Windows SDK Reference

All information subject to change without notice.

Document History

Author	Version	Date	Notes / Changes
Scott McGhee	01	4/23/10	Added csp2GetDSR
Scott McGhee	00	1/2/08	Initial Draft

Disclaimers

PLEASE READ THIS MANUAL CAREFULLY BEFORE INSTALLING OR USING THE PRODUCT.

Opticon Inc. does not warrant that the software documented here is error free, or that it meets any user's particular standards, requirements, or needs.

Opticon Inc. will in no event be liable for any direct, indirect, consequential or incidental damages arising out of use of the software documented here.

Trademarks

Trademarks used are the property of their respective owners.

Opticon Inc. and Opticon Sensors Europe B.V. are wholly owned subsidiaries of OPTOELECTRONICS Co., Ltd., 5-3, Tsukagoshi 5-chome, Warabi-shi, Saitama, Japan 335-0002. TEL +81-(0) 48-446-1183; FAX +81-(0) 48-446-1180

SUPPORT

USA

Phone: 425-651-2120

Email: support@opticonusa.com

Web: www.opticonusa.com

Europe

Email: support@opticon.com

Web: www.opticon.com

Contents

Table of Figures

1. Introduction

1.1. Purpose of this Document

The purpose of this document is to explain all the details of the OPN-2001 SDK, in order to facilitate its use in customized client applications that need to communicate with the OPN-2001 device.

1.2. Audience

The intended audience of this document is programmers charged with utilizing the OPN-2001 SDK to enable a client application to manipulate the device. It requires a working knowledge of C/C++/C# and Windows, and requires a copy of a 32-bit version of Windows. It assumes the reader is using some version of Visual Studio greater than or equal to Version 6.0 for the C/C++ version of the SDK, or greater than or equal to 2003 for the .NET version of the SDK (the examples in this document use Visual Studio 2005).

1.3. Scope of this Document

This document discusses the OPN-2001 SDK; in particular, it covers a description of the driver, proper usage of the driver in client projects, and a detailed description of all the API elements exported by the driver. Note that it does not cover the internal details and workings of the driver.

1.4. Terms and Acronyms

DLL	Dynamic-Link Library or Dynamically-Linked Library
API	Application Programming Interface
SDK	Software Development Kit
Driver, Library	In this context, they're used synonymously
USB	Universal Serial Bus

Table 1: Terms and Acronyms

2. Overview

2.1. Details

The OPN-2001 SDK is a user-space driver for the OPN-2001, in the form of a 32-bit dynamically-linked library (DLL). It provides a higher-level API for the serial-based interface needed to communicate with the OPN-2001, and it comes in two flavors: C/C++ and .NET (C#). Since the OPN-2001 was designed as a drop-in replacement for the Symbol CS1504, the former version of the driver has an API that is very much the same as the Symbol driver (though some functions are added only for compatibility reasons, as they are no longer needed; these will be noted in this document); the latter version is provided as a development option and is intended for developers with no prior investment in the Symbol CS1504 driver. *Application developers that were previously using the CS1504 driver (CSP2.DLL) will find that little, if any, of the code they've written needs to be changed to take advantage of the OPN-2001 C/C++ driver.*

Both the C/C++ and .NET flavors of the driver come in two versions: debug and release. With the debug version, the normal API is the same as before, but logging information can be outputted to a file for a number of different API functions (see section 2.2.3.g). This provides help for new users so that they can get their application debugged and running more quickly by getting a peak of what's going on under the hood, so to speak. When the application is working well, the debug version can be replaced with the release version of the DLL, which is smaller and faster than the debug version.

2.2. Usage

For complete, working examples of how to use the drivers, please see the sample project "OPN2001_DEMO" for the C/C++ driver and the project "Csp2.net.Test" for the .NET driver, included in the OPN-2001 SDK Kit. The following subsections can be used as a guide, in tandem with the sample projects, to illustrate the usage of the drivers.

2.2.1. Prerequisites

a) 32-bit Windows 95, 98, 2000, XP, or Vista

b) optousb driver package (serial-to-USB driver)

c) C/C++/C# compiler, linker (Visual Studio 2003 and higher are recommended)

d) The OPN-2001 SDK (CSp2.dll, CSp2.lib, CSp2.h, EGF31880.dll (C#))

Note: the .DLLs should be in the same directory as any application that use them, in the *\Windows\System* directory, or somewhere in the directories listed in the PATH environment variable. If the .DLLs are not in any of these locations, an error message will be displayed when the employing application is run. The .LIB and CSp2.h will be used for the C/C++ version of the driver only, and need to be included in your application's project. These files are **required.**

2.2.2. Visual Studio Settings

When using the C/C++ version of the driver, the .LIB file must be imported:

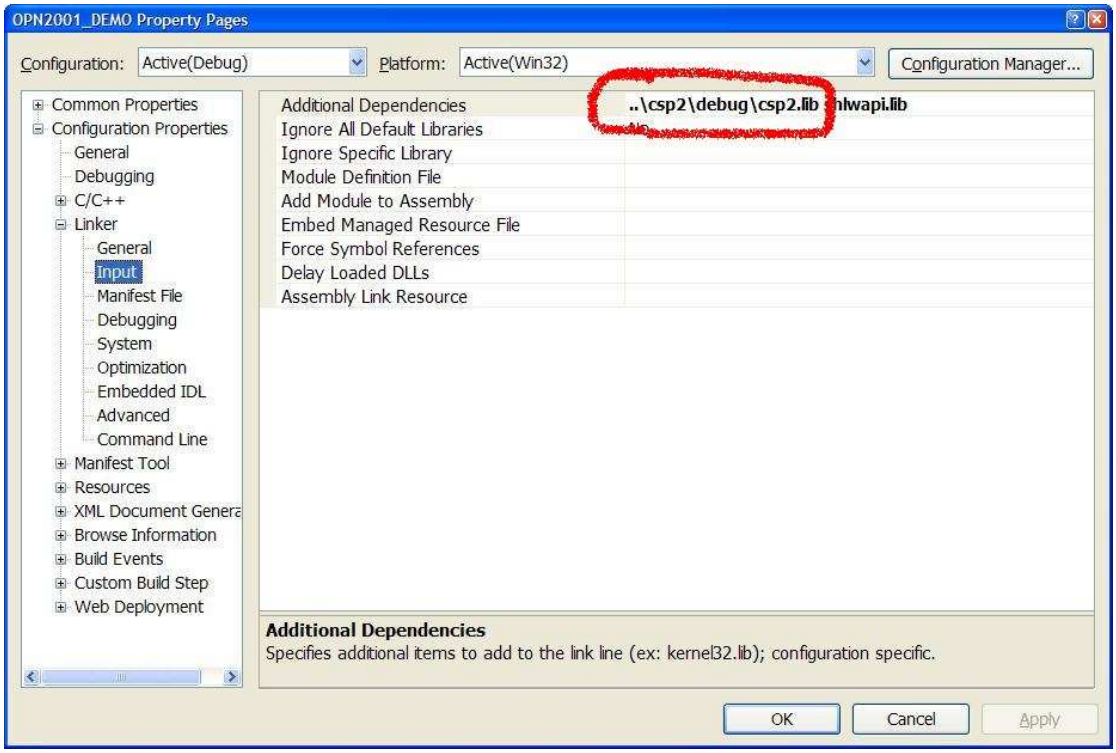

Figure 1: Importing the .LIB file into an application project

This .LIB file contains all the exported symbol information so that the application can look up the symbols in the .DLL when the latter is loaded at runtime.

The CSp2.h file must also be included in the code that will call into the driver's API:

```
#include "stdafx.h"
#include "OPN2001_DEMO.h"
#include "OPN2001_DEMODlg.h"
#include "csp2.h"          // csp2 dll header files
#include "SerialEnum.h" // serial port enumerator
#include <dbt.h>
```

Figure 2: Including the CSp2.h header

This header file contains all of the function prototypes and constants that will be needed to use the .DLL (the API).

2.2.3. Typical Usage Scenarios

a) Connecting to the OPN-2001

Prerequisites:

- OPN-2001 is connected to the computer via its USB cable
- optousb driver package is installed

- Developer has taken note of the COM port that has been assigned to the OPN-2001 (note: the COM port can be changed via Device Manager, or programmatically via the Windows registry)

All serial communications with the OPN-2001 via the driver are set at 9600 baud, 8 data bits, odd parity, and 1 stop bit. After all the above prerequisites are satisfied, all the application developer must provide the library is the chosen COM (serial) port in order to initiate a connection and communicate with the device. To do that, one calls the csp2InitEx() function, passing it the number of the desired COM port (valid numbers are 1-255). On successful return, the connection has been initialized, and communications can begin.

b) Checking for Barcode Data

Prerequisites:

- A connection has been successfully initialized via the csp2InitEx() function.

To check for barcode data in the OPN-2001, call the csp2DataAvailable() function, which returns TRUE if data is available, and FALSE if not.

c) Reading, Retrieving, and Deleting Barcode Data

Prerequisites:

- A connection has been successfully initialized via the csp2InitEx() function.

To read barcode data, one calls the csp2ReadData() function. This will return the number of barcodes that were read from the OPN-2001. After that, the csp2GetPacket() function should be called, which fills a passed-in char buffer with the requested barcode. This function only returns a single barcode; to read all the read barcodes, one must call it *n* number of times, where *n* is the value returned by csp2ReadData(). After that, should the developer want the barcode data inside the OPN-2001 cleared, the csp2ClearData() function should be called. This will erase the barcode data from the OPN-2001.

Note that the barcode data returned from csp2GetPacket() is returned as ASCII data only. There are a number of fields within the data returned by csp2GetPacket() which have special significance; see the detailed description of csp2GetPacket() in Section 4 for more detail on the format of this data and how to parse it. Two helper functions will be useful in helping to translate the symbology-type field and timestamp field after the data has been parsed: cps2GetCodeType() and csp2TimeStamp2StrEx(), respectively.

Also note that if the device doesn't have data, csp2ReadData() will just return a value of 0, indicating no barcodes are available.

d) Polling

Polling is an automatic mechanism by which the driver automatically detects whether or not the OPN-2001 has data, and if it does, an event is generated for client code to handle. This allows the client application to automatically get the barcode data from the OPN-2001, whenever the OPN-2001 has it.

Prerequisites:

- A connection has been successfully initialized via the csp2InitEx() function.

To start polling, call the csp2StartPolling() function, passing it a function pointer to a static callback function. This callback function is defined by the calling code and will be called when the event is raised, so it should do something useful in response to the event (e.g. read and get the barcode data).

To stop polling, call the csp2StopPolling() function.

e) Getting and Setting the Time of the Real-Time Clock (RTC)

Prerequisites:

- A connection has been successfully initialized via the csp2InitEx() function.

To get the current time of the RTC inside the OPN-2001, the csp2GetTime() function is called. This function will return a binary-formatted timestamp in a passed-in byte buffer. To convert the timestamp bytes into a human-readable string, the helper function csp2TimeStamp2StrEx() should be used.

To set the time of the RTC, the csp2SetTime() function is called. This function is passed a 6-byte buffer that contains the time and date information to set the RTC; see Section 4.10 for the format of the byte buffer.

f) Getting and Setting the OPN-2001 Parameters

Prerequisites:

- A connection has been successfully initialized via the csp2InitEx() function.

The OPN-2001 has a number of configurable parameters, or options, that can be read and written. (For a detailed list of all parameters and their values, please see Section 5.)

To get, or read, the value of a certain parameter, the csp2GetParam() function is used; it's passed the desired parameter (a key) and a small char buffer to hold the value of the parameter (a values corresponding to the key).

To set, or write, the value of a certain parameter, the csp2SetParam() function is used; it's passed the desired parameter and a small char buffer *containing* the value of the parameter.

To set all parameters back to their default values, the csp2SetDefaults() function is used.

g) Debugging

Prerequisites:

- A debug version of the library is being used.

In order to facilitate the debugging of applications that employ the OPN-2001 driver, the function csp2SetDebugMode() can be called to output a trace file ("debug.txt") in the current working directory. This provides debugging help and an insight to the sequence of operations taking place in the driver.

3. API – Constants

These apply to both the C/C++ and .NET versions of the driver library.

3.1. Function Return Status Codes

3.1.1. STATUS_OK

Value: 0

Return value of a function that executes with no errors.

3.1.2. COMMUNICATIONS_ERROR

Value: -1

Return value of a function that indicates there was a communication error.

3.1.3. BAD_PARAM

Value: -2

Return value of a function that indicates there was a bad parameter to the function.

3.1.4. SETUP_ERROR

Value: -3

Return value of a function that indicates the driver instance was in a state that didn't permit it to carry out the function.

3.1.5. INVALID_COMMAND_NUMBER

Value: -4

Return value of a function that indicates that the command sent to the driver was not defined.

3.1.6. COMMAND_LRC_ERROR

Value: -7

Return value of a function that indicates that the CRC check on written or read data failed.

3.1.7. RECEIVED_CHARACTER_ERROR

Value: -8

Return value of a function that indicates that the function encountered a value, returned from the device, that it did not expect.

3.1.8. GENERAL_ERROR

Value: -9

Return value of a function that indicates that some error occurred.

3.1.9. FILE_NOT_FOUND

Value: 2

Return value of a function that indicates that a file search failed.

3.1.10. ACCESS_DENIED

Value: 5

Return value of a function that indicates access to a requested resource was denied.

3.2. Other Constants

3.2.1. PARAM_OFF

Value: 0

Constant used as an argument for csp2SetDebugMode() and SetDebugMode() to turn off debugging (tracing).

3.2.2. PARAM_ON

Value: 1

Constant used as an argument for csp2SetDebugMode() and SetDebugMode() to turn on debugging (tracing).

3.2.3. DATA_AVAILABLE

Value: 1

Return value of csp2DataAvailable() and DataAvailable(), indicating that data is available in the OPN-2001.

3.2.4. NO_DATA_AVAILABLE

Value: 0

Return value of csp2DataAvailable() and DataAvailable(), indicating that no data is available in the OPN-2001.

3.2.5. DETERMINE_SIZE

Value: 0

Constant used as an argument for csp2GetPacket() and GetPacket(), which allows the function to return the length of a specific barcode without returning the data.

4. C/C++ API – Functions

4.1. csp2Init

Prototype: long CSP2_API csp2Init(long nCOMPort)

Description: Opens the COM port with one of the _serial_ports enum (see Section 5). The values that can be used are from COM1 to COM16. Do not use normal numbers, use the enumeration.

Arguments: *nCOMPort:* the constant _serial_ports number from COM1 to COM16.

Important: COM1 is used when nComport == 0 (see _serial_ports enum), COM16 is the highest port number that can be used; when higher ports are needed, use the csp2InitEx() function.

Returns: 0L on success, < 0L on failure.

4.2. csp2InitEx

Prototype: long CSP2_API csp2InitEx(long nCOMPort)

Description: Opens the COM port. This functions supports from COM1 to COM255. Do not use the constants from the _serial_ports enum, but use 1L for COM1, 2 for COM2 etc.

Argument: *nCOMPort:* the serial port number.

Important: COM1 is used when nCOMPort == 1, COM2 is used when nCOMPort == 2, etc.

Returns: 0L on success, < 0L on failure.

4.3. csp2Restore

Prototype: long CSP2_API csp2Restore(void)

Description: Closes the COM port that was opened by the csp2Init or csp2InitEx.

Returns: 0L on success, < 0L on failure.

4.4. csp2Wakeup

Prototype: long CSP2_API csp2Wakeup(void)

Description: Set the DTR (data-terminal-ready) signal high. This function is present for compatibility with Symbol's CSp2 library and isn't necessary for the OPN-2001.

Returns: 0L on success, < 0L on failure.

4.5. csp2DataAvailable

Prototype: long CSP2_API csp2DataAvailable(void)

Description: Checks if the CTS (clear-to-send) signal is ON. When the CTS line is ON Data is available in the OPN-2001.

Returns: > 0L for data available, 0L for no data available, < 0L on failure.

4.6. csp2ReadData

Prototype: long CSP2_API csp2ReadData(void)

Description: Read all barcode data from the OPN-2001 and store it in an internal buffer. To get barcode data from the internal buffer, use csp2GetPacket.

Returns: >= 0L indicating the amount of barcodes that are collected, < 0L on failure.

4.7. csp2ClearData

Prototype: long CSP2_API csp2ClearData(void)

Description: Removes all collected barcodes from the OPN-2001.

Returns: 0L on success, < 0L on failure.

4.8. csp2PowerDown

Prototype: long CSP2_API csp2PowerDown(void)

Description: Send a power down command to the OPN-2001. This function is present for compatibility with Symbol's CSp2 library and isn't necessary for the OPN-2001.

Returns: 0L on success, < 0L on failure.

4.9. csp2GetTime

Prototype: long CSP2_API csp2GetTime(unsigned char ucTime[])

Description: Get the current time from the OPN-2001.

Arguments: *ucTime* needs to be character buffer of at least 6 bytes. The format of the returned data:
ucTime[0] = Seconds
ucTime[1] = Minutes
ucTime[2] = Hours
ucTime[3] = Day
ucTime[4] = Month
ucTime[5] = Year

Returns: 0L on success, < 0L on failure.

4.10. csp2SetTime

Prototype: long CSP2_API csp2SetTime(unsigned char ucTime[])

Description: Set a new time and date in the OPN-2001.

Arguments: *ucTime* needs to be character buffer of at least 6 bytes. The format of the returned data:
ucTime[0] = Seconds
ucTime[1] = Minutes
ucTime[2] = Hours
ucTime[3] = Day
ucTime[4] = Month
ucTime[5] = Year

Returns: 0L on success, < 0L on failure.

4.11. csp2SetDefaults

Prototype: long CSP2_API csp2SetDefaults(void)

Description: Set the OPN-2001 into default mode, resetting any previously manipulated parameters.

Returns: 0L on success, < 0L on failure.

4.12. csp2GetPacket

Prototype: long CSP2_API csp2GetPacket(char szData[], long nBcrNr, long nMaxDataSz)

Description: Get data that was stored in the internal buffer by the csp2ReadData function.

Arguments: *szData:* buffer where the barcode data is stored in; *nBcrNr:* the desired barcode's index (which starts from 0L); *nMaxDataSz:* the maximum size of the szData buffer.

Format: The requested barcode data and its associated metadata (length, symbology ID, and timestamp) is returned in the *szData* buffer in the following format:

Length (1 byte)	Symbology ID (1 byte)	Barcode Data ((*Length* – 5) bytes)	Timestamp (4 packed bytes)

Figure 3: Format of returned barcode data

Note that the length byte does not include itself in its value. Also note that the symbology ID byte and the timestamp bytes can be extracted and passed to csp2GetCodeType() and csp2TimeStampToStringEx() to convert them into human-readable symbology names and date/time strings, respectively.

Returns: 0L on success, < 0L on failure.

4.13. csp2GetDeviceId

Prototype: long CSP2_API csp2GetDeviceId(char cDeviceId[8], long nMaxDeviceIdSz)

Description: Get the device ID number from the OPN-2001.

Arguments: *cDeviceId[8]*: the buffer where the device ID is copied into; *nMaxDeviceIdSz:* the maximum size of the *cDeviceId* buffer.

Important: In the *cDeviceId* buffer a hex respresentation of the Device ID is returned.

Returns: 0L on success, < 0L on failure.

4.14. csp2GetProtocol

Prototype: long CSP2_API csp2GetProtocol(void)

Description: Get the OPN-2001 protocol number.

Returns: >= 0L is the protocol version number, < 0L on failure.

4.15. csp2GetSystemStatus

Prototype: long CSP2_API csp2GetSystemStatus(void)

Description: Get the system status of the OPN-2001.

Returns: = STATUS_OK means the system is OK, = 22L signifies a low battery, < 0L on failure.

4.16. csp2GetSwVersion

Prototype: long CSP2_API csp2GetSwVersion(char szSwVersion[9], long nMaxSwVersionSz)

Description: Get the software version string from the OPN-2001.

Arguments: *szSwVersion[9]*: the buffer where the software version is copied to; *nMaxLength:* the maximum size of *szSwVersion*.

Returns: >= 0L for the needed buffer size to collect all version data including the '\0' character, < 0L on failure.

4.17. csp2GetASCIIMode

Prototype: long CSP2_API csp2GetASCIIMode(void)

Description: Returns a value telling if the OPN-2001 is in ASCII mode. Note: the only mode available is ASCII, so this is present for compatibility with the Symbol CSp2 library.

Returns: = 1L signifies ASCII mode ON, = 0L signifies ASCII mode OFF, < 0L on failure.

4.18. csp2GetRTCMode

Prototype: long CSP2_API csp2GetRTCMode(void)

Description: Return value tells if the OPN-2001 adds a timestamp to the collected barcode data.

Returns: = 1L signifies timstamp appended to barcode data, = 0L signifies no timestamp is used, < 0L on failure.

4.19. csp2SetRetryCount

Prototype: long CSP2_API csp2SetRetryCount(long nRetries)

Description: Set the number of interrogation retries.

Arguments: *nRetries:* specifies the number interrogation retries; values from 1L to 9L are supported.

Returns: STATUS_OK on success, < 0L on failure.

4.20. csp2GetRetryCount

Prototype: long CSP2_API csp2GetRetryCount(void)

Description: Get the number of interrogation retries.

Returns: 1 to 9, signifying the set number of retries.

4.21. csp2GetDllVersion

Prototype: long CSP2_API csp2GetDllVersion(char szDllVersion[], long nMaxDllVersionSz)

Description: Get the version of the DLL.

Arguments: *szDllVersion[]:* buffer that the DLL version is copied into; *nMaxDllVersionSz:* the maximum size of the *szDllVersion*.

Returns: STATUS_OK on success, FILE_NOT_FOUND on failure.

4.22. csp2TimeStamp2Str

Prototype: long CSP2_API csp2TimeStamp2Str(unsigned char *ucTimeStamp, char *szTimeStampResult, long nMaxTimeStampResult)

Description: Convert the packed *ucTimeStamp* array to a normal date/time string. This implementation has some problems but is present for compatibility with Symbol's CSp2 library.

Arguments: *ucTimeStamp:* pointer to the packed 4 byte timestamp; *szTimeStampResult:* string representation of the packed *ucTimeStamp* array; *nMaxTimeStampResult:* the maximum size of the *szTimeStampResult* buffer.

Format: The format of the 4 packed bytes in *ucTimeStamp*:

Seconds (6 bits)	Minutes (6 bits)	Hours (5 bits)	Days (5 bits)	Months (4 bits)	Years (6 bits)

Figure 4: Format of packed timestamps

Important: The *szTimeStampResult* buffer needs to be at least 21 bytes in size. The date/time string format: HH:MM:SS <AM or PM> MM/DD/YYYY

Returns: STATUS_OK on success, < 0L on failure.

4.23. csp2TimeStamp2StrEx

Prototype: long CSP2_API csp2TimeStamp2StrEx(unsigned char *ucTimeStamp, char *szTimeStampResult, long nMaxTimeStampResult)

Description: Convert the packed *ucTimeStamp* array to a normal date/time string. This is the correct implementation, and should be used in the stead of csp2TimeStamp2Str.

Arguments: *ucTimeStamp:* pointer to the packed 4 byte timestamp; *szTimeStampResult:* string representation of the packed *ucTimeStamp* array; *nMaxTimeStampResult:* the maximum size of the *szTimeStampResult* buffer.

Format: The format of the 4 packed bytes in *ucTimeStamp*:

Seconds (6 bits)	Minutes (6 bits)	Hours (5 bits)	Days (5 bits)	Months (4 bits)	Years (6 bits)

Important: The *szTimeStampResult* buffer needs to be at leat 21 bytes in size. The date/time string format: HH:MM:SS <AM or PM> MM/DD/YYYY

Returns: STATUS_OK on success, < 0L on failure.

4.24. csp2GetCodeType

Prototype: long CSP2_API csp2GetCodeType(unsigned long nBcrCodeID, char *szCodeTypeBuf, long nMaxCodeTypeBuf)

Description: Convert the OPN-2001 barcode symbology ID to a human-readable string.

Arguments: *nBcrCodeID:* integer value of the symbology ID; *szCodeTypeBuf:* pointer to the result buffer; *nMaxCodeTypeBuf:* the maximum size of the *szCodeTypeBuf* buffer.

Returns: STATUS_OK on success, < 0L on failure.

4.25. csp2ReadRawData

Prototype: long CSP2_API csp2ReadRawData(char aResultBuf[], long nMaxResultBuf)

Description: Read data from the OPN-2001 and store it in *aResultBuf* until *nMaxResultBuf* bytes are read. When *nMaxResultBuf* == 0L, csp2ReadRawData will return the size of the result buffer needed to collect all the data from the OPN-2001, and nothing in *aResultBuf*.

Arguments: *aResultBuf:* result buffer the read data is copied into; *nMaxResultBuf:* the maximum size of the *aResultBuf*.

Returns: >= 0L signifying the the needed buffer size to collect all data or < 0L on failure.

4.26. csp2SetParam

Prototype: long CSP2_API csp2SetParam(long nParamNr, char szNewParam[], long nMaxNewParam)

Description: Set one individual device parameter (option) value.

Arguments: *nParamNr:* the number of the parameter to change; *szNewParam:* holds the new parameter value; *nMaxNewParam:* specifies the amount of characters in *szNewParam*.

Returns: STATUS_OK on success, < 0L on failure.

4.27. csp2GetParam

Prototype: long CSP2_API csp2GetParam(long nParamNr, char szResultParam[], long nMaxResultParam)

Description: Read one individual device parameter (option) value.

Arguments: *nParamNr:* the number of the parameter to read; *szResultParam:* pointer to the buffer that will contain the read parameter value; *nMaxLength:* specifies the maximum size of *szResultParam* buffer.

Returns: STATUS_OK on success, < 0L on failure.

4.28. csp2Interrogate

Prototype: long CSP2_API csp2Interrogate(void)

Description: Request a response from the OPN-2001 to detect whether or not it is present. This function will also fill internal buffers with the software version, protocol version and system status, which can then be read by the appropriate functions.

Returns: STATUS_OK on success, < 0L on failure.

4.29. csp2GetCTS

Prototype: long CSP2_API csp2GetCTS(void)

Description: Returns the CTS (clear-to-send) signal status.

Returns: 1L == ON, 0L == OFF, < 0L on failure.

4.30. csp2SetDTR

Prototype: long CSP2_API csp2SetDTR(long nOn)

Description: Set the DTR (data-terminal-ready) flow control line status.

Arguments: nOn == 1L set DTR line ON (high), nOn == 0L clears DTR line OFF (low).

Returns: STATUS_OK on success, < 0L on failure.

4.31. csp2SetDebugMode

Prototype: long CSP2_API csp2SetDebugMode(long nOn)

Description: Sets the debug mode of the debug version of the driver.

Arguments: nOn == 1L means the debug mode is ON, nOn == 0L means the debug mode is OFF.

Important: Only works with the debug version of the CSP2.DLL.

Returns: STATUS_OK on success, < 0L on failure.

4.32. csp2StartPolling

Prototype: long CSP2_API csp2StartPolling(FARPROC csp2CallBack)

Description: This function creates a new thread to poll for the OPN-2001. The *csp2CallBack* function address is called by the thread to notify that a OPN-2001 has been detected after having been removed.

Arguments: *csp2CallBack:* address of the callback function. Format of callback function: int __stdcall csp2PollCallBack(void)

Returns: STATUS_OK on success, < 0L on failure.

4.33. csp2StopPolling

Prototype: long CSP2_API csp2StopPolling(void)

Description: Stops the polling thread and removes it from memory.

Returns: STATUS_OK on success, < 0L on failure.

4.34. csp2EnablePolling

Prototype: long CSP2_API csp2EnablePolling(void)

Description: Signals the polling thread to resume operation.

Returns: STATUS_OK on success, < 0L on failure.

4.35. csp2DisablePolling

Prototype: long CSP2_API csp2DisablePolling(void)

Description: Signals the polling thread to go into suspend mode.

Returns: STATUS_OK on success, < 0L on failure.

4.36. csp2GetCommInfo

Prototype: long CSP2_API csp2GetCommInfo(long nCOMPort)

Obsolete. This function is only present for compatibility with the Symbol CSp2 library.

4.37. csp2SetMultiParam

Prototype: long CSP2_API csp2SetMultiParam(char szMultiParams[], long nMaxMultiParams)

Obsolete. This function is only present for compatibility with the Symbol CSp2 library.

4.38. csp2GetDSR

Prototype: long CSPW_API csp2GetDSR(void)

Description: Returns the DSR (data-signal-ready) signal status.

Returns: 1 for ON, 0 for OFF, < 0L on failure.

5. .NET API – Functions

5.1. Init

Prototype: Int32 Init(Int32 nComPort)

Description: Opens the COM port. This functions supports from COM1 to COM255. Do not use the constants from the _serial_ports enum, but use 1L for COM1, 2 for COM2 etc.

Argument: *nComPort:* the serial port number.

Important: COM1 is used when nCOMPort == 1, COM2 is used when nCOMPort == 2, etc.

Returns: 0L on success, < 0L on failure.

5.2. Restore

Prototype: Int32 Restore()

Description: Closes the COM port that was opened by Init().

Returns: 0L on success, < 0L on failure.

5.3. WakeUp

Prototype: Int32 WakeUp()

Description: Set the DTR (data-terminal-ready) signal high. This function isn't necessary for communicating with the OPN-2001.

Returns: 0L on success, < 0L on failure.

5.4. DataAvailable

Prototype: Int32 DataAvailable()

Description: Checks if the CTS (clear-to-send) signal is ON. When the CTS line is ON Data is available in the OPN-2001.

Returns: > 0L for data available, 0L for no data available, < 0L on failure.

5.5. ReadData

Prototype: Int32 ReadData()

Description: Read all barcode data from the OPN-2001 and store it in an internal buffer. To get barcode data from the internal buffer, use GetPacket().

Returns: >= 0L indicating the amount of barcodes that are collected, < 0L on failure.

5.6. ClearData

Prototype: Int32 ClearData()

Description: Removes all collected barcodes from the OPN-2001.

Returns: 0L on success, < 0L on failure.

5.7. PowerDown

Prototype: Int32 PowerDown()

Description: Send a power down command to the OPN-2001. This function isn't necessary for proper use of the OPN-2001.

Returns: 0L on success, < 0L on failure.

5.8. GetTime

Prototype: Int32 GetTime(out DateTime Time)

Description: Get the current time from the OPN-2001.

Arguments: *Time:* variable that will hold the date & time of the RTC in the OPN-2001.

Returns: 0L on success, < 0L on failure.

5.9. SetTime

Prototype: Int32 SetTime(DateTime Time)

Description: Set a new time and date in the OPN-2001.

Arguments: *Time:* variable that holds the date & time value to set the RTC to.

Returns: 0L on success, < 0L on failure.

5.10. SetDefaults

Prototype: Int32 SetDefaults()

Description: Set the OPN-2001 into default mode, resetting any previously manipulated parameters.

Returns: 0L on success, < 0L on failure.

5.11. TimeStampToDateTime

Prototype: DateTime TimeStampToDateTime(Byte[] Stamp)

Description: Convert the packed *Stamp* array to an object of type DateTime.

Arguments: *Stamp:* the packed 4 byte timestamp.

Format: The format of the 4 packed bytes in *Stamp*:

Seconds (6 bits)	Minutes (6 bits)	Hours (5 bits)	Days (5 bits)	Months (4 bits)	Years (6 bits)

Returns: A DateTime object representing the date & time information stored in *Stamp*.

5.12. GetPacket

Prototype: Int32 GetPacket(out BarCodeDataPacket aPacket, Int32 nBarcodeNumber)

Description: Get data that was stored in the internal buffer by the ReadData() function.

Arguments: *aPacket:* variable that holds the barcode data and its associated metadata upon return; *nBarcodeNumber:* the desired barcode's index (which starts from 0).

Format: See Section 6.3 for the structure of the BarCodeDataPacket object.

Returns: 0L on success, < 0L on failure.

5.13. GetDeviceId

Prototype: Int32 GetDeviceId(out String DeviceId)

Description: Get the device ID from the OPN-2001.

Arguments: *DeviceId*: a string variable that will contain the device ID upon return.

Returns: 0L on success, < 0L on failure.

5.14. GetProtocol

Prototype: Int32 GetProtocol()

Description: Get the OPN-2001 protocol number.

Returns: >= 0L is the protocol version number, < 0L on failure.

5.15. GetSystemStatus

Prototype: Int32 GetSystemStatus()

Description: Get the system status of the OPN-2001.

Returns: = STATUS_OK means the system is OK, = 22L signifies a low battery, < 0L on failure.

5.16. GetSwVersion

Prototype: Int32 GetSwVersion(System.Text.StringBuilder szSwVersion, Int32 nMaxLength)

Description: Get the software version string from the OPN-2001.

Arguments: *szSwVersion*: the variable the software version is copied into; *nMaxLength:* the maximum size of *szSwVersion*.

Returns: >= 0L for the needed buffer size to collect all version data including the '\0' character, < 0L on failure.

5.17. GetASCIIMode

Prototype: Int32 GetASCIIMode()

Description: Returns a value telling if the OPN-2001 is in ASCII mode. Note: the only mode available is ASCII, so this function is not necessary for proper use of the OPN-2001.

Returns: = 1L signifies ASCII mode ON, = 0L signifies ASCII mode OFF, < 0L on failure.

5.18. GetRTCMode

Prototype: Int32 GetRTCMode()

Description: Return value tells if the OPN-2001 adds a timestamp to the collected barcode data.

Returns: = 1L signifies timstamp appended to barcode data, = 0L signifies no timestamp is used, < 0L on failure.

5.19. SetRetryCount

Prototype: Int32 SetRetryCount(Int32 nRetryCount)

Description: Set the number of interrogation retries.

Arguments: *nRetryCount:* specifies the number interrogation retries; values from 1L to 9L are supported.

Returns: STATUS_OK on success, < 0L on failure.

5.20. GetRetryCount

Prototype: Int32 GetRetryCount()

Description: Get the number of interrogation retries.

Returns: 1 to 9, signifying the set number of retries.

5.21. GetDllVersion

Prototype: Int32 GetDllVersion(System.Text.StringBuilder szDllVersion, Int32 nMaxLength)

Description: Get the version of the DLL.

Arguments: *szDllVersion:* variable that the DLL version is copied into; *nMaxLength:* the maximum size of the *szDllVersion*.

Returns: STATUS_OK on success, FILE_NOT_FOUND on failure.

5.22. TimeStampToStr

Prototype: Int32 TimeStampToStr([Out, MarshalAs(UnmanagedType.LPArray)] Byte[] Stamp, System.Text.StringBuilder szTimeStampResult, Int32 nMaxLength)

Description: Convert the packed *Stamp* array to a normal date/time string. This implementation is obsolete; TimeStampToStrEx() should be used.

Arguments: *Stamp:* the packed 4 byte timestamp; *szTimeStampResult:* string representation of the packed *ucTimeStamp* array; *nMaxLength:* the maximum size of the *szTimeStampResult* string.

Format: The format of the 4 packed bytes in *Stamp*:

Seconds (6 bits)	Minutes (6 bits)	Hours (5 bits)	Days (5 bits)	Months (4 bits)	Years (6 bits)

Important: The date/time string format: HH:MM:SS <AM or PM> MM/DD/YYYY

Returns: STATUS_OK on success, < 0L on failure.

5.23. TimeStampToStrEx

Prototype: Int32 TimeStampToStrEx([Out, MarshalAs(UnmanagedType.LPArray)] Byte[] Stamp, System.Text.StringBuilder szTimeStampResult, Int32 nMaxLength)

Description: Convert the packed *Stamp* array to a normal date/time string. This is the correct implementation, and should be used in the stead of TimeStampToStr().

Arguments: *Stamp:* the packed 4 byte timestamp; *szTimeStampResult:* string representation of the packed *ucTimeStamp* array; *nMaxLength:* the maximum size of the *szTimeStampResult* string.

Format: The format of the 4 packed bytes in *Stamp*:

Seconds (6 bits)	Minutes (6 bits)	Hours (5 bits)	Days (5 bits)	Months (4 bits)	Years (6 bits)

Important: The date/time string format: HH:MM:SS <AM or PM> MM/DD/YYYY

Returns: STATUS_OK on success, < 0L on failure.

5.24. GetCodeType

Prototype: Int32 GetCodeType(Int32 CodeID, System.Text.StringBuilder CodeType, Int32 nMaxLength)

Description: Convert the OPN-2001 barcode symbology ID to a human-readable string.

Arguments: *CodeID:* integer value of the symbology ID; *CodeType:* string variable in which the symbology name will be returned; *nMaxLength:* the maximum size of the *CodeType* string.

Returns: STATUS_OK on success, < 0L on failure.

5.25. ReadRawData

Prototype: Int32 ReadRawData([Out, MarshalAs(UnmanagedType.LPArray)] Byte[] aBuffer, Int32 nMaxLength)

Description: Read data from the OPN-2001 and store it in *aBuffer* until *nMaxLength* bytes are read. When *nMaxLength* == 0, ReadRawData() will return the size of the result buffer needed to collect all the data from the OPN-2001, and nothing in *aBuffer*.

Arguments: *aBuffer:* result buffer the read data is copied into; *nMaxLength:* the maximum size of the *aBuffer*.

Returns: >= 0L signifying the the needed buffer size to collect all data or < 0L on failure.

5.26. SetParam

Prototype: Int32 SetParam(Int32 nParam, [In, MarshalAs(UnmanagedType.LPArray)] Byte[] szString, Int32 nMaxLength)

Description: Set one individual device parameter (option) value.

Arguments: *nParam:* the number of the parameter to change; *szString:* holds the new parameter value; *nMaxLength:* specifies the amount of characters in *szString*.

Returns: STATUS_OK on success, < 0L on failure.

5.27. GetParam

Prototype: Int32 GetParam(Int32 nParam, [Out, MarshalAs(UnmanagedType.LPArray)] Byte[] szString, Int32 nMaxLength)

Description: Read one individual device parameter (option) value.

Arguments: *nParam:* the number of the parameter to read; *szString:* buffer that will contain the read parameter value; *nMaxLength:* specifies the maximum size of *szString* buffer.

Returns: STATUS_OK on success, < 0L on failure.

5.28. Interrogate

Prototype: Int32 Interrogate()

Description: Request a response from the OPN-2001 to detect whether or not it is present. This function will also fill internal buffers with the software version, protocol version and system status, which can then be read by the appropriate functions.

Returns: STATUS_OK on success, < 0L on failure.

5.29. GetCTS

Prototype: Int32 GetCTS()

Description: Returns the CTS (clear-to-send) signal status.

Returns: 1L == ON, 0L == OFF, < 0L on failure.

5.30. SetDTR

Prototype: Int32 SetDTR(Int32 nOnOff)

Description: Set the DTR (data-terminal-ready) flow control line status.

Arguments: *nOnOff* == 1L set DTR line ON (high), *nOnOff* == 0L clears DTR line OFF (low).

Returns: STATUS_OK on success, < 0L on failure.

5.31. SetDebugMode

Prototype: Int32 SetDebugMode(Int32 nOnOff)

Description: Sets the debug mode of the debug version of the driver.

Arguments: *nOnOff* == 1L means the debug mode is ON, *nOnOff* == 0L means the debug mode is OFF.

Important: Only works with the debug version of the CSP2.DLL.

Returns: STATUS_OK on success, < 0L on failure.

5.32. StartPolling

Prototype: Int32 StartPolling(csp2CallBackFunction csp2CallBack)

Description: This function creates a new thread to poll for the OPN-2001. The *csp2CallBack* function address is called by the thread to notify that a OPN-2001 has been detected after having been removed.

Arguments: *csp2CallBack:* address of the callback function. Format of callback function: delegate

void csp2CallBackFunction()

Returns: STATUS_OK on success, < 0L on failure.

5.33. StopPolling

Prototype: Int32 StopPolling()

Description: Stops the polling thread and removes it from memory.

Returns: STATUS_OK on success, < 0L on failure.

5.34. EnablePolling

Prototype: Int32 EnablePolling()

Description: Signals the polling thread to resume operation.

Returns: STATUS_OK on success, < 0L on failure.

5.35. DisablePolling

Prototype: Int32 DisablePolling()

Description: Signals the polling thread to go into suspend mode.

Returns: STATUS_OK on success, < 0L on failure.

6. API – Data

Unless otherwise noted, this section applies to both the C/C++ and .NET versions of the driver library.

6.1. typedef enum __serial_ports (C/C++)

Description: An enumeration whose values are used as arguments to csp2Init(), in order to specify the COM port to which the OPN-2001 is attached. This is defined in CSp2.h.

Structure:

```
typedef enum __serial_ports {
        COM1 = 0,
        COM2,
        COM3,
        COM4,
        COM5,
        COM6,
        COM7,
        COM8,
        COM9,
```

```
        COM10,
        COM11,
        COM12,
        COM13,
        COM14,
        COM15,
        COM16
} _serial_ports;
```

Note: This should only be used with the csp2Init(), and csp2InitEx() is preferential to csp2Init().

6.2. Parameters

6.2.1. Symbology Parameters

Parameter	Comments	Min/Max Values	Default	Param Num
Code-39		1=Enable / 0=Disable	Enabled	31 (0x1f)
UPC		1=Enable / 0=Disable	Enabled	9 (0x09)
Code-128		1=Enable / 0=Disable	Enabled	8 (0x08)
Code-39 Full ASCII		1=Enable / 0=Disable	Disabled	54 (0x36)
UPC Supps		0=No Supps 1=Supps only 2=Auto-D	2=Auto-D	53 (0x35)
Convert UPC-E to A		1=Enable / 0=Disable	Disabled	41 (0x29)
Convert EAN-8 to EAN-13		1=Enable / 0=Disable	Disabled	42 (0x2a)
Convert EAN-8 to EAN-13 Type	Set code type of converted EAN8 barcode to EAN8 or EAN13	1=Enable / 0=Disable	Disabled	55 (0x37)
Send UPC-A Check Digit		1=Enable / 0=Disable	Enabled	43 (0x2b)
Send UPC-E Check Digit		1=Enable / 0=Disable	Enabled	44 (0x2c)
Code-39 Check Digit		1=Enable / 0=Disable	Disabled	46 (0x2e)
Xmit Code-39 Check Digit		1=Enable / 0=Disable	Disabled	45 (0x2d)
UPC-E Preamble		0=None 1=System char 2=Sys char & country code	1 = System Char	37 (0x25)

UPC-A Preamble		0=None 1=System char 2=Sys char & country code	1 = System Char	36 (0x24)
EAN-128		1=Enable / 0=Disable	Enabled	52 (0x34)
Coupon Code		1=Enable / 0=Disable	Enabled	56 (0x38)
I 2of5		1=Enable / 0=Disable	Enabled	58 (0x3A)
I 2of5 Check Digit		0=Disabled 1=USS check digit 2=OPCC check digit	Disabled	65 (0x41)
Xmit I 2of5 Check Digit		1=Enable / 0=Disable	Disabled	64 (0x40)
Convert ITF14 to EAN-13		1=Enable / 0=Disable	Disabled	63 (0x3F)
I 2of5 Length 1, Length 2	One Discrete Length Two Discrete Lengths Length Within Range Any Length	L1 = length, L2 = 0 L1 > L2 L1 < L2 L1 = 0, L2 = 0	L1 = 14, L2 = 0	59 (0x3B) 60 (0x3C)
D 2of5		1=Enable / 0=Disable	Disabled	57 (0x39)
D 2of5 Length 1, Length 2	One Discrete Length Two Discrete Lengths Length Within Range Any Length	L1 = length, L2 = 0 L1 > L2 L1 < L2 L1 = 0, L2 = 0	L1 = 12, L2 = 0	61 (0x3D) 62 (0x3E)
UPC/EAN Security Level		0 – 3	0	47 (0x2f)
UPC/EAN Supplemental Redundancy (No_supp_max)	# of times to decode UPC/EAN barcode without supplementals	2 – 20	5	48 (0x30)

Table 2: Symbology parameters

6.2.2. Device Parameters

Parameter	Comments	Min/Max Values	Rev B Defaults	Param Num
Scanner On-Time		1 sec. – 10 sec. in 100 msec increments	30 (3.0 sec)	17 (0x11)
Volume		0=Off 1=On	1 = On	2 (0x02)

Comm Awake Time	How long scanner will stay awake for host communication	1 – 6 (20 seconds – 2 minutes in 20sec. Increments)	(1) 20 seconds	32 (0x20)
Baud Rate		3 = 300 4 = 600 5 = 1200 6 = 2400 7 = 4800 8 = 9600 9 = 19200	9600	13 (0x0d)
Baud Switch Delay	How long scanner will delay before sending a response to a new baud rate command	0 – 1second in 10ms increments	35 (350 ms)	29 (0x1d)
Reset Baud Rates	Determines if default baud rate will be used on power-up	1=Enable / 0=Disable	Enabled	28 (0x1c)
Reject Redundant Barcode	Disabling will allow same barcode to be stored consecutively	1=Enable / 0=Disable	Disabled	4 (0x04)
Host Connect Beep		1=Enable / 0=Disable	Enabled	10 (0x0a)
Host Complete Beep		1=Enable / 0=Disable	Enabled	11 (0x0b)
Low-Battery Indication	Determines how low battery condition will be handled.	0=No indication / No operation 1=No Indication / Allow operation 2=Indicate / No operation 3=Indicate / Allow operation.	3	7 (0x07)
Auto Clear	Clear barcodes after upload.	1=Enable / 0=Disable	Disabled	15 (0x0f)
Delete Enable	Determines operation of delete and clear all functions.	0= Delete Disabled/Clear All Disabled 1=Delete Disabled/Clear All Enabled 2=Delete Enabled/Clear All Disabled 3=Delete Enabled/Clear All Enabled 4=Radio Stamp 5=VDIU Voluntary Device Initiated Upload	3 = Delete Enabled /Clear All Enabled	33 (0x21)

Data Protection		1=Enable / 0=Disable	Disabled	49 (0x31)
Memory Full Indication		1=Enable / 0=Disable	Enabled	50 (0x32)
Memory Low Indication	EEPROM is 90% full	1=Enable / 0=Disable	Disabled	51 (0x33)
Max Barcode Len	Increased to 30 for Coupon Code	1 – 30	30	34 (0x22)
Good Decode LED On Time		250 ms. – 1 sec. in 250msec. Increments	4 (1.0 sec)	30 (0x1e)
Store RTC	Store Real Time Clock data	1=Enable / 0=Disable	Enabled	35 (0x23)
ASCII mode	Allows user to choose how unencrypted data is to be sent.	0 = Like encrypted data (CS-1504 Mode) 1 = As ASCII strings (like CS-2000)	0	79 (0x4f)
Beeper Toggle	Allows user to toggle beeper On/Off with the scan key	0 = No 1 = Yes	1	85 (0x55)
Beeper Auto On	Automatically turns on a beeper toggled off by the scan key after 8 hours	0 = No 1 = Yes	0	86 (0x56)
Scratch Pad	A 32 byte storage area unused by the CS-1504. For customer use.	None	No Default Values	38 (0x26)

Table 3: Device parameters

6.3. class BarCodeDataPacket (.NET)

Description: simple class that holds a single barcode and its associated metadata. Used with the GetPacket() method; only applies to the .NET version of the driver.

Structure:

```
public class BarCodeDataPacket
{
        public Int32 iId;                   //symbology ID
        public String strId;                //human-readable symbology name
        public String strBarData;           //barcode data
        public DateTime dtTimestamp;        //timestamp; only valid if RTC mode is enabled
}
```