Simple Serial Interface (SSI) Software Developer’s Kit

Programmer Guide

Simple Serial Interface (SSI) Software Developer's Kit Programmer Guide

72E-59860-03
Revision A
September 2007

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

Revision History

Changes to the original manual are listed below:

Change	Date	Description
-01 Rev A	12/2002	Initial Rev A Release.
-02 Rev A	1/2003	Add missing information.
-03 Rev A	9/2007	Add UpdateFirmware and AbortFWUpdate APIs, add new API return value error codes, add WM_FW_UPDATE_PROGRESS, WM_FW_UPDATE_BAUD, and WM_FW_UPDATE_FAILED Windows messages.

Table of Contents

About This Guide

Introduction

The *Simple Serial Interface (SSI) Software Developer's Kit Programmer Guide* describes the Application Programming Interface (API) to the SSIDLL, which provides a communications link between Motorola decoders and a Windows 95/98/2000/XP host using the serial port. This guide also provides instructions for adding and using the ActiveX control, which facilitates using the SSI DLL within Visual Basic.

Chapter Descriptions

- *Chapter 1, Simple Serial Interface (SSI) API* describes the SSIDLL, developed using Microsoft Visual C++ V6.0. The DLL implements serial communications, reader and writer threads, SSI message building and the SSI protocol.
- *Chapter 2, ActiveX Control* provides instructions for adding the SSIConnect.ocx component to a Visual Basic project. Refer to the on-line help for specific information on using the ocx and its properties, methods and events.

Notational Conventions

The following conventions are used in this document:

- "User" refers to anyone using an SSI compatible 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
  - Specific items in the general text
  - Dialog boxes and tabs within dialog boxes.

- bullets (•) indicate:
  - Action items
  - Lists of alternatives
  - Lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Documents

- *Universal Scan Engine Developer's Kit Installation Guide*, p/n 72-59636-xx
- *Simple Serial Interface (SSI) Programmer's Guide*, p/n 72-40451-xx
- *Simple Serial Interface (SSI) Developer's Guide*, p/n 72-50705-xx
- The *Product Reference Guide* or *Integration Guide* for your scanner for product-specific information on SSI.

For the latest versions of these guides go to: http://support.symbol.com.

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.symbol.com/contactsupport. If you purchased your Enterprise Mobility business product from a Motorola business partner, contact that business partner for support.

Before contacting, have the model number and serial number at hand. 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.

Chapter 1 Simple Serial Interface (SSI) API

Introduction

This implementation of SSI (Simple Serial Interface) uses handshaking (RTS/CTS) to communicate with an SSI device. These signals are required and must be connected. The SSI DLL implements serial communications, reader and writer threads, SSI message building, and the SSI protocol handling needed to provide a communications link between Motorola decoders and a Windows host. To set the host communication option to SSI, scan the bar code parameter for SSI Host.

SSI is a transaction-based protocol. After a command function is called, control returns to the host application while the scanner processes the command. After the command is processed by the scanner, the host application receives a Windows message indicating the command was processed. The host application should provide a message handler for the acknowledgement from the connected SSI device before initiating another command.

The Windows host program also receives Windows messages when the decoder has data to send to the host or when a timeout or error occurs. The Windows host program provides data storage for the DLL to use for returning scanner data to the application.

All function prototypes and #defines can be found in SSIdll.h.

API Descriptions

SSIConnect

Description

This must be the first call to the library. It opens the COM port using the indicated baud rate, COM port number, and handle of the window whose procedure will receive Windows messages from the library. No command is sent to the scanner during this API call. Unless a call to disable the scanner is issued, the scanner may send decode data at any time after it is connected. Therefore, the host application should call the API function *SetDecodeBuffer* after a successful call to SSIConnect if it wants to handle unsolicited decode data from the scanner. Also, all data, including Decode Data, must be sent using ACK/NAK handshaking. If this is not the default setting for your scanner, use bar code parameters or call the API function *SetParameters*() to set the Decode Data Packet parameter to use ACK/NAK protocol.

Syntax

```
SSIDLL_API int __stdcall SSIConnect(HWND hwnd, long Baud, int Port);
```

where:

- *hwnd* is the handle of the window whose procedure will receive Windows messages from the library.
- *Baud* is the baud rate to use
- *Port* is the COM port number to open

Return Values

- SSICOMM_NOERROR if the COM port was opened successfully.
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;

// using baud rate of 9600, com port number  1
status = SSIConnect(m_hWnd, 9600, 1);
```

SSIDisconnect

Description

Terminates the communications link and releases all memory used by the library. No command is sent to the scanner during this API call. Always call this function when the application is finished communicating with the decoder.

Syntax

```
SSIDLL_API int __stdcall SSIDisconnect(HWND hwnd, int nComPort);
```

where:

- *hwnd* is the handle of the window used during the call to *SSIConnect* for this COM port
- *Port* is the COM port number to close

Return Values

- SSICOMM_NOERROR if the Com Port was disconnected successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = SSIDisonnect(m_hWnd,  1); // using com port number  1
```

AbortFWUpdate

Description

Aborts the update firmware process on the scanner. Following this call, the message WM_FW_UPDATE_FAILED is issued with lparam set to FW_UPDATE_ABORTED.

Syntax

```
SSIDLL_API int __stdcall AbortFWUpdate(HWND hwnd,int nComPort);
```

where:

- *hwnd* is the handle of the window whose procedure receives windows messages from the library
- *nComPort* is the COM port number to use

Return Values

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define COM_PORT0x01//COM port number

int status;
int nComPort= COM_PORT;

status = AbortFWUpdate(m_hWnd, nComPort);
```

AbortImageXfer

Description

Tells the scanner to stop transmitting the image being sent.

Syntax

```
SSIDLL_API int __stdcall AbortImageXfer(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Values

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = AbortImageXfer ( 1); // using com port number  1
```

AbortMacroPdf

Description

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

Syntax

```
Description SSIDLL_API int __stdcall AbortMacroPdf(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = AbortMacroPdf( 1); // using com port number  1
```

AimOn/AimOff

Description

Sends the Aim On or Aim Off command to the scanner. *AimOn* turns on the aiming pattern of an imager, *AimOff* turns it off.

Syntax

```
SSIDLL_API int __stdcall AimOn(int nComPort);
SSIDLL_API int __stdcall AimOff(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = AimOn( 1); // using com port number  1
```

EnterLowPwrMode

Description

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

Syntax

```
SSIDLL_API int __stdcall EnterLowPwrMode(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = EnterLowPwrMode( 1); // using com port number  1
```

FlushMacroPdf

Description

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

Syntax

```
SSIDLL_API int __stdcall FlushMacroPdf(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = FlushMacroPdf( 1); // using com port number  1
```

LedOn/LedOff

Description

Sends a command to the scanner to turn on or turn off the LED. *LedOn* turns on the LED of the scanner, and *LedOff* turns it off. Which LED to turn on or off is specified as a bit in the *nLEDselection* parameter.

Syntax

```
SSIDLL_API int __stdcall LedOn(int nComPort, unsigned char nLEDselection);
SSIDLL_API int __stdcall LedOff(int nComPort, unsigned char nLEDselection);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *nLEDselector* is the bitwise indicator for the LED to be turned on or off (scanner-dependent)

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
unsigned char decode_led = 0x02; // assumes led is represented using bit 1

status = LedOn( 1, decode_led); // using com port number  1, turn on led
```

PullTrigger

Description

Sends a command to the scanner to perform a software trigger, causing the scanner to behave as if the trigger were pulled. Some scanners require setting the trigger mode to host mode. To do this, call *SetParameters* before the *PullTrigger* API function. See the documentation for your scanning device.

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

Syntax

```
SSIDLL_API int __stdcall PullTrigger(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = PullTrigger( 1); // using com port number  1
```

ReleaseTrigger

Description

Sends a command to the scanner to release the software trigger. On the VS 4004 in video mode (or decode mode if no bar code was decoded), the application must call *ReleaseTrigger*, after a call to *PullTrigger*. Laser-based decoders timeout automatically.

Call *ReleaseTrigger* to abort a decode attempt or video transmission.

Syntax

```
SSIDLL_API int __stdcall ReleaseTrigger(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = ReleaseTrigger( 1); // using com port number  1
```

RequestAllParameters

Description

Sends a request to the scanner to return all its parameters and their values. This call must be preceded by a call to *SetParameterBuffer* to provide a buffer for the data that will be returned from the scanner. After the scanner responds to the request and the DLL stores the parameter data, a Windows message is sent to the host application indicating that the data is available.

Syntax

```
SSIDLL_API int __stdcall RequestAllParameters(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define PARAM_RETURN_DATA_LEN 2000
//globally defined data storage
unsigned char ParamReturnData[PARAM_RETURN_DATA_LEN];

int status;
// Give the DLL a buffer to use when the scanner returns the parameter data
SetParameterBuffer(1, ParamReturnData, PARAM_RETURN_DATA_LEN);
// using com 1
status = RequestAllParameters( 1); // using com port number  1

// If status is good, the request was sent. Later, when the DLL receives the
// data back from the scanner, the data will be stored in the ParamReturnData
// buffer and the application will receive a windows message.
```

RequestParameters

Description

Requests the scanner to send parameter values specified in the given parameter string. This call must be preceded by a call to *SetParameterBuffer* to provide a buffer for the data that will be returned from the scanner. After the DLL receives the parameter data from the scanner, it sends the host application a Windows message indicating that the parameter data is available.

Syntax

```
SSIDLL_API int __stdcall RequestParameters(unsigned char *Params, int ParamBytes, int nComPort);
```

where:

- *Params* is a buffer of byte values which specifies the parameter numbers whose values are being requested from the scanner. The format of the parameter number is either:
  - *<param_num>* if the parameter number is in the range 0..EFh; or
  - *<extended parameter code><parm_num_offset>* for parameters whose param_num is 256 or higher.

  Refer to the documentation for the scanning device for parameter numbers.
- *nParamBytes* is the number of bytes in the Params buffer.
- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define SWTRIG_PARAMNUM0x8a  // software trigger parameter number
#define EXTENDED_PARAMNUM0xf0  // specify an extended parameter number
#define IMAGE_FILETYPE_PARAMNUM0x30
#define PARAM_RETURN_DATA_LEN 2000
//globally defined data storage
unsigned char ParamReturnData[PARAM_RETURN_DATA_LEN];

int status;
unsigned char Params[3];
Params[0] = SWTRIG_PARAMNUM; // get the software trigger setting
Params[1] = EXTENDED_PARAMNUM; // use extended param number for
Params[2] = IMAGE_FILETYPE_PARAMNUM // …image filetype setting
// Give the dll a buffer to use when the scanner returns the parameter data
// using com 1
SetParameterBuffer(1, ParamReturnData, PARAM_RETURN_DATA_LEN);
// Send the request which is stored in Params and is 3 bytes long
status = RequestParameters( Params, 3, 1); // using com port number  1
// If status is good, the request was sent. Later, when the DLL receives the
// data back from the scanner, the data will be stored in the ParamReturnData
// buffer and the application will receive a windows message.
```

RequestScannerCapabilities

Description

Sends a request to the scanner to send its capabilities data, i.e., the commands it can perform. This call must be preceded by a call to *SetCapabilitiesBuffer*. When the scanner responds to this command with its capabilities data, the DLL sends a Windows message to the host application indicating that capabilities data is stored.

Syntax

```
SSIDLL_API int __stdcall RequestScannerCapabilities(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define CAPABILITIES_RETURN_DATA_LEN 255
//globally defined data storage
unsigned char CapabilitesReturnData[CAPABILITIES_RETURN_DATA_LEN];

int status;
// Give the dll a buffer to use when the scanner returns the  data
// using com 1
SetCapabilitiesBuffer(1, CapabilitesReturnData, CAPABILITIES_RETURN_DATA_LEN);
status = RequestScannerCapabilities( 1); // using com port number  1
// If status is good, the request was sent. Later, when the DLL receives the
// data back from the scanner, the data will be stored in the
// CapabilitiesReturnData buffer and the application will receive a windows
// message.
```

ReturnDLLVersion

Description

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

Syntax

```
SSIDLL_API unsigned int __stdcall ReturnDLLVersion(void);
```

Return Value

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

Example

```
unsigned int version, major, minor;
CString msg;

version = ReturnDLLVersion();

major = (version & 0x0000ff00) >> 8;
minor = version & 0x000000ff;

msg.Format("Library Version %d.%d", major, minor);
```

ScanEnable/ScanDisable

Description

Sends a command to the scanner to enable or disable the scanner. *ScanEnable* enables scanning, while *ScanDisable* disables scanning. When scanning is disabled, the scanner does not respond to a physical or software trigger pull.

Syntax

```
SSIDLL_API int __stdcall ScanEnable(int nComPort);
SSIDLL_API int __stdcall ScanDisable(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = ScanEnable( 1); // using com port number  1
```

SetCapabilitiesBuffer

Description

Allows the application to specify the address and the length of a buffer for the DLL to use to store the capabilities data from the scanner. Set the capabilities data buffer immediately before a call to *RequestScannerCapabilities*. No command is sent to the scanner during this API call. The amount of data returned is variable; a buffer of length 256 should be sufficient.

Syntax

```
SSIDLL_API int __stdcall SetCapabilitiesBuffer(int nComPort, unsigned char *pData, long max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for capabilities data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define CAPABILITIES_DATA_LEN 255
//globally defined data storage
unsigned char CapabilitiesReturnData[CAPABILITIES_DATA_LEN];

int status;

status = SetCapabilitiesBuffer( 1, CapabilitiesReturnData, MAX_DATA_LEN);
// using com port number  1
```

SetDecodeBuffer

Description

Sets the decode data buffer and its length for the DLL to use to store decode data from the scanner. No command is sent to the scanner during this API call.

The length of decode data depends on the type of bar code scanned; if MacroPDF is buffered, large amounts of data are possible. When the DLL has decode data from the scanner, this buffer is filled and a Windows message is sent to the application to indicate that decode data is stored. The host application must then call *SetDecodeBuffer* again in order to receive additional decode data from the scanner.

Syntax

```
SSIDLL_API int __stdcall SetDecodeBuffer(int nComPort, unsigned char *pData, long max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for decode data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define DECODE_DATA_LEN 255
//globally defined data storage
unsigned char DecodeData[DECODE_DATA_LEN];

int status;

status = SetDecodeBuffer( 1, DecodeData, DECODE_DATA_LEN);
// using com port number  1
```

SetImageBuffer

Description

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

Call the *SetimageBuffer* function when the first image transfer status message is sent to the application, which holds the length information for the entire image.

Syntax

```
SSIDLL_API int __stdcall SetImageBuffer(int nComPort, unsigned char *pData, long max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for image data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
afx_msg LRESULT CSSIappView::OnSSIxferStatus(WPARAM w, LPARAM l)
{
    long running_total = (long)w;
    long expected_total = (long)l;

    // the buffer will not be needed until the image transfer has completed
    // images come in packets of 255 bytes (last packet may be less)
    if(running_total <= 255)  // the start of an image transfer
    {
        g_pImageData = new BYTE[expected_total + 1];
        if(g_pImageData != NULL)
        SetImageBuffer(Comport, g_pImageData, expected_total);
        // now  DLL has a place to put the image when done
    }
}
```

SetParameterBuffer

Description

Sets the user's destination buffer and its length for the DLL to use to store parameter data from the scanner. No command is sent to the scanner during this API call. Set the parameter data buffer immediately before calling *RequestParameters* or *RequestAllParameters*. A size of 2000 bytes should be sufficient to hold all the parameter number/value pairs. A call for a single parameter only requires a small buffer: 10 bytes is sufficient.

Syntax

```
SSIDLL_API int __stdcall SetParameterBuffer(int nComPort, unsigned char *pData, long       max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for parameter data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define PARAM_RETURN_DATA_LEN 2000
//globally defined data storage
unsigned char ParamReturnData[PARAM_RETURN_DATA_LEN];

int status;

// using com port number  1
status = SetParamterBuffer( 1, ParamReturnData, PARAM_RETURN_DATA_LEN);
```

SetParameters

Description

Commands the scanner to change one or more of the scanner's parameter values. The format of the parameters and their values is either:

- *<param_num><value>* if param num is in the range 0..EFh; or
- *<extended parameter code><parm_num_offset><value>* for parameters whose param_num is 256 or higher

if *<value>* is a two byte value rather than a one byte value, the param num/value pair is preceeded by the hex value 0xF4

Syntax

```
SSIDLL_API int __stdcall SetParameters(unsigned char *Params, int ParamBytes, int nComPort);
```

where:

- Params is a pointer to the param_num/value data
- ParamBytes is the size in bytes of the data stored in Params
- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define IMAGE_FILETYPE_PARAMNUM 0x30
// this specifies an extended parameter number
#define EXTENDED_PARAMNUM0xf0
#define BITMAP_TYPE 0x03
#define TRIGGER_MODE 0x8a
#define HOST_TRIGGER_TYPE0x08

int status;
unsigned char Params[5];
Params[0] = EXTENDED_PARAMNUM; // use extended param number for
Params[1] = IMAGE_FILETYPE_PARAMNUM; //…image file type
Params[2] = BITMAP_TYPE;     // this is the value for the image filetype param
Params[3] = TRIGGER_MODE; // now set a second parameter for trigger mode
Params[4] = HOST_TRIGGER_TYPE;  // and it's value to host

// 5 bytes were stored and we are using com port number  1
status = SetParamters( Params, 5, 1);
```

NOTE To enable ACK/NAK handshaking for decode data on the SE 1223 use 0xEE (decimal 238) for the parameter number, and use 1 for the value.

SetParamPersistance

Description

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

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

Syntax

```
SSIDLL_API int __stdcall SetParamPersistance(int nComPort, int bPersist);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *bPersist* is set to TRUE if persistance is desired, FALSE if not

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#int status;
status = SetParamPersistance( 1, FALSE); // using com port number  1
```

SetVersionBuffer

Description

Sets the user's destination buffer and its length for the DLL to use to store version data from the scanner. No command is sent to the scanner during this API call. Set the version data buffer immediately before calling *TransmitVersion*. The amount of data returned is variable; a buffer of length 256 should be sufficient.

Syntax

```
SSIDLL_API int __stdcall SetVersionBuffer(int nComPort, unsigned char *pData, long max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for version data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define VERSION_DATA_LEN 255
//globally defined data storage
unsigned char VersionData[VERSION_DATA_LEN];

int status;

// using com port number  1
status = SetVersionBuffer( 1, VersionData, VERSION_DATA_LEN);
```

SetVideoBuffer

Description

Sets the user's destination buffer and its length for the DLL to use to store video data from the scanner. No command is sent to the scanner during this API call. Video data is sent continuously when the scanner is in video mode, so when the application receives the Windows message notifying it that Video data has been stored, the host program should process the stored video frame then call *SetVideoBuffer* again to set the destination for the next frame. 5000 bytes is a sufficient video buffer size.

Syntax

```
SSIDLL_API int __stdcall SetVideoBuffer(int nComPort, unsigned char *pData, long max_length);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *pData* is a pointer to the destination buffer for video data returned from the scanner
- *max_length* is the size in bytes of the destination buffer

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define VIDEO_DATA_LEN 5000
//globally defined data storage
unsigned char VideoData[VIDEO_DATA_LEN];

int status;

// using com port number  1
status = SetVideoBuffer( 1, VideoData, VIDEO_DATA_LEN);
```

SnapShot

Description

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

If the trigger is pulled, the image data is sent in packets to the DLL. As each packet is received, a WM_XFERSTATUS is sent to the host application with information about the size of the image. When the first transfer status message is received, the host application should provide a destination buffer for the image by calling *SetImageBuffer*. After the entire image is transferred from the scanner to the DLL, the application receives a Windows message indicating that the data was stored.

Syntax

```
SSIDLL_API int __stdcall SnapShot(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = SnapShot( 1); // using com port number  1
```

SoundBeeper

Description

Sends a command to the scanner to turn the beeper on. See *Table 1-5 on page 1-35* for beep codes.

Syntax

```
SSIDLL_API_stdcall SoundBeeper(in nComPort, unsigned char nBeepCode);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*
- *nBeepCode* specifies the tone and duration for the beep

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
int status;
status = SoundBeeper( 1, ONESHORTHI); // using com port number  1
```

TransmitVersion

Description

Sends a request to the scanner to send its software release name. Call *SetVersionBuffer* before this call to provide a destination buffer for the version data when it is sent by the scanner. After the scanner responds with the version data, the DLL sends the host application a Windows message indicating that the version data is available.

Syntax

```
SSIDLL_API int __stdcall TransmitVersion(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define VERSION_DATA_LEN 255
// globally defined data storage
unsigned char VersionData[VERSION_DATA_LEN];

int status;

status = SetVersionBuffer( 1, VersionData, VERSION_DATA_LEN); // using com port number  1
// check status then…
status = TransmitVersion(1); ); // using com port number  1
```

TransmitVideo

Description

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

✓ ***NOTE*** The VS 4004 does not currently support this.

Syntax

```
SSIDLL_API int __stdcall TransmitVideo(int nComPort);
```

where:

- *nComPort* is the COM port number used in the call to *SSIConnect*

Return Value

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

UpdateFirmware

Description

Updates firmware on the scanner including firmware file processing. This function negotiates the baud rate with the scanner, so actual firmware update may occur at a higher baud rate than that set by the user. During firmware update, this function issues messages indicating update progress (WM_FW_UPDATE_PROGRS), failure (WM_FW_UPDATE_FAILED), or negotiated baud rate (WM_FW_UPDATE_BAUD). It returns SSICOMM_NOERROR for success, or a proper error code.

While the API always returns to SSI mode, after calling this function you must call **SSIDisconnect()** to disconnect the scanner, scan the appropriate **SSI Host Types** bar code in the scanner's *Product Reference Guide* to set the scanner to SSI mode, then call **SSIConnect()** to re-connect. In order to receive decode data from the scanner, you may also need to set the **Decode Data Packet Format** parameter to **Send Packeted Decode Data** in SSI Host Preferences in the scanner's *Product Reference Guide*.

Syntax

```
SSIDLL_API int __stdcall UpdateFirmware(HWND hwnd,int nComPort,char* file_path)
```

where:

- *hwnd* is the handle of the window whose procedure receives windows messages from the library
- *nComPort* is the COM port number to use
- *file_path* is the pointer to the buffer which contains the path to the firmware update file

Return Values

- SSICOMM_NOERROR if the command was sent successfully
- An error code if an error occurred (see *Table 1-1 on page 1-32*)

Example

```
#define COM_PORT      0x01      //COM port number

int status;
int nComPort= COM_PORT;
char file_path[]="c:\\firmware_file.dat";

status = UpdateFirmware(m_hWnd, nComPort, file_path);
```

Library Error Reporting

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

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

API Return Values

Table 1-1 *Error Codes (Return Values for API Calls)*

Define Name	Value	Description
SSICOMM_NOERROR	0	No error code is set; an API call was successful.
ERR_SSI_NOOBJECT	-1	Another API function is called before a successful call to *SSIConnect*; no connection established.
ERR_SSI_HWND	-2	The hwnd parameter to the SSIConnect function was NULL; no connection established.
SSICOMM_BAD_SETSTATE	-3	The library was unable to set the state of the COM port; no connection established.
SSICOMM_BADSETTIMEOUTS	-4	The library was unable to set the COM timeouts; no connection established.
SSICOMM_BAD_GETTIMEOUTS	-5	The library was unable to get the current COM timeouts; no connection established.
SSICOMM_BAD_GETCOMSTATE	-6	The library was unable to get the current COM state; no connection established.
SSICOMM_ALREADY_CLOSED	-7	Call to close COM port was made when the COM port is not open; there is no connection.
SSICOMM_UNABLE_PURGE	-8	Call to purge the COM port before closing it was not successful.
SSICOMM_THREADS_BADEXIT	-9	Fatal error - the threads didn't exit properly.
SSICOMM_ERROR_CLRDTR	-10	Unable to lower DTR when closing COM port.
SSICOMM_BAD_CREATEFILE	-11	Unable to open COM port.
SSICOMM_BAD_READTHREAD	-12	Unable to create the read/status thread; no connection.
SSICOMM_BAD_WRITETHREAD	-13	Unable to create the writer thread; no connection.
SSICOMM_BAD_CREATEEVENT	-14	Call to *CreateEvent* failed; fatal error.
SSICOMM_BUSY	-15	Not fatal; SSI driver is busy processing data.
SSICMD_UNIMPLEMENTED	-16	Not fatal; this command is not implemented in the library.
SSICOMM_ALREADYCONNECTED	-17	If already connected, can't connect without a call to disconnect.

Table 1-1 *Error Codes (Return Values for API Calls) (Continued)*

Define Name	Value	Description
ERR_SSI_MISMATCHHWND	-18	The hwnd parameter for the function does not match the stored hwnd for the connection.
SSICOMM_TOOMUCHDATA	-19	The maximum allowable input data length was exceeded.
SSICOMM_ERRVERSION	-20	Can't run on this version of Windows.
SSI_INPUTQ_FULL	-21	Unable to add new user request to input queue for transmitting to scanner; re-try request.
SSICOMM_BADDATA	-22	Parameter data is in incorrect format.
SSIFW_UPDATE_FAILED	-40	Generic error accessing firmware update function.
SSIFW_FILE_OPEN_FAIL	-41	Firmware file could not be opened.
SSIFW_INVALID_CHECKSUM	-42	Checksum calculation for firmware file failed.
SSIFW_COM_PORT_FAIL	-43	Unable to open COM port.
SSIFW_FW_UPDATE_MODE_FAIL	-44	Failed to enter firmware update mode.
SSIFW_PROG_COM_PORT_FAIL	-45	Unable to set requested baud rate on the COM port.
SSIFW_FILE_READ_FAIL	-46	Unable to read records from firmware file.
SSIFW_TX_FAIL	-47	Failure in serial transmission.
SSIFW_RX_FAIL	-48	Failure in serial receiving.
SSIFW_INVALID_REC	-49	Checksum calculation for the record failed.
SSIFW_UPDATE_FAILED_IN_SCN	-50	Failure occurred while updating firmware.
SSIFW_UPDATE_ABORTED	-51	User aborted firmware update process.
SSIFW_UPDATE_INPROGRESS	-52	Firmware update already in progress.
SSIFW_ABORT_FAILED	-53	Failed to acquire SSI handle.

SSI WM_ERROR Messages

The following generate WM_ERROR messages to the calling application. Most are fatal errors occurring during program execution, and are returned in the WPARAM associated with the WM_ERROR message.

Table 1-2 *WM_ERROR Messages*

Define Name	Value	Description
SSICOMM_WAITMOWRITER	-23	Wait for Multiple Objects resulted in WAIT_FAILED in writer procedure; if not fatal, protocol retry may recover.
SSITHREAD_CREATEWEVENT	-24	Failure to create write event; fatal error.
SSITHREAD_OLRESW	-25	Get overlapped result failed; fatal error.
SSITHREAD_WRITEERR	-26	Number of bytes written is not the number requested to be written; if not fatal, retry may recover.
SSITHREAD_WMOW	-27	Wait multiple objects failure in overlapped write; fatal.
SSITHREAD_WRITEFILEFAIL	-28	Call to Write failed, but isn't just delayed; fatal error.
SSITHREAD_BADSETEV	-29	Write thread returned error on set event.
SSIRTHREAD_ORESULT	-30	Read thread bad overlapped result; fatal error.
SSIRTHREAD_SETMASK	-31	Read thread bad set mask return; fatal error.
SSIRTHREAD_BADREAD	-32	Read thread bad read; fatal error.
SSIRTHREAD_CREATEREVENT	-33	Read thread bad create read event; error code set, API call will return false.
SSIRTHREAD_CREATESEVENT	-34	Read thread bad create status event; error code set, API call will return false.
SSIRTHREAD_WAITCEVENT	-35	Read thread wait COM event bad return; fatal error.

The following error codes are placed in the wParam of WM_ERROR messages during SSI protocol handling of scanner messages.

Table 1-3 *WM_ERROR Messages in wParam*

Define Name	Value	Description
COMMAND_NOTHANDLED	-36	Command not processed successfully by decoder.
ERR_UNSUPPORTED_COMMAND	-37	Command not processed successfully by decoder.
SSI_DATAFORMAT_ERR	-38	Scanner data packet not of correct format from decoder.
ERR_UNEXPECTEDDATA	-39	State machine received data unexpected for the current state.

SSI WM_TIMEOUT Messages

One of these OPCODES is placed in the LPARAM of the SSI WM_TIMEOUT message.

Table 1-4 *OPCODES in lParam*

DECODE_DATA_TIMEOUT	0xF3
IMAGE_DATA_TIMEOUT	0xB1
VIDEO_DATA_TIMEOUT	0xB4

Beep Command Parameters

Table 1-5 lists the beep codes for the Sound Beeper function.

Table 1-5 *Beep Codes*

Define Name	Value (hexadecimal)
ONESHORTHI	0x00
TWOSHORTHI	0x01
THREESHORTHI	0x02
FOURSHORTHI	0x03
FIVESHORTHI	0x04
ONESHORTLO	0x05
TWOSHORTLO	0x06
THREESHORTLO	0x07
FOURSHORTLO	0x08
FIVESHORTLO	0x09
ONELONGHI	0x0A
TWOLONGHI	0x0B
THREELONGHI	0x0C
FOURLONGHI	0x0D
FIVELONGHI	0x0E
ONELONGLO	0x0F
TWOLONGLO	0x10
THREELONGLO	0x11

Table 1-5 *Beep Codes (Continued)*

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

Data Returned by the DLL

The host application provides the destination data buffer for use by the DLL. When the scanner sends data to the DLL, the destination buffer is filled with the scanner's data. Data is formatted according to the SSI specification; refer to the *Simple Serial Interface Programmer’s Guide*. Once the destination buffer is filled by the DLL (if a buffer was set), the application is sent a WM_xxx message with the number of bytes of data that were stored indicated in the LPARAM. If the buffer wasn't large enough to hold all the data, WPARAM's last 2 bits are set to zero. If no buffer was given to the DLL for the data to be stored in, the last 2 bits of WPARAM are 01. If the data was stored correctly, the last 2 bits of WPARAM are 11. The following #defines are provided for this purpose.

Table 1-6 *#defines*

Define Name	Value
BUFFERSIZE_MASK	0x0003
BUFFERSIZE_GOOD	0x0003
BUFFERSIZE_ERROR	0x0000
NOBUFFER_ERROR	0x0001

After the message is sent, the DLL marks the buffer as NULL indicating no user buffer is available for storage. The host application should reset the buffer after a WM_xxx message occurs. A second call to set the data buffer causes the new buffer to be used for incoming data. For example, after a WM_DECODE message is sent to the application, the application should handle the message and process the data in the destination buffer, then call *SetDecodeBuffer* again.

Windows Messages Sent to Calling Process

Message:	**WM_DECODE**
Value:	WM_APP+1
Description:	Decode data is available from the scanner and is stored in the buffer provided by a previous call to *SetDecodeBuffer*.
Parameters:	wParam: buffer status of the data stored lParam: length of the data in bytes (cast to int)
Data Format:	The bar code type is stored in the first byte and the decoded message is contained in the *Decode Data* field. For bar code types, see the *Simple Serial Interface (SSI) Programmer's Guide*.

Bar Code Type	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 and the decode data is formatted into packets. In this case, the first byte of the *Decode Data* field contains the actual Bar Code Type, the 2nd byte contains the number of packets, and the remaining data is the packeted form of decode data.

For example, a packeted Decode Data message for Micro PDF417 would look like the format below, where the Decode Data field is broken out as follows:

Bar Code Type	Decode Data Field							
	Actual Bar Code Type	# of Packets	Spare Byte	Byte Length of Packet #1	Data	Spare Byte	Byte Length of Packet #2	Data
0x99	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.

Message:	**WM_IMAGE**
Value:	WM_APP+2
Description:	Image data is available from the scanner and is stored in the buffer provided by a previous call to *SetImageBuffer*.
Parameters:	wParam: buffer status of the data stored lParam: length of the data in bytes (cast to long)
Data Format:	An image preamble followed by the image data.

Preamble	Image Frame

Images sent from the decoder to the host are described by the image preamble contained in the first 10 bytes, followed by the image. The details of the image preamble follow.

The image preamble consists of the following fields:

Table 1-7 *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

Message:	**WM_VIDEOIMAGE**
Value:	WM_APP+3
Description:	A video frame is available from the scanner and is stored in the buffer provided by a previous call to *SetVideoBuffer*.
Parameters:	wParam: buffer status of the data stored lParam: length of the data in bytes (cast to int)
Data Format:	The first 10 bytes of a video frame contains the video preamble, described below. The remaining data is the JPEG data comprising the video frame.

Preamble	Video Frame

The video preamble consists of the following fields:

Table 1-8 *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
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

Message:	**WM_ERROR**
Value:	WM_APP+4
Description:	An error occurred. This message may be sent in response to an API Command or Request for data.
Parameters:	wParam: error code (cast to int) (see WM_ERROR codes list)

Message:	**WM_TIMEOUT**
Value:	WM_APP+5
Description:	Scanner did not respond to a request from the library within the timeout period during processing of unsolicited data (decode data, image data or video data) from the scanner.
Parameters:	wParam: set to zero (reserved for future use) lParam: code (int) indicating what message was being processed when the timeout occurred (see WM_TIMEOUT codes list)

Message:	**WM_CMDCOMPLETEMSG**
Value:	WM_APP+6
Description:	Verifies that the scanner has handled the command that was issued. API functions that request data from the scanner will not receive this message.
Parameters:	None

Message:	**WM_XFERSTATUS**
Value:	WM_APP+7
Description:	Image data is transferring from the scanner.
Parameters:	wParam: total number of bytes received so far (cast to uint) lParam: total number of bytes expected (cast to uint)

Message:	**WM_SWVERSION**
Value:	WM_APP+8
Description:	Software version information is available from the scanner and is stored in the buffer provided by a previous call to *SetVersionBuffer*. This message is received in response to an API request for version data.
Parameters:	wParam: buffer status code lParam: length of the data in bytes (cast to int)
Data Format:	Revision string:

S/W_REVISION <space> BOARD_TYPE <space> ENGINE_CODE <space> PGM_CHKSUM

where:

- **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 or four byte checksum of the program code (scanner dependent)

Message:	**WM_PARAMS**
Value:	WM_APP+9
Description:	Parameter information is available from the scanner and is stored in the buffer provided by a previous call to *SetParameterBuffer*. This message is received in response to an API request for parameter data.
Parameters:	wParam: buffer status lParam: length of the param data (cast to int)
Data Format:	Parameter numbers may be a single byte or two bytes. Parameter numbers consisting of two bytes begin with an extended parameter code of F0h (+256), F1h (+512), F2h (+768). These access parameters whose numbers are 256 and higher. For example, to access the first parameter in the 256-511 range, use F0h and 00h. Parameter values may also be a single byte or two bytes. If the value is two bytes instead of one, the parameter number is preceeded by a byte containing F4h.

Table 1-9 *Param Data Format*

Parameter Number	Data Format
0 through EFh	<param_num> <value>
>EFh	<extended parameter code> <param_num offset> <value>

Message:	**WM_CAPABILITIES**
Value:	WM_APP+10
Description:	Capabilities data is available from the scanner and is stored in the buffer provided by a previous call to *SetCapabilitiesBuffer*. This message is received in response to an API request for capabilities data.
Parameters:	wParam: buffer status lParam: length of the data in bytes (cast to int)
Data Format:	Scanner capabilities data is packed into four data fields: Baud Rates Supported, Misc. Serial Parameters, Multipacket Options, and Command List.

Table 1-10 *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	

Table 1-10 *Data Fields (Continued)*

Field	Size	Description		Supported
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	
Multipacket 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 opcodes of the SSI commands it supports. For example, imagers support video commands, while laser-based decoders do not. Commands associated with video mode do not appear in the list for laser-based decoders, but do for imagers (see *Simple Serial Interface (SSI) Programmer's Guide)*.		

Message:	**WM_EVENT**
Value:	WM_APP+11
Description:	Event data is available from the scanner. No destination buffer is required for this unsolicited data from the scanner; the data is sent in the WPARAM along with the message.
Parameters:	wParam: event data lParam: length of the data in bytes (always 1 byte)

Message:	**WM_FW_UPDATE_PROGRESS**
Value:	WM_APP+13
Description:	Indicates firmware update progress. It keeps the wParam value at 0 (Case 1) to indicate the initial stages of firmware update. See *Table 1-11*. lParam returns the current status. During firmware update, the same message indicates the number of records updated (Case 2). lParam indicates the number of records updated so far, and wParam contains the total number of records that must be sent to the scanner.
Parameters:	Case1: Used to report initial steps in firmware update process. wParam: Zero (cast to int) lParam: Indicates the current status of the firmware update process (cast to int)

Table 1-11 *lCurrent Status of Firmware Update Process*

Value	Description
10	Firmware file opened
9	Checsum validation successful
8	COM port opened successfully
7	Successful switch to firmware update mode
6	Baud rate negotiated successfully with scanner
5	Starting firmware download
4	Firmware update failed to complete
3	Firmware update completed successfully

Case 2: A message is sent after every record update during firmware update.
wParam: Non-zero. Total number of records in firmware file (cast to DWORD)
lParam : Number of records updated so far (cast to DWORD)

Message:	**WM_FW_UPDATED_BAUD**
Value:	WM_APP+14
Description:	Indicates the baud rate used during the firmware update. It attempts to negotiate the maximum baud rate with the scanner, and returns the baud rate used in lParam.
Parameters:	lParam: Negotiated baud rate used during firmware update (cast to DWORD)

Message:	**WM_FW_UPDATE_FAILED**
Value:	WM_APP+15
Description:	Firmware update process failed, most commonly due to a fatal error. The error code is returned in lParam associated with the message. See *Table 1-12*.
Parameters:	lParam: Error code indicating the cause of failure (cast to DWORD)

The following generate WM_FW_UPDATE_FAILED messages to the calling application. The error code is returned in the LPARAM associated with the WM_FW_UPDATE_FAILED message.

Table 1-12 *WM_FW_UPDATE_FAILED Messages*

Define Name	Value	Description
SSIFW_UPDATE_FAILED	-40	Generic error accessing firmware update function.
SSIFW_FILE_OPEN_FAIL	-41	Unable to open firmware file.
SSIFW_INVALID_CHECKSUM	-42	Checksum calculation for firmware file failed.
SSIFW_COM_PORT_FAIL	-43	Unable to open COM port.
SSIFW_FW_UPDATE_MODE_FAIL	-44	Failed to enter firmware update mode.
SSIFW_PROG_COM_PORT_FAIL	-45	Unable to set requested baud rate on the COM port.
SSIFW_FILE_READ_FAIL	-46	Unable to read records from firmware file.
SSIFW_TX_FAIL	-47	Failure in serial transmission.
SSIFW_RX_FAIL	-48	Failure in serial receiving.
SSIFW_INVALID_REC	-49	Checksum calculation for the record failed.
SSIFW_UPDATE_FAILED_IN_SCN	-50	Failure occurred while updating firmware.
SSIFW_UPDATE_ABORTED	-51	User aborted firmware update process.

Chapter 2 ActiveX Control

Introduction

SSIConnect.ocx is a component that may be added to a Visual Basic project. It allows you to send commands to a Motorola scanner and request data from the scanner using the serial port. The control also handles bar code and image data sent from the scanner. This chapter provides examples for using the ocx; refer to the on-line help for specific information on using the ocx and its properties, methods and events.

✓ ***NOTE*** SSI SDK 3.0, SSIConnect ActiveX control does not support **Update Firmware** and **Abort Firmware Update** features.

✓ ***NOTE*** When using ActiveX control in C# application, after adding SSIConnect Control onto Form control, remove the existing references for stdole, SSICONNECTLib, and AxSSICONNECTLib, and add references to stdole.dll, SSICONNECTLib.dll, and AxSSICONNECTLib.dll located in (Drive):\Program Files\Motorola\SSI SDK\SDK.

Adding the SSIConnect Component to Your Project

Once the control is registered, it may be added to your project. In Visual Basic 6.0, click on the project menu, then choose *Components*. A list of all the registered controls on your system appears, including the SSIConnect Active X control. Check this item and click OK. An icon for the control appears, which you may drag and drop onto your form.

Setting Properties

After you have dropped the control onto your form, you can set the properties for the control. Visual Basic assigns the control the default name *SSIConnect1*. This name allows you access to the properties and functionality of the control during run-time. You may change the name when you design the form. Do not change any other properties except for the following, which you must set according to your system:

- *ComPortNumber* - Set this to the COM port on your PC the scanner is attached to. Set the value to 1 for COM1, 2 for COM2 etc.
- *BaudRate* - Set to the baud rate the scanner uses. Most scanners default to 9600. See your scanner’s documentation for baud rates supported. Values are input as 9600 for 9.6kb, 115200 for 115.2kb, etc.

- *ParameterPersistance* - Set to TRUE if you want scanner parameters changed by your program to change permanently. Set to FALSE to cause parameter changes to be temporary.
- *ImageFilename* - Enter the name of the file to save image data to. Do not enter a filename extension; this is added by the control according to the image type the scanner sends. Note that only imaging scanners support image data.
- *SendMacroPDFDataOnAbort* - Set to TRUE for the scanner to send buffered MacroPDF data when it receives an abort command from your program. Set to FALSE to discard buffered data. Note that not all scanners support MacroPDF decoding.
- *LEDCode* - Set the LED(s) that will be turned off or on when you send the LED command. Values are scanner-specific.
- *BeepCode* - Set the beep sequence the scanner emits when it receives a beep command. Valid values are 0 through 25. See your scanner documentation for beep codes. For the VS 4000, beep codes are as follows:

Table 2-1 *VS 4000 Beep Codes*

Beep Code	Value
ONESHORTHI	0
TWOSHORTHI	1
THREESHORTHI	2
FOURSHORTHI	3
FIVESHORTHI	4
ONESHORTLO	5
TWOSHORTLO	6
THREESHORTLO	7
FOURSHORTLO	8
FIVESHORTLO	9
ONELONGHI	10
TWOLONGHI	11
THREELONGHI	12
FOURLONGHI	13
FIVELONGHI	14
ONELONGLO	15
TWOLONGLO	16
THREELONGLO	17
FOURLONGLO	18
FIVELONGLO	19
FASTHILOHILO	20

Table 2-1 *VS 4000 Beep Codes (Continued)*

Beep Code	Value
SLOWHILOHILO	21
HILO	22
LOHI	23
HILOHI	24
LOHILO	25

You may also also retrieve and change these properties while your program is running using the name of your control. For example, if you have not changed the default and your control is named SSIConnect1, add the following code to your VB program to retrieve the value for the COM port number:

```
Dim b As Long
b = SSIConnect1.ComPortNumber
```

Note that the COM port number is a Long value. To set the port number to use COM port 5, add:

```
SSIConnect1.ComPortNumber = 5
```

Baud rate is also a Long value, so it can be changed in the same manner.

To change the other properties, Dim a variable of the type expected for the property:

- Image filename requires a String variable
- LED code and Beep code require an Integer variable
- The remaining properties require a Boolean variable.

For example, to get and set the filename to a different value:

```
Dim name As String
' this gets the current name
name = SSIConnect1.ImageFilename
' this sets the name to hello
SSIConnect1.ImageFilename = "hello"
```

Communicating with the Scanner Using Your Control

After setting your control's properties, you may call methods to command the scanner to perform different functions. The control reports the results of your commands, and also reports when it receives decode and image data from the scanner.

You must add code to your VB program to send the commands to the scanner and to handle notification events. In your handler you must call other methods to retrieve data sent by the scanner.

Your communication with the scanner is like a transaction. The host sends a single command to the scanner; when the scanner performs the command, it is ready to accept a new command.

Remember to scan the SSI Host parameter bar code for the VS 4004 imager before testing your program. You must also set two important parameters, either via bar code parameter or using the *ChangeParameter* method: set *Trigger Mode* to Host, and *Decode Data Packet Format* to Send Packeted Decode Data. See *ChangeParameter(Parameter As Long, Value As Long) As Long on page 2-7* for details.

Command Methods

The first method you must call is *ConnectComPort*, and the last is *DisconnectComPort*. *ConnectComPort* opens your COM port for communication with the scanner. *DisconnectComPort* closes the COM port and releases the host PC's memory used during the scanning session.

To see how this works, in your VB project where you have dropped the SSIConnect control:

1. Drag and drop a button control onto your form.
2. Change the Caption property of the button to Connect, then double-click the button.
3. Visual Basic will have added an empty sub-routine. Add the following lines of code to this sub-routine:

```
Dim Status As Long
Status = SSIConnect1.ConnectComPort()
```

The status returned is zero if the function was successful. We recommend checking the return value and displaying a message indicating if the connection was successful.

If the return status is not zero, either your COM port number is incorrect or it is already in use. Make sure you are using the correct COM port number, and that no other program (e.g., ActiveSync) is using this port.

Add a button for disconnecting in the same manner, and add the following lines to the sub-routine:

```
Dim Status As Long
Status = SSIConnect1.DisconnectComPort()
```

The following sections describe the methods your program may call which command the scanner to perform an action. In Visual Basic, enter the name of the control, followed by a decimal point. A list of all available properties and methods appears. If you select a method from the list, when you enter an open parenthesis a list of the parameters the method requires from your program appears, along with their types.

The following methods send a command to the scanner. They all return the status, which is zero if the command was successful.

SendCommand(Command As Long) As Long

Description

The control allows you to send the commands in *Table 2-2* as a parameter to the method.

Example

```
Dim Status As Long
Status = SSIConnect1.SendCommand(ssiEnableScanner)
```

Table 2-2 *Commands*

Command	Value	Function
ssiTurnAimOn	0	Turns aiming light on*
ssiTurnAimOff	1	Turns aiming light off*
ssiTurnLedOn	2	Turns the LED specified by the LEDCode on*
ssiTurnLedOff	3	Turns the LED specified by the LEDCode off*
ssiEnableScanner	4	Enables scanner*
ssiDisableScanner	5	Disables scanner; scanner will not respond to a trigger pull*
ssiSendBeep	6	Sounds the beep specified by the BeepCode property*
ssiSWTriggerPull	7	Software trigger pull *, ***
ssiSWTriggerRelease	8	Software trigger release*
ssiAbortImage	9	If sent when the scanner is sending image data after taking a picture, the data transfer is aborted
ssiTakePicture	10	Sets the imager to snapshot mode, rather than decode mode. When the trigger is pulled, the imager takes a picture and sends the data to the PC *, ***
ssiStartVideoStream	11	Reserved - do not use
ssiRequestCapabilities	12	Commands the scanner to send its capabilities data to the PC**
ssiRequestAllParameters	13	Commands the scanner to send to the PC all supported parameters and their current values**

***These commands result in a status event that may be handled in your program. See *Events on page 2-10*.**
****These commands result in the scanner sending data to your program. When the data is available, you receive an event that can be handled by your program. If a problem occurred, your program can receive a status event. See *Events on page 2-10* for the types of events and how they may be handled in your program.**
*****After sending the commands ssiTakePicture and ssiSWTriggerPull consecutively, wait for related events to fire. If you must terminate before the full image is captured, send ssiAbortImage before terminating the task.**

Table 2-2 *Commands (Continued)*

Command	Value	Function
ssiRequestVersionData	14	Commands the scanner to send its version data to the PC**
ssiTerminateMacroPDF	15	When decoding MacroPDF, this aborts the current sequence. Any buffered data is either discarded or sent to the PC depending on the setting of the *SendMacroPDFDataOnAbort* property*
ssiEnterLowPowerMode	16	Commands the scanner to enter low power mode*

***These commands result in a status event that may be handled in your program. See *Events on page 2-10*.**
****These commands result in the scanner sending data to your program. When the data is available, you receive an event that can be handled by your program. If a problem occurred, your program can receive a status event. See *Events on page 2-10* for the types of events and how they may be handled in your program.**
*****After sending the commands ssiTakePicture and ssiSWTriggerPull consecutively, wait for related events to fire. If you must terminate before the full image is captured, send ssiAbortImage before terminating the task.**

RequestParameter(Parameter As Long) As Long

Description

This method commands the scanner to send the current setting for the given parameter number. In the example, the scanner is asked for the current setting for the trigger mode. The method returns zero if the command is successful. When the scanner sends the data requested to the host PC, your program is notified with an event. For information on handling the event, see *Events on page 2-10*.

Example

```
Dim Status As Long
Dim TriggerModeParam As Long
TriggerModeParam = 138
Status = SSIConnect1.RequestParameter(TriggerModeParam)
```

ChangeParameter(Parameter As Long, Value As Long) As Long

Description

This method commands the scanner to change the current setting for the given parameter number. In the example, the scanner is asked to change the current setting for the trigger mode to Host Mode. The method returns zero if the command is successful. When the command is processed, your program receives an event, which your program can handle. See *Events on page 2-10*.

✓ ***NOTE*** Some scanners require setting the trigger mode parameter to Host Mode during SSI communication. You must also set parameter number 238, Decode Data Packet Format, to 1.

Example

```
Dim Status As Long
Dim TriggerModeParam As Long
Dim HostMode As Long
TriggerModeParam = 138
HostMode = 8
Status = SSIConnect1.ChangeParameter(TriggerModeParam, HostMode)
```

Parameter Numbers

Parameter numbers are provided in hexadecimal format, for example Stop Bit Select is 0x9D; Decode Event is 0xF0 0x00. Note that Stop Bit Select has a single hex number, while Decode Event has two. Parameter numbers that have two hex numbers are called extended parameter numbers. In extended parameter numbers, the first number is always 0xF0, 0xF1 or 0xF2.

The two hex numbers represent a four-digit hex value. For instance, the parameter number for Decode Event 0xF0 0x00 is the hex value 0xF000, which in decimal is 61440.

Values may be either one or two hex numbers. The allowable values depend on the parameter. For example, the beeper tone parameter may only have values of low frequency, medium frequency or high frequency. When a value has 2 hex numbers, the parameter number requires an additional hex number: 0xF4 is added in front of the parameter number.

For example Beeper Tone has a non-extended parameter number with a value of only one hex number: 0x91, or 145 in decimal. Its values may be set to Low Frequency (value 0x02 or 2 in decimal), Medium Frequency (value 0x01 or 1 in decimal), or High Frequency (value 0x00 or 0 in decimal).

To use the control's ChangeParameter method to change the beeper tone to low frequency, use the following command:

```
Status = SSIConnect1.ChangeParameter(145, 2)
```

Decode Event has an extended parameter number whose value is only one hex number: 0xF0 0x00 (0xF000 in hex, 61440 in decimal). Its values may be set to 0 for disable, 1 for enable.

To use the control's *ChangeParameter* method to change the Decode Event parameter to its enable value, use the following command:

```
Status = SSIConnect1.ChangeParameter(61440, 1)
```

For an example of a parameter that requires a word value, Gain Setting is 0xF0 0x37 (or F0h, 37h), representing an extended parameter number of 0xF037. Its allowable values are 4 hex digits: 0080h, 00C0h, 0100h, 0140h, etc. Since these values are not 2 hex digits, the parameter number must be preceded by 0xF4, so the parameter number for Gain Setting becomes 0xF4F037, or 16052279 in decimal.

To use the control's *ChangeParameter* method to change the Gain Setting parameter to a value of 0080h (128 in decimal) use the following command:

```
Status = SSIConnect1.ChangeParameter( 16052279, 128)
```

Table 2-3 contains a sample of some parameters used in the VS 4000. These parameters may be enabled or disabled in the scanner. The associated values for these parameters are zero (disable) and one (enable).

Table 2-3 *VS 4000 Parameters*

Parameter Name	Parameter Number	Allowable Values
PARAMETER_SCANNING_PARAM	236 decimal or EC hex	0 or 1
BEEP_AFTER_GOOD_DECODE_PARAM	56 decimal or 38 hex	0 or 1
DECODING_AUTOEXPOSURE_PARAM	61481 decimal or F029 hex	0 or 1
DECODING_ILUMINATION_PARAM	61482 decimal or F02A hex	0 or 1
IMAGE_CAP_AUTOEXPOSURE_PARAM	61545 decimal or F069 hex	0 or 1

Table 2-3 *VS 4000 Parameters (Continued)*

Parameter Name	Parameter Number	Allowable Values
IMAGE_CAP_ILLUMINATION_PARAM	61544 decimal or F068 hex	0 or 1
UPCA_CODETYPE_PARAM	1 (decimal or hex)	0 or 1
UPCE_CODETYPE_PARAM	2 (decimal or hex)	0 or 1
UPCE1_CODETYPE_PARAM	12 decimal or 0C hex	0 or 1
EAN8_CODETYPE_PARAM	4 (decimal or hex)	0 or 1
EAN13_CODETYPE_PARAM	3 (decimal or hex)	0 or 1
BOOKLANDEAN_CODETYPE_PARAM	83 decimal or 53 hex	0 or 1
CODE39_CODETYPE_PARAM	0 (decimal or hex)	0 or 1
CODE39FULLASCII_CODETYPE_PARAM	17 decimal or 11 hex	0 or 1
TRIOPTICCODE39_CODETYPE_PARAM	13 decimal or 0D hex	0 or 1
CODE93_CODETYPE_PARAM	9 (decimal or hex)	0 or 1
COD128_CODETYPE_PARAM	8 (decimal or hex)	0 or 1
UCC128_CODETYPE_PARAM	14 decimal or 0E hex	0 or 1
ISBT128_CODETYPE_PARAM	84 decimal or 54 hex	0 or 1
CODABAR_CODETYPE_PARAM	7 (decimal or hex)	0 or 1
I2OF5_CODETYPE_PARAM	6 (decimal or hex)	0 or 1
D2OF5_CODETYPE_PARAM	5 (decimal or hex)	0 or 1
MSIPLESSEY_CODETYPE_PARAM	11 decimal or 0B hex	0 or 1
USPOSTNET_CODETYPE_PARAM	89 decimal or 59 hex	0 or 1
USPLANET_CODETYPE_PARAM	90 decimal or 5A hex	0 or 1
UKPOSTAL_CODETYPE_PARAM	91 decimal or 5B hex	0 or 1
JAPANPOSTAL_CODETYPE_PARAM	61474 decimal or F022 hex	0 or 1
AUSTRALIANPOST_CODETYPE_PARAM	61475 decimal or F023 hex	0 or 1
PDF417_CODETYPE_PARAM	15 decimal or 0F hex	0 or 1
MICROPDF417_CODETYPE_PARAM	227 decimal or E3 hex	0 or 1
DATAMATRIX_CODETYPE_PARAM	61476 decimal or F024 hex	0 or 1
MAXICODE_CODETYPE_PARAM	61478 decimal or F026 hex	0 or 1
QRCODE_CODETYPE_PARAM	61477 decimal or F025 hex	0 or 1

Events

Your program can receive the following events from the control:

Table 2-4 *Events*

Event	Cause
AllParametersAvailable	Command requesting all parameters
DecodeDataAvailable	Bar code was scanned
EventDataAvailable	Monitored event occurred
ImageDataAvailable	Trigger pull during imager mode
ImageTransferStatusAvailable	Image data is being sent from the scanner
ParameterAvailable	Command requesting a single parameter
ScannerCapabilities	Command requesting the scanner's capabilities
ScannerStatusAvailable	Scanner's command ack or timeout/error
VersionDataAvailable	Command requesting the scanner's version
VideoFrameAvailable	Reserved

You may handle one or more of these events in your program. For instance, if you never make a call to *SendCommand(ssiRequestVersionData)*, you will not handle the *VersionDataAvailable* event since you will never receive version data.

Event Handling in Visual Basic

To handle an event in Visual Basic:

1. Select the name of your control from the drop-down menu on the left in the Code window for your form.
2. Select an event from the drop-down menu on the right.
3. Visual Basic creates a sub-routine for this event. This sub-routine is the event handler. Some event handlers get the event information as a parameter value, while others require calling a method to retrieve this data during the event handling.

AllParametersAvailable

Description

The event handler for *AllParametersAvailable* is paired with a previous request using the *SendCommand(ssiRequestAllParameters)* method. To retrieve the data, you must wait until the data is available; then this event handler is called by the control. To retrieve the data during the event handler, use the *GetNextParameter* method. The method may only be called during this event handler, and may be called repeatedly until all parameters and their values are retrieved. Its parameter *NumValues* indicates how many parameter values were returned. If *NumValues* is less than or equal to zero, an error occurred and the *GetNextParameter* method should not be called.

Example

```
Dim ParamNumber As Long
Dim ParamVal As Long
Dim Status As Integer

Do While NumValues > 0  'NumValues came as an input param to this event handler

'GetNextParameter returns the next Parameter number and its value
'… and a status code which is set to zero when there is no parameter to return, or the
' …index of the parameter returned within NumValues

Status = SSIConnect1.GetNextParameter(ParamNumber, ParamVal)

'  check the status and do something like display the param number and value
' … here if the status is not zero.  If it is zero, break out of the while loop.

Loop
```

DecodeDataAvailable

Description

The event handler for *DecodeDataAvailable* has a single parameter for the length of the data received from the scanner. To retrieve the available data, call the method *GetDecodeData* during this event handler if the length parameter is greater than zero. If the length parameter is less than or equal to zero, an error occurred and *GetDecodeData* should not be called.

Example

```
Dim SSIcodetype As Integer
Dim DecodeData As Variant
Dim NumDataChars As Long

' Send the Variant DecodeData to be filled with the scanner's data along with the
' Visual Basic Constant vbString to indicate the type of data to be returned and the
' …variable SSIcodetype which will be set to the ssi codetype id of the data.
NumDataChars = SSIConnect1.GetDecodeData(DecodeData, vbString, SSIcodetype)
```

Notes

The value for *NumDataChars* is one less than the value sent with the event handler since the data returned from the scanner includes the SSICodetype. The maximum *NumDataChars* that will be returned is 5000. Note that with PDF417 bar codes, the data may contain unprintable characters, including the null character. Although the data may be accessed within the variant, it does not display as a string (e.g., fingerprint data held in the bar code).

The type specifier is needed because although Visual Basic programs may manipulate the data using a vbString, Visual C++ programmers use a data type specifier of VT_UI1, and *DecodeData* would be a pointer to *ColeSafeArray*, allowing the returned data to be manipulated as an array of unsigned character.

If the SSI CodeType returned has a value of 99 hex, the DecodeData is formatted as decode data packets as described in the *SSI Programmer’s Guide*.

EventDataAvailable

Description

The event handler for *EventDataAvailable* sends a parameter that is the event code.

You will receive scanner event type data if you set *Event Reporting* parameters to monitor a particular type of event, and that event occurs. Refer to *Standard Default Parameters* in the scanner’s *Product Reference Guide* for information on *Event Reporting* parameters.

ImageDataAvailable

Description

The event handler for *ImageDataAvailable* sends a parameter that contains the image data as a picture. To display the picture in your program, drag and drop a picture box control onto your form. In the event handler, set its picture to the image sent by the SSIConnect.

For example, for a picture box with the default name Picture1:

```
Picture1.Picture = Image
```

ImageTransferStatusAvailable

Description

The event handler for *ImageTransferStatusAvailable* sends two parameters:

- *TotalFileLength* of the image
- *CurrentFileLength* indicates how much data was transferred so far.

If you include this handler, you may build a status message that displays in a label control while the scanner is transferring data:

```
StrMessage = "Received " & Str(CurrentFileLength) & "bytes of " & Str(TotalFileLength)
```

ParameterAvailable

Description

The event handler for *ParameterAvailable* has no parameters. This event is received in response to an earlier method call to *RequestParameter*. To retrieve the data, call the method *GetParameter* during this event handler sub-routine.

Example

```
Dim TriggerModeParamNum = 138  ' set the parameter number to the parameter that you
Dim ParamValue As Long         ' …requested earlier in the call to RequestParameter.
Dim Success As Boolean

' Send in the Parameter number, get back the Parameter's value
Success = SSIConnect1.GetParameter(TriggerModeParamNum, ParamValue)
```

The method returns TRUE if the parameter data that was sent was for the parameter number requested - in this example, if the *ParamValue* sent by the scanner was for the trigger mode parameter, the return value is true. If you are trying to get the value for a different parameter number than the one you requested, the function returns FALSE. This function also returns FALSE if you call the *GetParameter* method more than once during the handler, or if you call *GetParameter* anywhere other than in the *ParameterAvailable* event handler sub-routine.

Following is an example of program flow:

Pressing the connect button calls the *SSIConnect1.ConnectComPort* method. If that returns successful status, the call to get the value of the trigger mode parameter is made:

Status = SSIConnect1.RequestParameter(TriggerModeParam).

This method's return value indicates if the command was sent successfully. After your event handler is called by the control, the data is available for access using the *GetParameter* method.

ScannerCapabilities

Description

The event handler for *ScannerCapabilities* is paired with a previous call to *SendCommand(ssiRequestCapabilities)*. During the resulting *ScannerCapabilities* event handler you may call the following methods to retrieve the scanner's capability data as long as the *NumCommands* parameter is greater than zero:

- GetParityCheckCapabilities
- GetParityCapabilities
- GetStopBitsSupported
- MultiPacketOptionCapabilities
- GetSSICommandsSupported
- GetBaudRateCapabilities

Example

```
Dim NumBaudRates As Long
Dim BaudRates As Variant
Dim NumSSICommands As Long
Dim SSICommands As Variant
Dim odd As Boolean
Dim even As Boolean
Dim noparity As Boolean
Dim onestop As Boolean
Dim twostops As Boolean
Dim myopt1 As Boolean
Dim myopt2 As Boolean
Dim myopt3 As Boolean
Dim checkparity As Boolean
Dim dontchekparity As Boolean
Dim mystatus As Boolean

' mystatatus will be true if the capabilities were retrieved successfully

' the following functions return the capabilities of the scanner
' For example, if the scanner supports parity checking, the checkparity param will be true
'…If the scanner supports odd parity, the parameter odd will be true, etc.

mystatus = SSIConnect1.GetParityCheckCapabilities(checkparity, dontchekparity)
mystatus = SSIConnect1.GetParityCapabilities(odd, even, noparity)
mystatus = SSIConnect1.GetStopBitsSupported(onestop, twostops)
mystatus = SSIConnect1.MultiPacketOptionCapabilities(myopt1, myopt2, myopt3)

' This method returns the capabilities of the scanner with regard to the ssi commands
' ..that it supports.  SSICommands contains the list of  SSI Command opcodes as Long values, and ' its
method returns the number of SSI commands that are supported.

NumSSICommands  = GetSSICommandsSupported(SSICommands)

' This method returns the number of baud rates supported by the scanner and the param
'  BaudRates contains the list of supported baud rates as Long values
NumBaudRates = GetBaudRateCapabilities(BaudRates)
```

ScannerStatusAvailable

Description

The event handler for *ScannerStatusAvailable* sends a parameter that is the status code. The status code can represent either command completion (zero) or an error/timeout code.

When you send a simple command to the scanner (see *Table 2-2 on page 2-5*; note items with a single asterisk) you receive either command completion status or the error/timeout code if the command was not handled. You do not receive command completion status for commands that request data; in this case, you either receive the data or an error/timeout code.

VersionDataAvailable

Description

The event handler for *VersionDataAvailable* has a single parameter for the length of the data received from the scanner. The event is paired with a previous call to *SendCommand(ssiRequestVersion)*. During this event handler you may call the method *GetVersionData* to retrieve the scanner's version data if the length parameter is greater than zero. If the length parameter is less than or equal to zero, an error occurred and *GetVersionData* should not be called.

Example

```
Dim VersionData As Variant
Dim NumDataChars As Long

' Send the Variant VersionData to be filled with the scanner's data along with the
' Visual Basic Constant vbString to indicate the type of data to be returned is a string
NumDataChars = SSIConnect1.GetVersionData(VersionData, vbString)
```

The type specifier is needed because although Visual Basic programs may manipulate the data using a vbString, Visual C++ programmers would use a data type specifier of VT_UI1, and *VersionData* would be a pointer to *ColeSafeArray*, allowing the returned data to be manipulated as an array of unsigned character.

The VersionData string returned is scanner dependent. If the data format follows that of the SSI specification, the string returned is formatted as 4 labeled fields separated by CRLF: Software Revision:, Board Type:, Engine Code:, and Program Checksum:. If the data format does not follow the specification, raw unformatted data is returned. In this case, non-printable characters, including null characters, may be present in the data. The data is still available but does not display as normal string data.

✓ ***NOTE*** Engine Code and Checksum are returned as hex digits.

Index

MOTOROLA

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

72E-59860-03 Revision A - September 2007