	H	0xC5	Activates aim pattern.	A-6
BATCH_DATA	D	0xD6	Transmits stored decode data.	A-8
BATCH_REQUEST	H	0xD5	Requests stored decode data.	A-10
BEEP	H	0xE6	Sounds the beeper.	A-11
CAPABILITIES_REPLY	D	0xD4	Lists commands which decoder will perform.	A-13
CAPABILITIES_REQUEST	H	0xD3	Requests commands which decoder will perform.	A-15
CMD_ACK	H/D	0xD0	Positive acknowledgment of received packet.	A-16
CMD_NAK	H/D	0xD1	Negative acknowledgment of received packet.	A-18
DECODE_DATA	D	0xF3	Decode data in SSI packet format.	A-20
EVENT	D	0xF6	Event indicated by associated event code.	A-30
FLUSH_MACRO_PDF	H	0x10	Terminates MacroPDF sequence and transmits captured segments.	A-32
Note: D = Decoder, H = Host, H/D = Host/Decoder				

Table A-1. SSI Commands (Continued)

Name	Type	Opcode	Description	Page
FLUSH_QUEUE	H	0xD2	Tells the decoder to eliminate all packets in its transmission queue.	A-33
IMAGE_DATA	D	0xB1	Data comprising the image.	A-34
IMAGER_MODE	H	0xF7	Commands imager into operational modes.	A-36
LED_OFF	H	0xE8	Extinguishes LEDs.	A-37
LED_ON	H	0xE7	Activates LED output.	A-38
PARAM_DEFAULTS	H	0xC8	Sets parameter default values.	A-39
PARAM_REQUEST	H	0xC7	Requests values of certain parameters.	A-40
PARAM_SEND	H/D	0xC6	Sends parameter values.	A-43
REPLY_REVISION	D	0xA4	Replies to REQ_REV with decoder's software/hardware configuration.	A-45
REQUEST_REVISION	H	0xA3	Requests the decoder's configuration.	A-47
SCAN_DISABLE	H	0xEA	Prevents the operator from scanning bar codes.	A-48
SCAN_ENABLE	H	0xE9	Permits bar code scanning.	A-49
SLEEP	H	0xEB	Requests to place the decoder into low power.	A-50
START_SESSION	H	0xE4	Tells decoder to attempt to decode a bar code.	A-51
STOP_SESSION	H	0xE5	Tells decoder to abort a decode attempt.	A-52
VIDEO_DATA	D	0xB4	Data comprising the video.	A-53
WAKEUP	H	N/A	Wakes up decoder after it's been powered down.	A-55
Note: D = Decoder, H = Host, H/D = Host/Decoder				

<Blue>Table A-2 lists the SSI commands by Opcode.

Table A-2. SSI Commands by Opcode

Opcode	Name
0x10	FLUSH_MACRO_PDF
0x11	ABORT_MACRO_PDF
0xA3	REQUEST_REVISION
0xA4	REPLY_REVISION
0xB0	Reserved
0xB1	IMAGE_DATA
0xB4	VIDEO_DATA

Table A-2. SSI Commands by Opcode (Continued)

Opcode	Name
0xC4	AIM_OFF
0xC5	AIM_ON
0xC6	PARAM_SEND
0xC7	PARAM_REQUEST
0xC8	PARAM_DEFAULTS
0xD0	CMD_ACK
0xD1	CMD_NAK
0xD2	FLUSH_QUEUE
0xD3	CAPABILITIES_REQUEST
0xD4	CAPABILITIES_REPLY
0xE4	START_SESSION
0xE5	STOP_SESSION
0xE6	BEEP
0xE7	LED_ON
0xE8	LED_OFF
0xE9	SCAN_ENABLE
0xEA	SCAN_DISABLE
0xEB	SLEEP
0xF3	DECODE_DATA
0xF6	EVENT
0xF7	IMAGER_MODE
N/A	WAKEUP

ABORT_MACRO_PDF

Description

Terminates MacroPDF sequence and discards all captured segments.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	11h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	11h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type (for parameters)** 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

None.

Decoder Requirements

The decoder terminates the current MacroPDF sequence and discards all captured MacroPDF segments.

AIM_OFF

Description

Turns off aiming pattern.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	C4h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	C4h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type (for parameters)** 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

This command applies only to decoders that support an aim pattern.

Decoder Requirements

The decoder turns off the aim pattern, and responds with a CMD_ACK (if ACK/NAK handshaking is enabled).

If the aim pattern is not supported, the decoder responds with NAK_DENIED (if ACK/NAK handshaking is enabled).

AIM_ON

Description

Turns on aiming pattern.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	C5h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	C5h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type (for parameters)** 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

This command applies only to decoders which support an aim pattern.

Decoder Requirements

The decoder turns on the aim pattern, and responds with a CMD_ACK (if ACK/NAK handshaking is enabled).

If the aim pattern is not supported, the decoder responds with NAK_DENIED (if ACK/NAK handshaking is enabled).

The Aim Duration parameter controls the amount of time the aiming pattern stays on during a trigger pull. The valid values for this parameter are 0 - 99, which equal 0.1 to 9.9 seconds in 100 msec increments. <Blue>Table A-3 lists Aim mode behavior in various situations.

Table A-3. Aim Mode

Command Sequence	Action performed	Aim duration parameters
AIM_ON	Turns on the aiming pattern indefinitely.	aim duration = 0
AIM_OFF	Turns off the aiming pattern.	aim duration = 0
AIM_ON, START_DECODE	Turns on the aiming pattern, when START_DECODE received turns on scan pattern and begins decoding.	aim duration = 0
AIM_ON,AIM_OFF, START_DECODE	Turns on aiming pattern, turns off aiming pattern, turns on scan pattern and begins decoding.	aim duration = 0
START_DECODE	Turns on aiming pattern for aim duration time, turns on scan pattern and begins decoding.	aim duration > 0

BATCH_DATA

Description

Transmits stored decode data as a reply to the BATCH_REQUEST command. Scanners that can not store scans send a NAK DENIED or NAK BAD CONTEXT response.

Packet Format

Length	Opcode	Message Source	Status	Bar Code String(s)	Checksum
	D6h	00h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D6h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Bar Code String(s)	See Text	Variable	Data from a bar code scan in the bar code string format. Multiple instances of this field may be repeated in one message. For multipacket messages, a partial string may be sent, continued in the next packet.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Bar Code String

Each string is stored in this message in three components: Size, Type, and Scan Data. To specify a bar code string these components are combined in the order specified.

- Size: One byte value that contains the length of the Scan Data component
- Type: One byte value that indicates the bar code type of the data scanned:
  - A = UPC/EAN
  - B = Code 39
  - D = EAN 128
  - F = Interleaved 2 of 5
  - G = Discrete 2 of 5
  - K = Code 128
  - N = Coupon code
  - W = Web Code
- Scan Data: One or more bytes of the scanner bar code data in ASCII.

BATCH_REQUEST

Description

Requests stored decode data from the scanner. The scanner responds with the BATCH_REQUEST command. Scanners that can not store scans respond with a NAK DENIED or NAK BAD CONTEXT.

Packet Format

Length	Opcode	Message Source	Status	Bar Code String(s)	Checksum
	D5h	04h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D5h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

BEEP

Description

Sounds the beeper.

Packet Format

Length	Opcode	Message Source	Status	Beep Code	Checksum
05h	E6h	04h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E6h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type (for parameters)** 0 = Temporary change 1 = Permanent change
Beep Code	See <Blue>Table A-4.	1 Byte	Number that identifies a beep sequence.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This Opcode instructs the receiver to sound the beep sequence indicated by the Beep Code field.

For <Blue>Table A-4, Duration is the length of a sound, Pitch is the pitch of the sound, and Number of Beeps indicates the number of times a beep pitch is repeated at the specified duration.

Table A-4. Beep Code Definitions

Beep Code	Duration	Pitch	Number of Beeps	Beep Code	Duration	Pitch	Number of Beeps
00h	Short	High	1	**0Dh**	Long	High	4
01h	Short	High	2	**0Eh**	Long	High	5
02h	Short	High	3	**0Fh**	Long	Low	1

Table A-4. Beep Code Definitions (Continued)

Beep Code	Duration	Pitch	Number of Beeps	Beep Code	Duration	Pitch	Number of Beeps
03h	Short	High	4	**10h**	Long	Low	2
04h	Short	High	5	**11h**	Long	Low	3
05h	Short	Low	1	**12h**	Long	Low	4
06h	Short	Low	2	**13h**	Long	Low	5
07h	Short	Low	3	**14h**	Fast Warble	Hi-Lo-Hi-Lo	4
08h	Short	Low	4	**15h**	Slow Warble	Hi-Lo-Hi-Lo	4
09h	Short	Low	5	**16h**	Mix 1	Hi-Lo	2
0Ah	Long	High	1	**17h**	Mix 2	Lo-Hi	2
0Bh	Long	High	2	**18h**	Mix 3	Hi-Lo-Hi	3
0Ch	Long	High	3	**19h**	Mix 4	Lo-Hi-Lo	3

Host Requirements

The host sends this command to cause the decoder to beep. The host may also send these beep codes as part of the PARAM_SEND directive.

Decoder Requirements

When the decoder receives this command, it beeps the sequence provided in the BEEP directive. If ACK/NAK handshaking is enabled, the decoder ACKs if a valid beep code is requested. Otherwise it sends CMD_NAK, host directive denied.

CAPABILITIES_REPLY

Description

Decoder details the serial capabilities.

Packet Format

Length	Opcode	Message Source	Status	Data	Checksum
04h	D4h				

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D4h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Data			<Xref>Table A-5 on page A-14.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

The host must not CMD_ACK or CMD_NAK this message, as this is a natural response to the CAPABILITIES_REQUEST message.

Decoder Requirements

The decoder sends this message upon receipt of the CAPABILITIES_REQUEST message.

Table A-5. Data Fields

Field	Size	Description		Supported
Baud Rates Supported	2 Bytes Bit mapped	**Bit**	**Definition**	1 = Supported 0 = Not Supported
		0	300 Baud	
		1	600 Baud	
		2	1200 Baud	
		3	2400 Baud	
		4	4800 Baud	
		5	9600 Baud	
		6	19200 Baud	
		7	28800 Baud	
		8	38400 Baud	
		9	57600 Baud	
		10	115200 Baud	
		11	Reserved	
		12	Reserved	
		13	Reserved	
		14	Reserved	
		15	Reserved	
Misc Serial Parameters	1 Byte Bit Mapped	**Bit**	**Definition**	1 = Supported 0 = Not Supported
		0	Odd Parity	
		1	Even Parity	
		2	Parity None	
		3	Check Parity	
		4	Do Not Check Parity	
		5	One Stop Bit	
		6	Two Stop Bits	
Multi Packet Options	1 Byte Bit Mapped	**Bit**	**Definition**	1 = Supported 0 = Not Supported
		0	Option 1	
		1	Option 2	
		2	Option 3	
Command List	1 Byte per Command	In this sequential list, the decoder details the commands it supports. For example, imagers support video commands, while laser-based decoders do not. Commands associated with video mode will not appear in the list for laser-based decoders, but will for imagers.		

CAPABILITIES_REQUEST

Description

Requests the decoder's serial capabilities.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	D3h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D3h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

The host transmits this message to request the serial capabilities of the decoder system.

Decoder Requirements

Upon receipt of this command, the decoder responds with the CAPABILITIES_REPLY message.

CMD_ACK

Description

Positive acknowledgment of received packet.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	D0h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D0h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder 4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This message is sent to the SSI packet transmitter when the received packet passes the checksum check and no negative acknowledgment conditions apply (see CMD_NAK). If the data is in response to a command (e.g., PARAM_REQUEST, REQUEST_REVISION, etc.), no ACK is sent.

NOTE ACK/NAK handshaking can be disabled, although we recommend it remain enabled.

NOTE DO NOT respond to a valid ACK or NAK message.

Host Requirements

A CMD_ACK or response data must be sent by the decoder within the programmable Serial Response Time-out to acknowledge receipt of all messages, unless noted otherwise in the message description section. If the host sends data and does not receive a response within the programmable serial response time-out, it should resend the message (with the retransmit status bit set) before declaring a failure. The host should limit the number of retries.

Decoder Requirements

A CMD_ACK or response data must be sent by the decoder within the programmable Serial Response Time-out to acknowledge receipt of all messages, unless noted otherwise in the message description section. If the decoder does not receive an ACK within this time period, it sends the previous message again (retry). The decoder retries two more times (with the retransmit status bit set) before declaring a transmit error.

CMD_NAK

Description

Negative acknowledgment of received packet.

Packet Format

Length	Opcode	Message Source	Status	Cause	Checksum
05	D1h				

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D1h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder 4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Cause	Reason code	1 Byte	Identifies the reason the NAK occurred: 0 = Reserved 1 = (RESEND) Checksum failure 2 = (BAD_CONTEXT) Unexpected or Unknown message 3 = Reserved 4 = Reserved 5 = Reserved 6 = (DENIED) Host Directive Denied 7 = Reserved 8 = Reserved 9 = Reserved 10 = (CANCEL) Undesired Message
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This message is sent when the received packet fails the checksum verification or some error occurred while handling the message.

NOTE ACK/NAK handshaking can be disabled, although we recommend it remain enabled.

NOTE DO NOT respond to a valid ACK or NAK message.

NAK types supported by the decoder are listed in <Blue>Table A-6.

Table A-6. Decoder-Supported NAK Types

NAK Type	Meaning	Receiver Action
BAD_CONTEXT	Host does not recognize the command.	
CANCEL	Host does not want the message in progress.	Decoder discards the current message.
DENIED	Host is unable to comply with the requested message (e.g., beep code is out of range).	Do not send data with this message again. Developer should check values with specified values. Developer should ensure the proper character is sent, if using wake-up character.
RESEND	Checksum incorrect.	Ensure checksum is correct. Limit number of resends. Send packet again with resend bit set.

The decoder only resends a message twice. If the message has not been sent successfully at that time, the decoder declares a transmit error, and issues transmit error beeps (LO-LO-LO-LO).

CMD_NAK, cancel is a special message used when the decoder is sending a message the host does not want, for example a very large image message. The message is discarded by the decoder upon receipt of the CMD_NAK, cancel. This only affects the first queued message. Subsequent messages are not touched. If the host wants the decoder to discard all messages, the host must send a FLUSH_QUEUE message.

DECODE_DATA

Description

Decode data in SSI packet format.

Packet Format

Length	Opcode	Message Source	Status	Bar code Type	Decode Data	Checksum
	F3h	00h				

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	F3h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Bar Code Type	See <Blue>Table A-7	1 Byte	Identifies the scanned data code type. 0 = Not Applicable
Decode Data	<data>	Variable	Data is decoded data including prefix and suffix sent in ASCII format.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This Opcode is used by the decoder when packeted data is selected to send decoded bar code data to the host. The decoded message is contained in the *Decode Data* field.

If the decoded data contains more structure than can be presented in the standard format, the Bar Code Type field is set to 0x99 to indicate the Decode Data message contains multiple packets. The format of the Decode

Data field contains the actual Bar Code Type and a packeted form of decode data. For example, a packeted Decode Data message for Micro PDF417 would look like:

Length	Opcode	Message Source	Status	Bar code Type	Decode Data	Checksum
12	F3h	00h	0	99	see below	

where the Decode Data field is broken out as follows:

Actual Bar code Type	# of Packets	Spare Byte	Byte Length of Packet #1	Data	Spare Byte	Byte Length of Packet #2	Data
1A	2	0	00 03	ABC	0	00 04	DEFG

Note that the *Packet Length* subfields consist of two bytes, where the first byte represents the high value of length x 256.

Structured Append

Structured append data for PDF417 and Micro PDF417 can be transmitted in either an unstructured format which adheres to the PDF417 specification, or a structured "smart format" using the multipacketed format above. The *Bar Code Type* field contains 0x99 and data is sent from a single structured append symbol in two Decode Data packets. The first packet contains the main bar code data, and the second contains any bar code identification enabled for transmission (e.g., the control block, optional fields, symbol terminator). Each field begins with its identifying marker codeword (e.g., \928 for control blocks, \923 for optional fields, and \922 for the symbol terminator).

<Blue>Table A-7 and <Blue>Table A-8 lists all supported code types, by code name and hex value (SSI ID). The associated hex value for each code (as required) is entered in the *Code Type* field.

NOTE For multipacketed data, this code type appears in every packet.

Table A-7. Code Types and Identifiers

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
Bookland	0x16	L	X	0
Codabar	0x02	C	F	0 (1) - standard (ABC)
Code 11	0x0C	H	H	0 (1) [2] - 1 (2) [0] check digits included
Code 128	0x03	D	C	0 (also see UCC/EAN-128)
Code 32	0x20	B	A	Same rules as for Code 39

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-7. Code Types and Identifiers (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
Code 39	0x01	B	A	0 - no check digit 1 (3) - check digit included (excluded)
Code 39 Full ASCII	0x13	B	A	4 - no check digit 5 (7) - check digit included (excluded)
Code 93	0x07	E	G	0
Composite (CC-A + EAN-128)	0x51	T	See <Blue>Table A-9	
Composite (CC-A + EAN-13)	0x52	T	See <Blue>Table A-9	
Composite (CC-A + EAN-8)	0x53	T	See <Blue>Table A-9	
Composite (CC-A + RSS Expanded)	0x54	T	See <Blue>Table A-9	
Composite (CC-A + RSS Limited)	0x55	T	See <Blue>Table A-9	
Composite (CC-A + RSS-14)	0x56	T	See <Blue>Table A-9	
Composite (CC-A + UPC-A)	0x57	T	See <Blue>Table A-9	
Composite (CC-A + UPC-E)	0x58	T	See <Blue>Table A-9	
Composite (CC-B + EAN-128)	0x61	T	See <Blue>Table A-9	
Composite (CC-B + EAN-13)	0x62	T	See <Blue>Table A-9	
Composite (CC-B + EAN-8)	0x63	T	See <Blue>Table A-9	
Composite (CC-B + RSS Expanded)	0x64	T	See <Blue>Table A-9	
Composite (CC-B + RSS Limited)	0x65	T	See <Blue>Table A-9	
Composite (CC-B + RSS-14)	0x66	T	See <Blue>Table A-9	

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-7. Code Types and Identifiers (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
Composite (CC-B + UPC-A)	0x67	T	See <Blue>Table A-9	
Composite (CC-B + UPC-E)	0x68	T	See <Blue>Table A-9	
Composite (CC-C + EAN-128)	0x59	T	See <Blue>Table A-9	
Coupon Code	0x17	N	E+C [1]	0+1
Cue CAT Code	0x38	Q	X	0
D25	0x04	G	S	0
Data Matrix	0x1B	P00	d	4 (1) - ECC 200 with (w/o) ECI
EAN-128	0x0F	K	C	1 (2) - character 1 (2) is Function 1 (F1)
EAN-13	0x0B	A	E	0
EAN-13 + 2	0x4B	A	E + E[2]	0 for main block; 1 for supplemental
EAN-13 + 5	0x8B	A	E + E[2]	0 for main block; 2 for supplemental
EAN-8	0x0A	A	E	4
EAN-8 + 2	0x4A	A	E + E[2]	4 for main block; 1 for supplemental
EAN-8 + 5	0x8A	A	E + E[2]	4 for main block; 2 for supplemental
IATA	0x05	G	S	0
ISBT-128	0x19	D	C	0
ISBT-128 Concat.	0x21	D	C	4
ITF	0x06	F	I	Same rules as for Code 39
Macro Micro PDF	0x9A	X	L	Same rules as for Micro PDF-417
Macro PDF-417	0x28	X	L	Same rules as for PDF-417
Maxicode	0x25	P02	U	1 - Mode 0, 2 or 3, without ECI 3 (1) - Extended EC with (w/o) ECI
Micro PDF	0x1A	X	L	3 - Code 128 emul: implied F1 in 1st position 4 - Code 128 emul: F1 after 1st letter/digits 5 - Code 128 emul: no implied F1

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-7. Code Types and Identifiers (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
MSI	0x0E	J	M	0 - Modulo 10 symbol check character validated and transmitted 1 - Modulo 10 symbol check character validated but not transmitted
Multipacket Format	0x99	N/A	N/A	Data is packeted; SSI ID is embedded in decode data.
Parameter (FNC3)	0x33	N/A	N/A	
PDF-417	0x11	X	L	0 - Conforms with 1994 PDF-417 spec 1 - Backslash characters doubled 2 - Backslash characters not doubled
Planet (US)	0x1F	P04	X	
Postal (Australia)	0x23	P09	X	0
Postal (Dutch)	0x24	P08	X	0
Postal (Japan)	0x22	P05	X	0
Postal (UK)	0x27	P06	X	0
Postbar (CA)	0x26	P07	X	0
Postnet (US)	0x1E	P03	X	0
QR Code	0x1C	P01	Q	0
RSS Expanded	0x32	R		
RSS Limited	0x31	R		
RSS-14	0x30	R		
Scanlet Webcode	0x37	W	X	0
TLC-39	0x5A	T	See <Blue>Table A-9	
Trioptic	0x15	M	X	0
UPCA	0x08	A	E	0
UPCA + 2	0x48	A	E + E[2]	0 for main block; 1 for supplemental
UPCA + 5	0x88	A	E + E[2]	0 for main block; 2 for supplemental
UPCE [3]	0x09	A	E	0
UPCE + 2	0x49	A	E + E[2]	0 for main block; 1 for supplemental
UPCE + 5	0x89	A	E + E[2]	0 for main block; 2 for supplemental

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-7. Code Types and Identifiers (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
UPCE1	0x10	A	E	0
UPCE1 + 2	0x50	A	E + E^2	0 for main block; 1 for supplemental
UPCE1 + 5	0x90	A	E + E^2	0 for main block; 2 for supplemental

Notes:
1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-8. Code Types by SSI ID

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
Code 39	0x01	B	A	0 - no check digit 1 (3) - check digit included (excluded)
Codabar	0x02	C	F	0 (1) - standard (ABC)
Code 128	0x03	D	C	0 (also see UCC/EAN-128)
D25	0x04	G	S	0
IATA	0x05	G	S	0
ITF	0x06	F	I	Same rules as for Code 39
Code 93	0x07	E	G	0
UPCA	0x08	A	E	0
UPCE [3]	0x09	A	E	0
EAN-8	0x0A	A	E	4
EAN-13	0x0B	A	E	0
Code 11	0x0C	H	H	0 (1) [2] - 1 (2) [0] check digits included
MSI	0x0E	J	M	0 - Modulo 10 symbol check character validated and transmitted 1 - Modulo 10 symbol check character validated but not transmitted

Notes:
1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-8. Code Types by SSI ID (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
EAN-128	0x0F	K	C	1 (2) - character 1 (2) is Function 1 (F1)
UPCE1	0x10	A	E	0
PDF-417	0x11	X	L	0 - Conforms with 1994 PDF-417 spec 1 - Backslash characters doubled 2 - Backslash characters not doubled
Code 39 Full ASCII	0x13	B	A	4 - no check digit 5 (7) - check digit included (excluded)
Trioptic	0x15	M	X	0
Bookland	0x16	L	X	0
Coupon Code	0x17	N	E+C[1]	0+1
ISBT-128	0x19	D	C	0
Micro PDF	0x1A	X	L	3 - Code 128 emul: implied F1 in 1st position 4 - Code 128 emul: F1 after 1st letter/digits 5 - Code 128 emul: no implied F1
Data Matrix	0x1B	P00	d	4 (1) - ECC 200 with (w/o) ECI
QR Code	0x1C	P01	Q	0
Postnet (US)	0x1E	P03	X	0
Planet (US)	0x1F	P04	X	
Code 32	0x20	B	A	Same rules as for Code 39
ISBT-128 Concat.	0x21	D	C	4
Postal (Japan)	0x22	P05	X	0
Postal (Australia)	0x23	P09	X	0
Postal (Dutch)	0x24	P08	X	0
Maxicode	0x25	P02	U	1 - Mode 0, 2 or 3, without ECI 3 (1) - Extended EC with (w/o) ECI
Postbar (CA)	0x26	P07	X	0
Postal (UK)	0x27	P06	X	0
Macro PDF-417	0x28	X	L	Same rules as for PDF-417
RSS-14	0x30	R		

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-8. Code Types by SSI ID (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
RSS Limited	0x31	R		
RSS Expanded	0x32	R		
Parameter (FNC3)	0x33	N/A	N/A	
Scanlet Webcode	0x37	W	X	0
Cue CAT Code	0x38	Q	X	0
UPCA + 2	0x48	A	E + E [2]	0 for main block; 1 for supplemental
UPCE + 2	0x49	A	E + E[2]	0 for main block; 1 for supplemental
EAN-8 + 2	0x4A	A	E + E[2]	4 for main block; 1 for supplemental
EAN-13 + 2	0x4B	A	E + E[2]	0 for main block; 1 for supplemental
UPCE1 + 2	0x50	A	E + E[2]	0 for main block; 1 for supplemental
Composite (CC-A + EAN-128)	0x51	T	See <Blue>Table A-9	
Composite (CC-A + EAN-13)	0x52	T	See <Blue>Table A-9	
Composite (CC-A + EAN-8)	0x53	T	See <Blue>Table A-9	
Composite (CC-A + RSS Expanded)	0x54	T	See <Blue>Table A-9	
Composite (CC-A + RSS Limited)	0x55	T	See <Blue>Table A-9	
Composite (CC-A + RSS-14)	0x56	T	See <Blue>Table A-9	
Composite (CC-A + UPC-A)	0x57	T	See <Blue>Table A-9	
Composite (CC-A + UPC-E)	0x58	T	See <Blue>Table A-9	
Composite (CC-C + EAN-128)	0x59	T	See <Blue>Table A-9	
TLC-39	0x5A	T	See <Blue>Table A-9	
Composite (CC-B + EAN-128)	0x61	T	See <Blue>Table A-9	
Composite (CC-B + EAN-13)	0x62	T	See <Blue>Table A-9	

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-8. Code Types by SSI ID (Continued)

Symbology	SSI ID	Code ID	AIM ID Letter	AIM ID Modifier
Composite (CC-B + EAN-8)	0x63	T	See <Blue>Table A-9	
Composite (CC-B + RSS Expanded)	0x64	T	See <Blue>Table A-9	
Composite (CC-B + RSS Limited)	0x65	T	See <Blue>Table A-9	
Composite (CC-B + RSS-14)	0x66	T	See <Blue>Table A-9	
Composite (CC-B + UPC-A)	0x67	T	See <Blue>Table A-9	
Composite (CC-B + UPC-E)	0x68	T	See <Blue>Table A-9	
UPCA + 5	0x88	A	E + E[2]	0 for main block; 2 for supplemental
UPCE + 5	0x89	A	E + E[2]	0 for main block; 2 for supplemental
EAN-8 + 5	0x8A	A	E + E[2]	4 for main block; 2 for supplemental
EAN-13 + 5	0x8B	A	E + E[2]	0 for main block; 2 for supplemental
UPCE1 + 5	0x90	A	E + E[2]	0 for main block; 2 for supplemental
Multipacket Format	0x99	N/A	N/A	Data is packeted; SSI ID is embedded in decode data.
Macro Micro PDF	0x9A	X	L	Same rules as for Micro PDF-417

Notes:

1. **E+C** denotes 2 AIM IDs are transmitted: one for the UPC/EAN block; the second prefixes the extended EAN-128 data.
2. **E+E** denotes 2 AIM IDs are transmitted: the first prefixes the main UPC/EAN block; the second prefixes the supplemental block.
3. UPCE, UPCE1, and UPCA are converted to EAN-13 for AIM ID.

Table A-9. Composite Code Data Formats

1D Component	Data Format	
	Standard Mode	**EAN-128 Emulation Mode**
EAN-13, UPC-A, UPC-E	1D:]E0 2D:]e0 See note 5	1D:]E0 2D:]C1 before each EAN-128 split transmission See notes 3 -5
EAN-8	1D:]E4 2D:]e0 See note 5	1D:]E4 2D:]C1 before each EAN-128 split transmission See notes 3 -5
RSS-14 **RSS Limited**	1D:]e0 2D:]e1 See note 2	]C1 before each EAN-128 split transmission See notes 3 -5
Code 39 (TLC39)	*ANSI MH10.8.3M syntax:* 06 Format: []>$^{R}_{S}$06 $^{G}_{S}$ 6P 1D $^{G}_{S}$ S 2D $^{R}_{S}$ EOT 05 Format: []>$^{R}_{S}$05 $^{G}_{S}$ 906P 1D $^{G}_{S}$ 8004 2D $^{R}_{S}$ EOT See note 6	
EAN-128 **RSS Expanded**	If the last AI in the EAN-128 is a predefined, fixed length:]e0 Otherwise,]e0 GS See note 2	]C1 before each EAN-128 split transmission See notes 3 and 4

Notes:

1. All Function 1 characters in the 1D and 2D are sent as $^{G}_{S}$ (29_{10}); the first Function 1 in the EAN-128 is not transmitted.
2. In standard mode, the data following symbol separator begins with AIM ID "]e1". The data following the composite component escape mechanism begins with AIM ID "]e2" if ECI interpretation is enabled, "]e3" if ECI interpretation is not enabled.
3. In EAN-128 emulation mode, each packet is split on an AI boundary and limited to less than 48 characters.
4. In EAN-128 emulation mode, data is discarded after the first symbol separator or escape mechanism.
5. If the UPC/EAN component has a supplemental ,]E1 precedes a 2-digit supplemental and]E2 precedes the 5-digit supplemental
6. RS is character 30_{10} and EOT is character 04. The transmitted format (05 or 06) is data dependent.

Host Requirements

If DECODE_EVENT reporting is enabled, the decode event message is received before the DECODE_DATA message. If ACK/NAK handshaking is enabled, the host responds to each of these messages.

Decoder Requirements

Decode data is sent in this format if packeted decode data is selected via parameter. The host responds to this message with a CMD_ACK, if ACK/NAK handshaking is enabled.

EVENT

Description

Indicates selected events occurred.

Packet Format

Length	Opcode	Message Source	Status	Event Code	Checksum
05h	F6h	00h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	F6h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Event Code	Type of Event Code.	1 Byte	See <Xref>Table A-10 on page A-31.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This message is sent by the decoder when an enabled event occurs. Use <Blue>Table A-10 and parameters F0h 00h through F0h 07h to determine which events you would like to be reported.

Host Requirements

The host receives this message when a selected event occurs.

Decoder Requirements

Generate this message when a selected event occurs. Events may vary by decoder type.

Table A-10. Event Codes

Event	Code
Boot Event	03h
Decode Event	01h
Parameter Defaults	0Ah
Parameter Entry Error	07h
Parameter Num Expected	0Fh
Parameter Stored	08h

FLUSH_MACRO_PDF

Description

Terminates MacroPDF sequence and sends all captured segments.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	10h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	10h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type (for parameters)** 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

None.

Decoder Requirements

The decoder terminates the current MacroPDF sequence and transmits the captured MacroPDF segments.

FLUSH_QUEUE

Description

Eliminates content of decoder's transmission queue.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	D2h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	D2h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This message is sent by the host to instruct the decoder to discard or flush the transmission queue. This is useful when the decoder is attempting to send a lengthy multipacket message. If the host does not want the message, the host can interrupt the decoder (by asserting RTS) and send a FLUSH_QUEUE message.

The decoder ACK/NAKs the FLUSH_QUEUE message. No further packets in the transmission queue are sent. Note that this does not abort decoder actions that cause packets to be added to the transmission queue.

We recommend issuing a SCAN_DISABLE prior to issuing a FLUSH_QUEUE so that new elements are not added to the queue just after it is emptied. Also, paradoxical cases may arise if a SCAN_DISABLE is not issued first.

IMAGE_DATA

Description

A JPEG, BMP, or TIFF image.

Packet Format

Length	Opcode	Message Source	Status	Data	Checksum
	B1h	01h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	B1h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Data Content		Up to 251 Bytes	Image Data records.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This packet contains image information. Images sent from the decoder to the host are described by the image preamble contained in the first 10 bytes of the first packet of the image. The details of the image preamble follow. Due to the small packet size of SSI, multiple packets of image data should be received by the host and re-assembled in the order given by the decoder. The packets describe, for example, a JPEG image when re-assembled.

The image preamble consists of the following fields:

Table A-11. Image Preamble Fields

Field	Field Size	Description
File size	4 byte field	Number of bytes in the overall image.
Image Width	2 byte field	Image width in pixels
Image Height	2 byte field	Image height in pixels
Image Type	1 byte field	0x31 = JPEG Image File 0x33 = BMP Windows Bit Map File 0x34 = TIFF File Note: These values are ASCII.
Bits per Pixel	1 byte field	Number of bits per pixel in image 0 = 1 bit/pixel Black White Image 1 = 4 bit/pixel 16 Grayscale Image 2 = 8 bit/pixel 256 Grayscale Image
Note: The preamble only appears in the first packet of a multipacket message.		

In a multipacketed environment, one image frame is spread over several packets in the following format:

Packet 1

Header	Preamble	Image Data, Part 1	Checksum

Packet 2

Header	Image Data, Part 2	Checksum

.

.

.

Packet N

Header	Last of Image Data	Checksum

This is re-assembled by the host into:

Preamble	Image Frame

IMAGER_MODE

Description

Commands Imager into Operational Modes.

Packet Format

Length	Opcode	Message Source	Status	Data	Checksum
05h	F7h	00h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	F7h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Data Content		1 Byte	Value 0, 1, or 2 decimal
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

This command is supported by the imager only. The host sends this command with the data field set to 0 for decode mode, 1 for image capture mode, and 2 for video mode.

Decoder Requirements

The decoder (imager) sends a CMD_ACK if the mode is valid, and CMD_NAK if not.

LED_OFF

Description

De-activates LED output.

Packet Format

Length	Opcode	Message Source	Status	LED Selection	Checksum
05h	E8h	04h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E8h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
LED Selection	Bit 0 - 7: LED bit numbers to turn off.	1 Byte	Bit 0 = Decode LED See your product's Product Reference Guide for further bit information.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

The host sends this message to turn off the specified decoder LEDs.

Host Requirements

None.

Decoder Requirements

The decode LED is turned off by the decoder.

LED_ON

Description

Activates LED output.

Packet Format

Length	Opcode	Message Source	Status	LED Selection	Checksum
05h	E7h	04h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E7h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
LED Selection	Bit 0 - 7: LED bit numbers to turn on.	1 Byte	Bit 0 = Decode LED See your product's Product Reference Guide for further bit information.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

The host sends this message to turn on the specified decoder LEDs.

Host Requirements

None.

Decoder Requirements

The decode LED is turned on by the decoder.

PARAM_DEFAULTS

Description

Sets the parameters to their default values.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	C8h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	C8h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This command returns all parameters to their default settings.

Host Requirements

The host sends this command to reset the decoder's parameter settings to the default values.

Decoder Requirements

Upon receiving this command, the decoder resets all its parameters to the default values. This is equivalent to scanning a SET DEFAULTS bar code.

PARAM_REQUEST

Description

Requests values of selected parameters.

Packet Format

Length	Opcode	Message Source	Status	Request Data	Checksum
	C7h	04h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	C7h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Request Data	<Param_num><Param_num><Param_num>...	Variable	
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

The host uses this message to request selected parameters from the decoder.

Host Requirements

The host requests the decoder's current values for specific parameters by listing the parameter numbers in the Request_Data field. If the host asks for a parameter value not supported by the decoder, the decoder does not send a value for this unsupported param_num. If none of the requested values is supported, an empty PARAM_SEND message is transmitted. If the host requests the value of all the parameters, it sends a special param_num called ALL_PARAMS (FEh) in the first position of the Request_Data field.

NOTE The decoder's response to this command is PARAM_SEND, not ACK. Depending on the time-out set, and the number of parameters requested, this reply may fall outside the programmable Serial Response Timeout. It should not be considered an error if the time-out is exceeded. To compensate, increase the time-out.

Decoder Requirements

When the decoder receives this message, it processes the information by formatting a PARAM_SEND message containing all requested parameters that are supported, and their values. The programmable Serial Response Time-out may be exceeded when processing this message, depending on the time-out set, and the number of parameters requested.

Hints for Requesting Parameter Values

Before forming a PARAM_REQUEST, be sure you are requesting parameters supported by the decoder (<Blue>Table A-12). To find out what parameters are supported, send an FEh (request all parameters). The decoder responds with a PARAM_SEND which contains all the supported parameters and their values. This response may be multipacketed; ACK responses are not necessary.

Table A-12. Example of Supported Parameter Numbers

Supported Parameter Number	Associated Parameter Values
01	00
02	01
9C	07
E6	63

When using the FEh, it must be in the first position of the request_data field, or it is treated as an unsupported parameter.

Unsupported parameters are not listed in the PARAM_SEND response. Requesting unsupported parameters has no effect, but can cause delays in responding to requests for valid parameters. See <Blue>Table A-13 for example requests and responses.

Table A-13. Example Requests and Replies

PARAM_REQUEST message		Response PARAM_SEND message
#ALL	05 C7 04 00 FE FE 32	0D C6 00 00 FF 01 00 02 01 9C 07 E6 63 FC 3E
#1, 9C	06 C7 04 00 01 9C FE 92	09 C6 00 00 FF 01 00 9C 07 FD 8E
#All, 1, 9C	07 C7 04 00 FE 01 9C FD 93	0D C6 00 00 FF 01 00 02 01 9C 07 E6 63 FC 3E
#1, 9C, ALL	07 C7 04 00 01 9C FE FD 93	09 C6 00 00 FF 01 00 9C 07 FD 8E

Table A-13. Example Requests and Replies (Continued)

PARAM_REQUEST message		Response PARAM_SEND message
#4	05 C7 04 00 04 FF 2C	05 C6 00 00 FF FE 36
#ALL - 3 times	07 C7 04 00 FE FE FE FC 34	0D C6 00 00 FF 01 00 02 01 9C 07 E6 63 FC 3E
#1 -3 times	07 C7 04 00 01 01 01 FF 2B	0B C6 00 00 FF 01 00 01 00 01 00 FE 2D

PARAM_SEND

Description

Responds to a PARAM_REQUEST, changes particular parameter values.

Packet Format

Length	Opcode	Message Source	Status	Beep Code	Param data	Checksum
	C6h					

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	C6h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder 4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Beep code	See <Xref>Table A-4 on page A-11.	1 Byte	If no beep is required, set this field to FF.
Param_data	See <Xref>Table A-14 on page A-44.		The parameter numbers and data to be sent to the requester.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This message is sent by the decoder in response to the PARAM_REQUEST message, or by the host to change the decoder's parameter values.

Parameter numbers F0h (+256), F1h (+512), F2h (+768) access parameters whose numbers are 256 and higher. For example, to access the first parameter in the 256-511 range, use F0h and 00h.

Table A-14. Param Data Format

Parameter Number	Data Format
0 through EFh	<param_num> <value>
F0h, F1h, F2h	<extended parameter code> <param_num offset> <value>

Host Requirements

NOTE Due to the processing time of interpreting and storing parameters contained in the message, it may not be possible for the decoder to send an ACK within the programmable Serial Response Timeout. It should not be considered an error if the time-out is exceeded. To compensate, increase the time-out.

The host transmits this message to change the decoder's parameters. Be sure the Change Type bit in the Status byte is set as desired. If no beep is required, the beep code must be set to FFh, or the decoder beeps as defined in <Blue>Table A-4.

Decoder Requirements

When the decoder receives a PARAM_SEND, it interprets and stores the parameters, then ACKs the command (if ACK/NAK handshaking is enabled). These parameters are stored permanently only if the Change Type (bit 3 of the Status byte) is set to 1. If bit 3 is set to 0 the changes are temporary, and are lost when the decoder is powered down.

If the PARAM_SEND sent by the host contains a valid beep code, the decoder issues the requested beep sequence, and changes the requested parameter values.

The decoder issues a PARAM_SEND in response to a PARAM_REQUEST from the host. It sends the values for all the supported parameter values requested in the PARAM_REQUEST message. No value is sent for any unsupported param_num. If none of the requested values is supported, the PARAM_SEND message is transmitted with no parameters. When sending this command, the Change Type bit (bit 3 of Status byte) can be ignored.

NOTE For multipacketed PARAM_SEND, the beep code appears in every packet.

REPLY_REVISION

Description

Replies to REQUEST_REVISION command with software revision string.

Packet Format

Length	Opcode	Message Source	Status	Revision	Checksum
	A4h	00h		S/W_REVISION <space> BOARD_TYPE <space> ENGINE_CODE <space> PGM_CHKSUM	

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	A4h	1 Byte	Identifies this Opcode type.
Message Source	0 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Revision	ASCII data	variable	Revision String fields indicate: **S/W_REVISION** is the release name of the software **BOARD_TYPE** is N for non-flash decoder board, F for flash **ENGINE_CODE** indicates the type of scan engine paired with the decoder (see the scan engine's Integration Guide for the engine code value) **PGM_CHKSUM** is the two byte checksum of the program code
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

None.

Decoder Requirements

The decoder sends its revision string to the host. The revision string is decoder-dependent.

REQUEST_REVISION

Description

Requests the software revision string from the decoder.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	A3h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	A3h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

The host sends this message to request revision information from the decoder. The decoder responds with REPLY_REVISION.

Decoder Requirements

The decoder sends its revision string to the host. See *"REPLY_REVISION,"* for format.

SCAN_DISABLE

Description

Prevents the decoder from scanning bar codes.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	EAh	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	EAh	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

All scan attempts are disabled by this command until either a SCAN_ENABLE is sent, or the decoder is reset.

Decoder Requirements

When the decoder receives this command, it ignores all trigger/START_SESSION requests until a SCAN_ENABLE command is received.

SCAN_ENABLE

Description

Permits the decoder to scan bar codes.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	E9h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E9h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

The host sends the SCAN_ENABLE command to tell the decoder to allow scanning. Scanning is enabled upon power-up, so this command need only be send if a prior SCAN_DISABLE command has been sent.

Decoder Requirements

The decoder allows scanning and decoding upon receipt of this command.

NOTE At initial power-up, the decoder should assume SCAN_ENABLED.

SLEEP

Description

Requests to place the decoder into low power mode.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	EBh	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	EBh	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

Host Requirements

The host sends this command to place the decoder into low power mode. If the low power mode parameter is enabled, the scanner goes into low power mode automatically, and the SLEEP command is not necessary.

NOTE The decoder may not sleep immediately upon acknowledging the command, as it may be busy processing data at the time.

Decoder Requirements

None.

START_SESSION

Description

Tells decoder to attempt to obtain the requested data.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	E4h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E4h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This command tells the decoder to start a scan session. See <Blue>Table A-15 for the decoder's reactions to this command in each operational mode.

Table A-15. START_SESSION Actions

Operational Mode	Actions upon Receipt of Command	End Result of Session
Decode Mode	The decoder attempts to decode a bar code	Successful decode, or STOP_SESSION command
Image Capture	The decoder clicks the shutter	An image is captured
Video Mode	The decoder continuously produces a video stream	STOP_SESSION command

Decoder Requirements

Trigger Mode must be set to Host or a CMD_NAK DENIED response is issued.

STOP_SESSION

Description

Tells decoder to abort a decode attempt or video transmission.

Packet Format

Length	Opcode	Message Source	Status	Checksum
04h	E5h	04h		

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	E5h	1 Byte	Identifies this Opcode type.
Message Source	4 = Host	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

This command tells the decoder to stop a scan and decode attempt.

Host Requirements

Multipacketing option 2 or 3 should be selected to achieve reasonable throughput.

Decoder Requirements

None.

VIDEO_DATA

Description

Imager transmission of a video frame in JPEG format.

Packet Format

Length	Opcode	Message Source	Status	Data	Checksum
	B4h	00h			

Field Descriptions

Field Name	Format	Size	Description
Length	Length of message (not including checksum).	1 Byte	Length Field
Opcode	B4h	1 Byte	Identifies this Opcode type.
Message Source	00 = Decoder	1 Byte	Identifies where the message is coming from.
Status		1 Byte	**Bit 0: Retransmit** 0 = First transmission 1 = Subsequent transmission **Bit 1: Continuation** 0 = Last packet of a multipacket message 1 = Intermediate packet **Bit 2: Reserved** Always 0 **Bit 3: Parameter Change Type** (for parameters) 0 = Temporary change 1 = Permanent change
Data		Up to 251 Bytes	Image data.
Checksum	2's complement sum of message contents excluding checksum.	2 Bytes	Checksum of message.

The first packet of a video frame contains the video preamble, described below. The first packet also contains the JPEG data comprising the video frame. Multipacketing is expected in video mode.

The video preamble consists of the following fields:

Table A-16. Video Preamble Fields

Field	Field Size	Description
File size	4 byte field	Number of bytes in the overall image.
Image Width	2 byte field	Image width in pixels
Image Height	2 byte field	Image height in pixels

Table A-16. Video Preamble Fields (Continued)

Field	Field Size	Description
Image Type	1 byte field	0x31 = JPEG Image File 0x33 = BMP Windows Bit Map File 0x34 = TIFF File Note: These values are ASCII.
Bits per Pixel	1 byte field	Number of bits per pixel in image 0 = 1 bit/pixel Black White Image 1 = 4 bit/pixel 16 Grayscale Image 2 = 8 bit/pixel 256 Grayscale Image

In a multipacketed environment, one video frame is spread over several packets in the following format:

Packet 1

Header	Preamble	Video Data, Part 1	Checksum

Packet 2

Header	Video Data, Part 2	Checksum

.

.

.

Packet N

Header	Last of Video Data	Checksum

This is re-assembled by the host into:

Preamble	Video Frame

WAKEUP

Description

Wakes up decoder after it's been put into low power operation.

If the decoder is in low power mode, sending the single character **NULL** (00) wakes up the decoder. This character is only needed when hardware handshaking is not being used or is bypassed.

Host Requirements

Once the WAKEUP character is sent, the host must wait a period of time to permit the decoder to wake up. The decoder remains awake for a fixed period of time after wake up. These time periods vary by decoder.

Decoder Requirements

The decoder must not go back into low power mode for a decoder-dependent time period after waking up.

NOTE The mechanism to wake up a decoder in this manner also works if characters other than WAKEUP are sent to the decoder. There is, however, no guarantee that these commands are interpreted correctly upon power-up. Therefore, it is not recommended that characters other than WAKEUP be used to awaken the decoder.

The WAKEUP character has no effect if sent when the scanner is awake. If the host is unsure of the scanner state, it should send the wakeup character when it wants to communicate with the scanner.

Index

E

H

I

K

M

N

O

P

S

T

U

V

Motorola, Inc.
One Motorola Plaza
Holtsville, New York 11742, USA
1-800-927-9626
http://www.motorola.com/enterprise

72E-50705-01 Revision C - June 2008