CONTEX SCANNER

SOFTWARE DEVELOPMENT KIT (SDK)

Document Number:
Author: DIN
Issue Revision: 14.0
Issue Date: 16-10-18

DOCUMENT REVISION RECORD			DRR
CHANGE	DATE	PAGES AFFECTED	BRIEF DESCRIPTION OF CHANGE
11.0	10-11-11	All	Major updates of the document
13.0	06-09-16	All	Introduction of both 32 and 64 bit support
14.0	15-10-18	All	Support for the HD Ultra X Scanner Series

TABLE OF CONTENTS

1 Introduction ____ *4*

2 Requirements ____ *4*

3 Functional Overview ____ *4*

4 Function call description ____ *6*

4.1 Avoiding data-type alignment errors ____ **6**

4.2 The HD Ultra X scanner series ____ **6**

4.3 Driver system functions ____ **6**

4.3.1 scanOpenLib ____ 6

4.3.2 scanCloseLib ____ 7

4.3.3 scanRescanScsiBus ____ 8

4.4 Scanner system functions ____ **8**

4.4.1 scanGetNextScanner ____ 8

4.4.2 scanOpenScanner ____ 8

4.4.3 scanCloseScanner ____ 9

4.4.4 scanGetLastAsc ____ 9

4.4.5 scanGetHardwareType ____ 9

4.4.6 scanGetHostIp ____ 9

4.4.7 scanGetHostName ____ 10

4.5 Scanner I/O functions ____ **10**

4.5.1 scanTestUnitReady ____ 10

4.5.2 scanReserveUnit ____ 10

4.5.3 scanReleaseUnit ____ 10

4.5.4 scanInquiry ____ 11

4.5.5 scanInquiryPage ____ 11

4.5.6 scanScan ____ 11

4.5.7 scanRead ____ 12

4.5.8 scanSend ____ 12

4.5.9 scanSetWindow ____ 13

4.5.10 scanObjectPosition ____ 13

4.5.11 scanWriteBuffer ____ 14

4.5.12 scanReadBuffer ____ 14

4.5.13 scanReceiveDiagnostic ____ 14

4.5.14 scanSendDiagnostic ____ 15

4.5.15 scanGetScannerNameToDisplay ____ 15

4.6 Error Messages ____ **16**

4.6.1 scanGetErrorMessage ____ 16

4.6.2 scanGetDiagnosticsErrorMessage ____ 16

5 Accessing the CTX_SCAN_2000 functions ____ *17*

5.1 The SimpleScan sample program ____ **17**

6 Configuring your project to use the SDK ____ *17*

1 Introduction

This document describes the Contex Software Development Kit (SDK) for Contex scanners.

The SDK is a library that provides a low-level interface to all commands in the CONTEX scanners. The SDK library supports only USB and network connected CONTEX scanners, but the communication is based on the SCSI command protocol by historical reasons.

This document does not describe the actual parameter setting for the individual SCSI commands. Please refer to FSS/ICD/005 for those details.

2 Requirements

This SDK supports:

- Windows 7
- Windows Server 2008
- Windows 8.1
- Windows 10

WIDEsystem automatically installs all drivers, INI and INF files and the required DLL files to support both 32 and 64 bit programs. Therefore these files are not included with the SDK.

A 32 bit program uses the library ctx_scan_2000.dll and a 64 bit program uses the library CtxScan64.dll, but this is transparent as the supplied source code **ctx_scan_2000_loader.cpp** loads the appropriate library. Both libraries are in in the following referred to as CTX_SCAN_2000.

3 Functional Overview

CTX_SCAN_2000 provides a low-level scanner interface to the scanner. Before any functions can be called the libraries must be initialized using **scanOpenLib**. This function will search the SCSI bus for scanner devices. Handles to all supported scanners on the busses can be inquired using the **scanGetNextScanner** function. The handles returned are of the type HSCANNER, and may be platform dependent.

Note: The scanners handles are unique on a PC; i.e. if you have more than one scanner connected and two applications uses scanner handle 100 then WIDEsystem will ensure both applications is communicating with the same scanner.

Once the initialization is completed, commands can be sent to the scanner. Before commands can be send to a specific scanner it must however be opened using **scanOpenScanner**. When the program is finished with a scanner it must be closed with **scanCloseScanner**. It is recommended that the scanner should be closed once an operation on the scanner is completed. The library is closed by calling **scanCloseLib**. It is allowed to open the library or a scanner more than once, as long as the equivalent units are closed the same number of times.

To communicate with the scanner CTX_SCAN_2000 provides a function call for each command supported by the scanners. All these functions return as follows:

- All functions that succeeds returns 0.
- In case of hardware, driver, protocol error or system error, the functions return a negative value indicating the nature of the error. The possible values are defined in CTX_SCAN_2000.H with the prefix SCSI_ERROR or STI_ERROR.
- If the scanner returns with a *check condition*, the #define SCSI_STATUS_CHECK_CONDITION is returned by the function. The actual reason for

this can be investigated using the **scanGetLastAsc** command. This function returns the ASC/ASCQ[⊗] information from the last SCSI command.

When the application that uses CTX_SCAN_2000 exits, it is very important to call **scanCloseLib** in order to properly clean up any resources allocated by the library.

To use the low level SDK in your project see section 6 Configuring your project to use the SDK.

[⊗] The **A**dditional **S**ense **C**ode and **A**dditional **S**ense **C**ode **Q**ualifier are listed in the description of the REQUEST SENSE command in the FSS/ICD/005 document.

4 Function call description

In the following, all function calls to CTX_SCAN_2000 are described. The actual use for the individual scanner commands and the scanner data are not described in this document so please refer to FSS/ICD/005 for this information.

4.1 Avoiding data-type alignment errors

When passing data buffers to the commands, make sure that your data structure are aligned along 8 byte boundaries. Failure to do so may cause strange results when e.g. reading image data. If you are using Microsoft Visual Studio version 2010 the alignment preference is set in the project settings, under C/C++ -> Code Generation –> struct member alignment. This will cause your structures and buffers to be allocated with the proper memory alignment.

4.2 The HD Ultra X scanner series

With the HD Ultra X scanner series is introduced an USB3 scanner connection and a new connection type – see **4.4.5 scanGetHardwareType**.

An HD Ultra X scanner may change to *Sleep mode* either when a certain idle time has expired or programmatically from an application – see the SimpleScan sample program. The scanner will only enter *Sleep mode* when "*Enter Sleep mode automatically"* is enabled on the *Timer* tab in WIDEsystem and no other scanner application has an open connection to the same scanner.

An application can also wake up the scanner if connected with an USB cable; this is done automatically whenever the *scanOpenScanner* functions is called.

Note: A program may get a HD Ultra scanner into *Sleep mode* even when scanner is Ethernet connected, but the scanner cannot be wake up, as the scanner does not respond to Ethernet messages during *Sleep mode.*

Function calls to CTX_SCAN_2000 are handled like this when scanner is in *Sleep mode*:

- Call to *scanInquiry* or *scanInquiryPage* returns cached inquiry data and return code 0.
- Call to scanRead with code 0x80 and qualifier 0x00 (Scanner status) returns the Scanner Mode 0x0A (Standby mode).
- Call to *scanReceiveDiagnostic* returns no diagnostic data and return code 0.
- All other calls involving the scanner returns SCSI_STATUS_CHECK_CONDITION and a subsequent call to *scanGetLastAsc* will return code 0xA015 (ERROR_IN_POWER_DOWN_MODE)

4.3 Driver system functions

These functions are used to do driver specific things.

4.3.1 scanOpenLib

Prototype: int scanOpenLib(void);

Description: This function **must** be called before any other functions in the library can be called, as it makes a delay load of the ctx_scan_2000.dll file.
The function need only be called once per process. When this function is called all SCSI ID's is searched for SCSI devices.

Parameters: void

Returns: 0 if successful.

< 0 on error.

SCSI_ERROR_NO_DRIVER (-111) if the drivers are not found.

SCSI_ERROR_NO_SCANNER_FOUND (-117) if no Contex scanner is found.

SCSI_ERROR_UNABLE_TO_OPEN_DRIVER (-118) if ctx_sti.dll is not found on the system.

SCSI_ERROR_DLL_LOAD_FAILURE (-99) if load of DLL failed (installation is faulty)

4.3.2 scanCloseLib

Prototype: int scanCloseLib(void);

Description: Must be called when the library is not to be used anymore, in order to release allocated system resources.

Parameters: void

Returns: 0 if successful

< 0 on error.

SCSI_ERROR_SCANNERS_STILL_OPEN (-119) if the library could not be closed because a scanner is still open. Make sure to close the scanner as many times as it is opened.

4.3.3 scanRescanScsiBus

Prototype: int scanRescanScsiBus(void);

Description: Can be used to make the operating system rescan the SCSI bus. Is typically used if scanOpenLib fails. scanOpenLib must be called again after this call to initialize the driver. The scanRescanScsiBus function may take an extended period of time to complete.

Parameters: void

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

4.4 Scanner system functions

These functions are used identify the scanners in the system, to open and close scanners and to get and set information for scanners.

4.4.1 scanGetNextScanner

Prototype: int scanGetNextScanner(HSCANNER *hScanner, BOOL *isOpen, BOOL restart);

Description: This function can be used to iterate through the scanners found on the SCSI bus.

Parameters:

HSCANNER* hScanner	Used to return a handle to a scanner. The value of hScanner is undefined when no scanner is found.
BOOL* isOpen	TRUE when scanner is already open. Even if the scanner is already open it is recommended that every thread of execution opens and closes the scanner before and after use respectively.
BOOL restart	If TRUE the function will return the first ID number. If it is FALSE the next ID number after the previous is returned.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

4.4.2 scanOpenScanner

Prototype: int scanOpenScanner(HSCANNER hScanner);

Description: This function is used to open a scanner for I/O.

Parameters: HSCANNER hScanner Handle to the scanner to open.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION : Call the **scanGetLastAsc** function to get the value of ASC. Gigabit Ethernet scanners may return the defined error: ERROR_MAX_SCANNER_CONNECTIONS.

4.4.3 scanCloseScanner

Prototype: int scanCloseScanner(HSCANNER hScanner);

Description: This function is used to close a scanner for I/O.
Note: remember to close a scanner once for each time it is opened. If the scanOpenScanner and scanCloseScanner calls do not match, then scanCloseLib will fail.

Parameters: HSCANNER hScanner — Handle to the scanner to close.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

4.4.4 scanGetLastAsc

Prototype: int scanGetLastAsc(HSCANNER hScanner);

Description: If a function that sends a SCSI command returns SCSI_STATUS_CHECK_CONDITION the **scanGetLastAsc** function should be called to determine the nature of the problem.

Parameters: HSCANNER hScanner — Handle of the scanner to get the last ASC/ASCQ from.

Returns: ASC*256 + ASCQ

4.4.5 scanGetHardwareType

Prototype: ctxConnectionType scanGetHardwareType(HSCANNER hScanner);

Description: Used to detect which kind connection the scanner is using.

Parameters: HSCANNER hScanner — Handle of the scanner.

Returns: returns one of the following values of type ctxConnectionType:

```
typedef enum{
  CTX_CONNECTION_UNKNOWN=0,   // No interface selected
  CTX_CONNECTION_USB,         // connected via. USB
  CTX_CONNECTION_SCSI,        // connected via. SCSI
  CTX_CONNECTION_1394,        // connected via. 1394/FireWire
  CTX_CONNECTION_ETHERNET,    // connected via. TCP/IP
  CTX_CONNECTION_USBCY,       // connected via. Cypress USB3
} ctxConnectionType;
```

4.4.6 scanGetHostIp

Prototype: int scanGetHostIp(HSCANNER hScanner, char *buffer, BYTE lenBuffer);

Description: Used to get the IP number of a scanner that is connected via IP.

Parameters: HSCANNER hScanner — Handle of the scanner.

char* bBuffer — Pointer to the buffer to get the IP number in.

BYTE lenBuffer — Number of bytes in the buffer.

Returns: returns one of the following values:

0: Success.
1: This scanner not connected via TCP/IP.
2: Buffer is not big enough.

4.4.7 scanGetHostName

Prototype: int scanGetHostName(HSCANNER hScanner, char *buffer, BYTE lenBuffer);

Description: Used to get the host name number of the scanner or PC that shared the scanner via IP. This routine may simply return the IP number, like scanGetHostIP, if no name is registered with the IP number.

Parameters:

HSCANNER hScanner	Handle of the scanner.
char* buffer	Pointer to the buffer to get the host name in.
BYTE lenBuffer	Number of bytes in the buffer.

Returns: returns one of the following values:

0: Success.
1: This scanner not connected via TCP/IP.
2: Buffer is not big enough.

4.5 Scanner I/O functions

These functions are used to send and read data from a specific scanner.

4.5.1 scanTestUnitReady

Prototype: int scanTestUnitReady(HSCANNER hScanner);

Description: Sends a TEST UNIT READY command to the scanner.

Parameters: HSCANNER hScanner Handle of the scanner that the command is sent to.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.2 scanReserveUnit

Prototype: int scanReserveUnit(HSCANNER hScanner);

Description: Sends a RESERVE UNIT command to the scanner.

Parameters: HSCANNER hScanner Handle of the scanner that the command is sent to.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.3 scanReleaseUnit

Prototype: int scanReleaseUnit(HSCANNER hScanner);

Description: Sends a RELEASE UNIT command to the scanner.

Parameters: HSCANNER hScanner Handle of the scanner that the command is sent to.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.4 scanInquiry

Prototype: int scanInquiry(HSCANNER hScanner, BYTE *buffer, BYTE length);

Description: Sends an INQUIRY command to the scanner and returns the inquiry data in *buffer*.

Parameters:		
	HSCANNER hScanner	Handle to the scanner that the command is sent to.
	BYTE* buffer	Pointer to the area in which the inquiry data should be placed.
	BYTE length	Number of bytes to be transferred from the scanner.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.5 scanInquiryPage

Prototype: int scanInquiryPage(HSCANNER hScanner, BYTE *buffer, BYTE length, BYTE page);

Description: Sends an INQUIRY command to the scanner and returns the inquiry data of the specified page in *buffer*.

Parameters:		
	HSCANNER hScanner	Handle to the scanner that the command is sent to.
	BYTE* buffer	Pointer to the area in which the inquiry data should be placed.
	BYTE length	Number of bytes to be transferred from the scanner.
	BYTE page	The INQUIRY page to read.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

Note: Only to be used with fourth generation - or higher - scanners.

4.5.6 scanScan

Prototype: int scanScan(HSCANNER hScanner, BYTE *buffer, BYTE length);

Description: This function sends the scan command to the scanner.

Parameters:		
	HSCANNER hScanner	Handle of the scanner that the command is sent to.
	BYTE* buffer	Pointer to the buffer where the scan command data is placed (data is sent to the scanner).
	BYTE length	Length of *buffer* in bytes.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

Note: The *buffer* should contain the value 0 and have a *length* of 1 byte; other values may lead to unpredictable results.

4.5.7 scanRead

Prototype: int scanRead(HSCANNER hScanner, BYTE *buffer, int iRequested, BYTE DataType, WORD DataTypeQualifier, int *iReceived);

Description: This function sends a READ command to the scanner. The image data read from the scanner are stored in *buffer*.

Parameters:

HSCANNER hScanner	Handle of the scanner that the command is sent to.
BYTE* buffer	Pointer to the buffer where the READ data is placed (data is received from the scanner).
int iRequested	Specifies the number of bytes to try to read from the scanner.
BYTE DataType	Together with the DataTypeQualifier it specifies the kind of data to read. See FSS/ICD/005 for more information.
BYTE DataTypeQualifier	Together with the DataType it specifies the kind of data to read. See FSS/ICD/005 for more information.
int* iReceived	The number of bytes actually received.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

Note: This function can be used to read image data on all scanners.

4.5.8 scanSend

Prototype: int scanSend(HSCANNER hScanner, BYTE *buffer, int iLength, BYTE DataType, WORD DataTypeQualifier);

Description: This function sends a SEND command to the scanner. The data in *buffer* are send to the scanner.

Parameters:

HSCANNER hScanner	Handle of the scanner that the command is sent to.
BYTE* buffer	Pointer to the buffer where the READ data is placed (data is received from the scanner).
int iLength	The number of bytes to be send to the scanner.
BYTE DataType	Together with the DataTypeQualifier it specifies the kind of data to send. See FSS/ICD/005 for more information.
BYTE DataTypeQualifier	Together with the DataType it specifies the kind of data to send. See FSS/ICD/005 for more information.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.9 scanSetWindow

Prototype: int scanSetWindow(HSCANNER hScanner, BYTE *buffer, WORD length);

Description: This function sends a SET WINDOW command to the scanner. The buffer should be filled with the window data as specified in FSS/ICD/005.

Parameters:		
	HSCANNER hScanner	ID of the scanner that the command is sent to.
	BYTE* buffer	Pointer to the buffer where the SET WINDOW data is placed (data is sent to the scanner).
	WORD length	Specifies the number of bytes to send to the scanner.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.10 scanObjectPosition

Prototype: int scanObjectPosition(HSCANNER hScanner, int function, long absolutePosition);

Description: This function sends an OBJECT POSITION to the scanner.

Parameters:		
	HSCANNER hScanner	ID of the scanner that the command is sent to.
	int function	**SCAN_OBJ_POS_UNLOAD**: ejects the original
		SCAN_OBJ_POS_LOAD: returns the original to the start scanning position.
		SCAN_OBJ_POS_ABSOLUTE: Positions the paper at the absolute position specified by *absolutePosition*. Can only be used during scanning.
	long absolutePosition	Only valid in connection with the SCAN_OBJ_POS_ABSOLUTE function. Specifies the new scanning position in 1/1200 inch.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.11 scanWriteBuffer

Prototype: int scanWriteBuffer(HSCANNER hScanner, BYTE *buffer, BYTE mode, BYTE bufferId, DWORD off, DWORD length);

Description: This function sends a WRITE BUFFER command to the scanner.

Parameters:		
	HSCANNER hScanner	Handle of the scanner that the command is sent to.
	BYTE* buffer	Pointer to the buffer where the WRITE BUFFER data is placed (data is sent to the scanner).
	BYTE mode	Reserved; must be 1.
	BYTE bufferId	ID number of the buffer to write.
	DWORD off	Specifies the offset to start the write from.
	DWORD length	Specifies the number of bytes to send.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.12 scanReadBuffer

Prototype: int scanReadBuffer(HSCANNER hScanner, BYTE *buffer, BYTE mode, BYTE bufferId, DWORD off, DWORD length);

Description: This function sends a READ BUFFER command to the scanner.

Parameters:		
	HSCANNER hScanner	Handle of the scanner that the command is sent to.
	BYTE* buffer	Pointer to the buffer where the READ BUFFER data is placed (data is received from the scanner).
	BYTE mode	Reserved; must be 1.
	BYTE bufferId	ID number of the buffer to read.
	BYTE off	Specifies the offset to start the read from.
	DWORD length	Number of bytes to read from the scanner.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.13 scanReceiveDiagnostic

Prototype: int scanReceiveDiagnostic(HSCANNER hScanner, BYTE *buffer, WORD length);

Description: This function sends a RECEIVE DIAGNOSTIC command to the scanner.

Parameters:		
	HSCANNER hScanner	Handle of the scanner that the command is sent to.
	BYTE* buffer	Pointer to the buffer where the RECEIVE DIAGNOSTIC data is placed (data is sent to the scanner).
	WORD length	Number of data to send to the scanner.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.14 scanSendDiagnostic

Prototype: int scanSendDiagnostic(HSCANNER hScanner, BYTE *buffer, WORD length);

Description: This function sends a SEND DIAGNOSTIC command to the scanner.

Parameters:

HSCANNER hScanner	Handle of the scanner that the command is sent to.
BYTE* buffer	Pointer to the buffer where the SEND DIAGNOSTIC data is placed (data is sent to the scanner).
WORD length	Number of data to send to the scanner.

Returns: 0 if successful

< 0 on error. (Error codes defined with SCSI_ERROR_ prefix in CTX_SCAN_2000.H).

SCSI_STATUS_CHECK_CONDITION on check condition.

4.5.15 scanGetScannerNameToDisplay

Prototype: int scanGetScannerNameToDisplay (HSCANNER hScanner, BYTE *buffer, BYTE length);

Description: This function is used to get the scanner name with the same format as used in WIDEsystem.

Parameters:

HSCANNER hScanner	Handle of the scanner that the command is sent to.
BYTE* buffer	Pointer to the buffer where the scanner name is placed.
BYTE length	Length of the buffer. (The name is truncated when the buffer is too small.)

Returns: 0 if successful

SCSI_ERROR_PARM_ERROR on invalid scanner handle.

ERROR_NOT_SUPPORTED when running on a WIDEsystem version not implementing this function.

4.6 Error Messages

These functions are used to read localized error message corresponding to the error codes returned from the library.

4.6.1 scanGetErrorMessage

Prototype: int scanGetErrorMessage(int iAscAscq, char *message, int szMessage);

Description: This function returns the localized error message corresponding to a specific error code (which have been returned from ScanGetLastAsc()).

Parameters:		
	int iAscAscq	Error code to obtain string for. WORD with ASC in high BYTE and ASCQ in low BYTE.
	char* message	Here the string is returned
	int szMessage	Maximum length of message.

Returns: 0 if successful

E_FAIL if matching string was not found in resources.

4.6.2 scanGetDiagnosticsErrorMessage

Prototype: int scanGetDiagnosticsErrorMessage(int errorCode, char *message, int szMessage);

Description: Gets the localized error message corresponding to the give error code retrieved with scanReceiveDiagnostic on a 5G+ scanner.

Parameters:		
	int errorCode	Error code to obtain string for. The code is the error code returned in scanReceiveDiagnostic. Remember the error codes from scanReceiveDiagnostic must be byte swapped before a call to scanGetDiagnosticsErrorMessage()!
	char* message	Here the string is returned
	int szMessage	Maximum length of message.

Returns: 0 if successful

E_FAIL if matching string was not found in resources.

5 Accessing the CTX_SCAN_2000 functions

All the available functions are declared in the include file: CTX_SCAN_2000.H. This file should be included in order to use the functions.

To access the functions you need to include the file CTX_SCAN_2000_LOADER.CPP in your project. This file makes a delay load of the appropriate DLL library installed on the PC. The CPP files implements all the available functions by calling the equivalent function in the DLL library.

The CTX_SCAN_2000_LOADER.CPP file is found in the INCLUDE folder along and should be compiled without using precompiled header files.

The purpose of this implementation is to allow your application to be started even when WIDEsystem has not been installed on the PC, but the call to the first function scanOpenLib will of course fail with the error SCSI_ERROR_DLL_LOAD_FAILURE if the 32or 64 DLL library cannot be loaded.

Note: The delay load of the DLL library also ensures that the application may handle the situation where new interfaces are being used, but the WIDEsystem version installed does not implement the equivalent functions.
The function scanGetScannerNameToDisplay is an example of a function that is only available in WIDEsystem 4.0.0 and above.

5.1 The SimpleScan sample program

The Contex Software Development Kit contains the source code for the SimpleScan program.

The program is a small demonstration of how to make a scan and store the image in bitmap file. The program should be compiled using Visual Studio 2010 or higher. The solution can be compiled in both 32 and 64 bit.

6 Configuring your project to use the SDK

There are only two tasks that need to be done to use the low level SDK: Including the ctx_scan_2000.h header file and add the ctx_scan_2000_loader.cpp file to your project. Here we describe how to do this in Visual Studio 2010:

Open the Properties dialog for your project from the Project->Properties menu item. Select the **General** Page under the **C/C++ Settings** section. Modify the **Additional Include Directories** to include the absolute or relative path to the location of the ctx_scan_2000.h file.

Add the file ctx_scan_2000_loader.cpp to the source files in your project. Open Properties for the ctx_scan_2000_loader.cpp file and in the C/C++ section select “Precompiled Headers”. Set the “Precompiled Header” to “Not Using Precompiled Headers”.