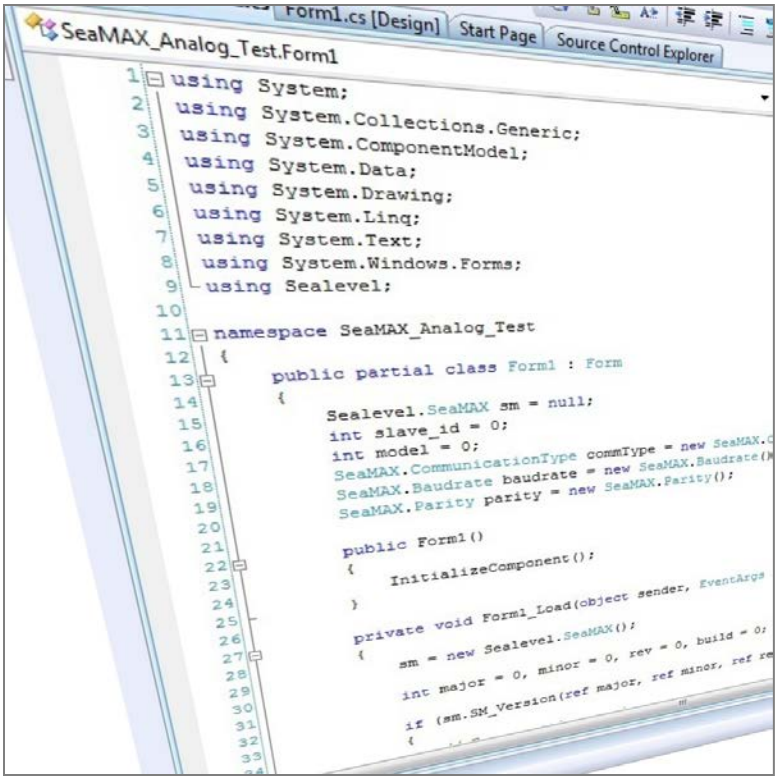
SeaMAX™ Software API Manual

USER MANUAL



How to Get Assistance

When calling for technical assistance, please have the device installed and ready to run diagnostics. If possible, have your hardware manual and current settings ready.

The Sealevel website is an excellent resource located at www.sealevel.com. The most current software updates and user manuals are available via our homepage by clicking on the 'Product Manuals' or 'Software Drivers' links located under ‘Support’. Manuals and software can also be downloaded from the product page for your device.

The support section of our website provides answers for many common questions. Refer to this helpful resource by visiting http://www.sealevel.com/support/.

TECHNICAL SUPPORT

Monday – Friday
8:00 am to 5:00 pm EST
Phone: +1 (864) 843-4343
Email: support@sealevel.com

ISO 9001:2000

SL9210 Revision 10/2009
As further evidence to Sealevel’s commitment to Total Customer Satisfaction the Company’s Management System has achieved registration to ISO 9001:2000 in 2002. This provides one of the strongest assurances of product/service quality available. ISO 9001:2000 registration ensures Sealevel’s customers that the proper processes and business practices are in place to guarantee their satisfaction. Sealevel is audited by Quality Management Institute (QMI), North America’s largest management system registrar.

TRADEMARKS

Sealevel Systems, Incorporated acknowledges that all trademarks referenced in this manual are the service mark, trademark, or registered trademark of the respective company.

Contents

Chapter 1

SeaMAX API Documentation

1.1 License Agreement

This software is licensed solely for use with a Sealevel Systems, Incorporated product. The user may operate the software only when utilizing the affiliated Sealevel Systems, Incorporated product.

This license is in effect until terminated by Sealevel Systems, Incorporated or the user. The user may terminate the license only if the provided software, any copies, modifications and enhancements are returned or destroyed.

In addition to all remedies, which Sealevel Systems, Incorporated may have legal and equitable, if user should breach or threaten to breach, the provisions of this license, Sealevel Systems, Incorporated may individually and without notice, terminate the license granted here under. At such time all copies, modifications and enhancements must be returned or destroyed.

By installing the software provided, the user acknowledges and agrees to all the terms and conditions of this License Agreement.

1.2 Introduction

Sealevel digital and analog I/O modules supported by the SeaMAX software suite are designed to work with third party applications via the SeaMAX API. To help simplify application development, the following documentation details the functions of the SeaMAX API. To help you get started, example code is included for popular languages and compilers.

There are four modules that are included in the SeaMAX API:

- **SeaMAX API**
  The foundation of SeaMAX with functions for configuring, interfacing, and modifying supported Sealevel digital I/O modules and devices.
- **SeaMAX Ethernet Discovery / Configuration API**
  The Ethernet module contains functions related to the discovery and configuration of Sealevel I/O devices with an Ethernet interface.
- **SeaDAC Lite Discovery API**
  The SeaDAC Lite module contains the programmatic functions for locating and identifying SeaDAC Lite modules.
- **Version 2.x Deprecated Interface**
  SeaMAX v3 has been completely rewritten with a focus on increasing speed and improving reliability. This module provides backwards compatibility with legacy SeaMAX v2 code.

1.3 Getting Started

The SeaMAX API manual contains numerous documents to assist you with application development. Three key documents are highlighted in order of importance.

- SeaMAX by Sealevel Model Number
  Provides a quick breakdown by model number of all SeaMAX API functions useful to specific Sealevel I/O devices.
- Integrating SeaMAX into Your Project
  Covers methods for incorporating the versatile SeaMAX API into your project's language.
- Example Code & Instructions
  Includes several task based examples helping you quickly develop applications using SeaMAX.

The 'Modules' tab above lists all SeaMAX modules and their functions. For additional information regarding legacy products and technical specifications, please refer to the 'Related Pages' tab above.

1.4 Quickstart for Modbus Users

SeaI/O and SeaDAC modules are designed to be Modbus RTU / TCP compatible.

- Modbus Function by Sealevel Model Number
  Provides a quick reference of applicable Modbus functions based on Sealevel model number.

1.5 Warranty

Sealevel Systems, Incorporated makes no implied or express warranty of any kind with regard to this software or the documentation that is included. Sealevel Systems, Incorporated will not be liable for damages of any type resulting from the performance, use or furnishing of any of the supplied software. Any liability of Sealevel Systems, Incorporated will be limited to a refund of the purchase price or product replacement.

1.6 Copyright

This software is protected by United States copyright law, and international treaty provisions. User acknowledges that no title to the intellectual property in the software is transferred to user. User further acknowledges that title and full ownership rights to the software will remain the exclusive property of Sealevel Systems, Incorporated, and user will not acquire any ownership rights to the software. User agrees that any copies of the software will contain the same proprietary notices, which appear on and in the software.

1.7 Questions & Comments

Send your questions and comments to Sealevel Systems. For technical assistance, please include your model or part number and any device settings that may apply. Technical support is available Monday to Friday from 8:00AM to 5:00PM (US Eastern Time Zone, UTC-5 hours) by email (support@sealevel.com) or by phone at +1 (864) 843.4343.

Chapter 2

SeaMAX Change Log

2.1 Version 3.4

2.1.1 Version 3.4.0

08.20.15 (0) - Added support for eIO 160. Added SM_SetPolarity() feature. Fixed MaxSSD memory leak on input and output tabs.
02.11.16 (1) - Added support for 370. Added support for SeaIO 580. Ethernet discovery API rework. Updated timing information.

2.2 Version 3.3

2.2.1 Version 3.3.6

05.15.15 (0) - SeaRAQ 6513 / 9243 rev B modifications. NOTE: This includes renaming of the INLINE_RESISTOR_X channel range enumeration value.

2.2.2 Version 3.3.5

02.08.15 (0) - SeaRAQ 6513 / 9243 support added.
02.10.15 (1) - Update SeaMAX and Modbus documentation for SeaRAQ: 6513, 6525, and 8512.
02.23.15 (2) - Fixed version build increment. Adding MaxSSD support for 6520/6525. Fixed MaxSSD bug with 8512.

2.2.3 Version 3.3.4

07.23.14 (0) - Implemented C-Style errors, including SM_GetLastError(), SM_GetLastWin32Error(), and SM_GetLast↩FTDIError() methods.
07.25.14 (1) - Updated SeaMAX and Modbus documentation for SeaRAQ 6520.
07.29.14 (2) - Reworked AnalogConfig structure and supporting enum.
07.30.14 (3) - Fixed logic bug in Get/Set AnalogConfig() methods.

2.2.4 Version 3.3.3

01.17.14 (0) - Release candidate.
02.17.14 (1) - Updated FTDI driver to 2.08.30 for Windows 8 and 8.1 support.
03.05.14 (2) - Added support for 8511, 8521, and 9188.
03.14.14 (3) - Fixed issue with SDL_GetSerial() in .NET.
04.23.14 (4) - Fixed issue with error handling for PIO SeaDAC Lite devices.
04.28.14 (5) - Updated FTDI driver to 2.10.00.
05.20.14 (6) - Updated documentation.
05.21.14 (7) - SeaIO Ethernet Config now automatically requests Administrator privileges.
07.03.14 (8) - Updated documentation for Analog Input Range methods.

2.2.5 Version 3.3.2

02.13.13 (0) - Release candidate.
02.26.13 (1) - Fixed x64 native build.
05.10.13 (2) - Added support for baud rates up to 921600.
05.22.13 (3) - Ethernet Discovery now uses global broadcasts, eliminating some issues with device discovery.
06.07.13 (4) - Added a slight delay in Ethernet Discovery for XP compatibility.
08.13.13 (5) - Removed configuration options for the eI/O 170 analog input tab in MaxSSD which are not applicable.
08.21.13 (6) - Changed references to SeaIO to Module in MaxSSD and Ethernet Config. Added additional documentation for 462/463 units.
09.04.13 (7) - Added missing .NET hooks for SM_GlobalCommsReset() and baud rates up to 921600.
10.11.13 (8) - Redesigned MaxSSD AtD code to support 6510, 6511, 6512, 9181, 9187, 9189.
10.11.13 (9) - Fixed an issue where Modbus errors would not return the proper error code in some cases.
10.24.13 (10) - Added SM_GetAnalogConfig(), SM_SetAnalogConfig(), AnalogConfig struct, and Analog Channel Configuration Flags.
10.24.13 (11) - Added AES Encryption functionality, including SM_OpenSecure().
11.01.13 (12) - Added SME_ConfigureDeviceSecurity().
11.06.13 (13) - Added SME_RemoveDeviceSecurity().
11.12.13 (14) - Added missing .NET hooks for SDL_GetSerial().

2.2.6 Version 3.3.1

08.04.11 (0) - Release candidate.
09.15.11 (1) - Fixed buffer overflow error in SM_CustomMessage().
05.25.12 (2) - Removed erroneous configuration options from the 8126 in MaxSSD.
06.22.12 (3) - Added additional documentation for the 570.
06.22.12 (4) - Fixed a logic error that would prevent Modbus exceptions from being properly reported.
10.31.12 (5) - Changed recovery instruction text in SeaMAX Ethernet Configuration.
02.07.13 (6) - Added support for the eI/O 170 module.
02.12.13 (7) - Fixed issue with firmware update in MaxSSD.
02.13.13 (8) - Updated documentation.
02.13.13 (9) - Removed ftd2xx.dll from Windows directory.

2.2.7 Version 3.3.0

03.22.11 (0) - RC for eI/O.
04.12.11 (1) - Rolled back changes to Ethernet discovery due to broadcast problems under XP.
04.20.11 (2) - Ethernet Configuration Tool device recovery now properly recovers both eI/O and SeaI/O devices under XP and 7.
04.20.11 (3) - Updated FTDI Drivers to version 2.8.14.1.
05.26.11 (4) - Fixed a bug in MaxSSD concerning AtD Voltage Ranges on 470 devices.
09.27.11 (4b) - Re-release of 3.3.0.4. Contains recompile of drivers.
06.23.11 (5) - Fixed a bug in MaxSSD which caused outputs to be improperly masked on SeaDAC Lite devices under certain conditions.
06.24.11 (6) - Added SM_CustomMessage().
08.04.11 (7) - Added SDL and SME .NET Documentation.

2.3 Version 3.2

2.3.1 Version 3.2.4

09.22.10 (0) - Minor Release.
09.22.10 (0) - Fixed issues with 16-bit A/D conversion.
09.22.10 (0) - Added DeviceConfig struct, SM_GetDeviceConfig(), and deprecated SM_GetConfig().
10.06.10 (1) - Increased default serial port timeouts.
11.04.10 (2) - Updated FTDI Drivers.
12.07.10 (3) - Updated SeaMAX.def file to allow SM_GetDeviceConfig() to be used from managed code.
01.31.11 (4) - Added SM_COMTYPE_P_O_ETH (Power over Ethernet) and SM_COMTYPE_P_O_USB (Power over USB) Enumerations.
03.09.11 (5) - Added support for non A, B, C type subnet masks. (i.e. 255.255.255.252)
03.18.11 (6) - Fixed a bug which prevented devices on multiple network interfaces from being discovered. *03.18.11* (6) - Fixed a bug which was causing MaxSSD to put 470 units in an invalid analog input state.

2.3.2 Version 3.2.3

08.30.10 (3) - Minor Release.
08.30.10 (3) - Added Floating and 15-volt AtD capability.
08.30.10 (4) - Fixed several issues with 16-bit AtD.

2.3.3 Version 3.2.1

11.11.09 (15) - Minor Release.
11.11.09 (15) - Added dll version number and company info.

2.3.4 Version 3.2.0.14

05.14.09 (13) - Minor Release.
05.15.09 (13) - Removed 10 msec interpoll delay when communicating over Ethernet.
05.15.09 (13) - Changed BitBang reads to use getBitMode method.

05.15.09 (14) - Removed missed CAN API from documentation.

2.3.5 Version 3.2.0.12

02.09.09 (1) - Release Candidate.
02.02.09 (9) - Added 2191 functionality: USB to CAN. See Documentation.
02.16.09 (10) - Added DualCanOpen API call.
02.16.09 (12) - Removed CAN API from documentation.

2.3.6 Version 3.1.0.4

01.02.09 (1) - BETA Release.
01.09.09 (2) - Added new device functionality: USB to CAN. See Documentation.
01.11.09 (3) - Added dual USB access functionality for ECE project 9107.
01.21.09 (4) - fixed dot net dll access issue.

2.3.7 Version 3.0.7

11.26.08 (1) - BETA Release.
11.26.08 (2) - Added functionality: SM_GlobalCommsReset. See Documentation.
11.26.08 (12) - Use SM_WriteHoldingRegisters for Address ID/Baudrate range change starting at address 300.

2.3.8 Version 3.0.6

04.18.07 (1) - BETA Release.
04.18.07 (2) - Added functionality: SDL_GetSerial and SDL_GetNameBySerial. See Documentation.
06.06.08 (12) - Added Item Number 8123 to SeaMAX API, Updated Documentation.
11.19.08 (13) - Full release.

2.3.9 Version 3.0.5

08.23.07 (1) - Public release.
09.06.07 (15) - Fixed bug associated with null comm type, baudrate, or parity in SM_GetConfig.

2.3.10 Version 3.0.4

05.21.07 (1) - Released for internal testing. Includes support for wireless SeaI/O and 8126 SeaDAC Lite.
05.23.07 (34) - Fixed bug which reset the 8126 I/O direction on open.
05.29.07 (43) - Initializes the 8126 line direction on open - update to build 34 fix.
06.01.07 (67) - Updated library to support multi-platform builds.
06.11.07 (88) - Updated the device address of the I2C interface chips of the 8126.
06.15.07 (96) - SM_ConfigureSerialConnection now internally closes and re-opens the serial connection on error.
06.19.07 (102) - Fixed bug assoicated with ascii interpretation of hexidecimal keys for wireless enabled devices.
06.22.07 (138) - Overhauled the wireless interface for easier use.

06.22.07 (148) - Updated deprecated CEthernet interface to support wireless module types.
06.26.07 (356) - Provided library threading support. Implemented NotifyOnInputChange using the new thread support.
06.27.07 (387) - Modbus CRC check now excludes false Modbus messages. Note: Changed the 'Invalid Device Configuration' return value for SM_SetADDAConfig().
07.02.07 (409) - Patched memory allocation for deprecated CEthernet API (caused fatal crash on some systems).
07.12.07 (423) - Inverted the directional GPIO lines for the 8126 PIO.
07.16.07 (430) - SeaMAX has grown and several function names no longer apply. Updated (and deprecated) several function names for clarity.
07.23.07 (440) - Corrected false error code when using wireless (no encryption) and yet supplying a non-null password.
07.25.07 (450) - Fixed incorrect register addressing bug associated with Modbus SetPIOPresets.
07.30.07 (468) - Deprecated SM_Get/SetADDAConfig to two seperate functions in order to support upcoming lines.
07.31.07 (522) - Deprecated additional analog related functions and replaced with more standardized names.
08.02.07 (595) - Updated the internal Modbus class for thread saftey.
08.14.07 (639) - Added SM_ConfigureSerialTimeouts to provide for high latency serial connections.
08.14.07 (645) - Added SME_Ping to determine whether an Ethernet module is alive.
08.21.07 (611) - Rebuilt for support with 3rd party driver API (FT).

2.3.11 Version 3.0.3

02.26.07 (1) - Added basic support for the SeaDAC Lite product line.
02.27.07 (38) - Removed extraneous exports from the release library.
02.27.07 (43) - Added extractor for SeaDAC Lite device ID.
03.15.07 (61) - Fixed bug in SM_SetADDAConfig which forced some channels to be ZERO_TO_FIVE.
03.20.07 (76) - Added check for null parameters in SM_GetConfig.
03.30.07 (92) - Fixed bug associated with SM_GetPIOPresets that reversed byte orders.
04.03.07 (96) - Included SME_GetNetworkConfigBytes() into SeaMAX.h.
04.03.07 (98) - Changed several function signatures from char* to unsigned char*.
04.03.07 (110) - Removed the slaveID parameter for SM_Open() to accommodate future products.
04.03.07 (112) - Fixed bug in SM_ReadCoils and SM_ReadDiscreteInputs when using SeaDAC Lite modules.
04.04.07 (123) - Increased the SeaDAC Lite USB baudrate divisor to 921600.
04.05.07 (133) - Fixed bug associated with accessing the current module in SME after calling SME_SetNetworkConfig.
04.11.07 (165) - Added SME_GetNetworkSerialParams and SME_SetNetworkSerialParams.
04.11.07 (165) - Fixed SDL_GetModel - now returns an decimal model number (rather than hex).
04.14.07 (186) - Public release.
04.16.07 (189) - Added support for the SeaDAC Lite models 8113, 8114, & 8115.
04.24.07 (244) - Implemented I2C software interface for USB products.
04.25.07 (301) - Refactored the internal direct mode, bit bang, and I2C connection types.
04.27.07 (412) - Added support for the SeaDAC Lite 8126 programmable IO device.
04.27.07 (418) - Updated the communications type structure for compatibility with legacy code (SM_COMTYPE_RS232).
04.30.07 (429) - Addes SME network-to-serial parameter functions as well as module by MAC address.
05.01.07 (440) - Fixed bug associated with SeaI/O discovery across multiple interfaces.
05.04.07 (540) - Patched the Modbus response handler to retry up to two times, due to a long-standing uC issue.
05.15.07 (648) - Fixed a USB location ID problem related to dual port SeaDAC Lite USB chipsets.
05.21.07 (668) - Fixed bug associated with repetitive ARP spoofing Ethernet SeaI/O modules.

2.3.12 Version 3.0.2

02.08.07 (1) - Released for internal beta testing.
02.13.07 (2) - Added support for SeaMAX Version 3 handles from the SeaMAX Version 2 deprecated interface.
02.15.07 (25) - Added support as 64-bit library API release.
02.15.07 (41) - Fixed VS2005 bug associated with MSVCRT80.DLL by linking in static library.

02.16.07 (46) - Return handle by parameter in SME_Initialize.
02.16.07 (46) - Changed the SME_GetName parameters to remove the size constraint.
02.22.07 (96) - Refactored UDP broadcast timeout associated with out-of-subnet XPorts.

2.3.13 Version 3.0.1

12.14.06 (5) - Initial write of SeaMAX version 3.0.1.
12.18.06 (112) - Incorporated deprecated CEthernet functionality into SeaMAX.
12.20.06 (115) - Internal Alpha Release for testing.
01.07.07 (203) - Preliminary deprecated SeaMAX v2 interface created.
01.16.07 (302) - Included throttle delay for ModbusPoll of zero bytes.
01.17.07 (309) - Writing a single register or coil now calles the single vs. multiple Modbus cmd.
01.25.07 (314) - Included SM_SetSerialCommunications to configure the local serial port.
01.26.07 (316) - Fixed register addressing bug associated with setting the PIO presets.
01.30.07 (332) - Identified longstanding Ethernet ModbusTCP bug in SeaIO bridge firmware. Patched in SeaMAX for legacy customers.
02.06.07 (407) - Changed SM_GetADDAConfig(), SM_SetADDAConfig() to use bytes instead of bit values for language compatibility.
02.06.07 (411) - Changed SM_SetPIODirection(), SM_GetPIODirection() to be byte, rather than bit, oriented.

Chapter 3

Example Code & Instructions

3.1 Introduction

SeaMAX features an easy-to-use functional interface with each function returning a single integer indicating success (zero or greater) or failure (less than zero). Any data sent to or received from the functions are handled as pass by reference/value parameters.

Note

Some SeaMAX example projects have been included in the default installation to assist in quick development. View the **Examples** folder in your SeaMAX installation directory for full project examples.

Before jumping head first into SeaMAX code examples, it may be useful to first understand how to incorporate SeaMAX into your code project. To learn more about including SeaMAX into your project, read Integrating SeaMAX into Your Project.

Actual example code may be found in your SeaMAX installation directory, "C:\\Program Files\\Sealevel Systems\\SeaMAX\\Example Projects\\". Refer to those samples for additional details and illustrations on the use of SeaMAX.

Chapter 4

Integrating SeaMAX into Your Project

4.1 Introduction

SeaMAX has been created with careful consideration of common programming languages and language interoperability. Language support is bounded by four datatypes: Integers (32 or 64-bit, system dependent), double precision floating points, byte arrays (or strings), and pointers (system dependent). If your project uses a language or compiler not listed below, yet supports these four datatypes and allows access to DLL functions, then your languages is most likely compatible with SeaMAX.

4.1.1 Language Support

Language support is bound by three datatypes: 4-byte integers (or longs), double precision floating points, and byte arrays (or strings). If your project uses a language or compiler not covered below, yet supports the datatypes listed previously and allows access to DLL functions, then your language should be compatible with SeaMAX.

4.1.2 Language and Compiler Examples

The list of languages and compilers listed below is not all-inclusive. It is a sample of the most requested languages and should not be considered complete.

- MS Visual C++ 6.0
- MS Visual Basic 6.0
- Borland Delphi 7
- Microsoft .NET Languages (C#, J#, VB .NET, C++ .NET)

4.2 DLL Errors and Exceptions

When starting a new project, you must copy all of the SeaMAX related DLLs (C:\Program Files\Sealevel Systems\Sea↩ MAX\Dll) into your project's executable folder before executing your binary. **If you are receiving 'DLL Not Found' exceptions or are receiving warnings, there is a good chance you have not copied the DLLs to the correct folder of your project.** DLLs should be located in the same folder as your project's compiled executable. The required DLLs include ftd2xx.dll, SeaMAX.dll, and if you are using the managed interface, SeaMAX Dot Net.dll.

4.3 MS Visual C++ 6.0

To use SeaMAX in an existing Visual C++ 6 project initially requires three steps. First, the Vistual Studio 6.0 include and library directorys must be updated to point to the install path for SeaMAX, which (assuming a default installation) can by accomplished by the following:

1. Choose 'Tools->Options...' from the menu bar in Visual Studio 6.0
2. Select the 'Directories' tab and choose 'Include Files' from the 'Show Directories for' drop-down list.
3. Add a new entry that is the directory "C:\Program Files\Sealevel Systems\SeaMAX\Include"
4. Add a new entry to the 'Library Files' listing that is the directory "C:\Program Files\Sealevel Systems\SeaMAX\Lib"

Second, add the following include pre-processor definition to the beginning of your application source:

```
#include <SeaMAX.h>
```

Finally, add the SeaMAX library file to your project by completing the following:

1. Choose 'Project->Settings' from the menu bar.
2. Select the 'Link' tab.
3. At the end of the 'Object/library modules' listing, append the following text: "SeaMAX.lib"
4. Select the 'Debug' tab.
5. Enter 'C:\Program Files\Sealevel Systems\SeaMAX\DLL' as your working directory.
   This will allow your program to see the SeaMAX DLL at runtime.

Your project should now be ready to begin using SeaMAX!

4.4 MS Visual Basic 6.0

In Visual Basic 6.0, the SeaMAX dll can be declared function-by-function in a separate module. It is not necessary to declare all of SeaMAX in your project, only the functions you need to use.

Before inserting the DLL imports into a new module, it is best to copy the required DLLs to your project directory. The required DLLs will be located in your SeaMAX installation directory under the DLL folder. All DLLs located in this folder must be copied to your VB project folder for your executable to correctly execute.

After copying the DLLs to your project folder, create a module in your VB 6 project. Right-click on the 'modules' folder in your project tree pane. Select the 'Add->Module' option. Click the 'Open' button to insert a blank module, and paste any or all of the declarations below into the new module.

Note

Visual Basic 6.0 handles strings differently from a C-style DLL. In order to use any SeaMAX function which returns a string (such as SME_GetName() or SME_GetNetworkConfig()), the VB string passed to the DLL must already have space allocated to it - the DLL will not internally allocate memory to store it's value. An example of how to do this is below:

```
Dim handle As Long

' Any string passed to a C-type DLL must have space pre-allocated
' which is accomplished by the * 30 or * 64, indicating that the
' string should have 30 or 64 characters

Dim module As String * 30
Dim ip As String * 64
Dim mac As String * 64

' Initialize SeaMAX

handle = SME_Initialize()

' Search for the modules and iterate through the list

For i = 1 To SME_SearchForModules(handle)

  ' Get all of the module details such as name, ip, and MAC address

  SME_GetName handle, module, 64
  SME_GetNetworkConfig handle, ip, 0, 0
  SME_GetMACAddress handle, mac

  ' Output the strings
  MsgBox CtoVBStr(module) & " at " & CtoVBStr(ip) & " [" & CtoVBStr(mac) & "]"

  SME_NextModule handle

Next i

' Release the memory allocated by the API

SME_Cleanup (handle)
```

Since the strings in the above example have to be pre-allocated, VB 6 does not recognize the C-style string ending character of NULL, and therefore has problems displaying the strings properly. To correct this problem, the function CtoVBStr() has been provided below:

```
Public Function CtoVBStr(ByVal cString As String) As String
  CtoVBStr = Left(cString, InStr(cString, Chr(0)) - 1)
End Function
```

4.5 Borland Delphi 7

The following code provides one example for importing a SeaMAX function into a Delphi 7 project. There may be other more applicable methods depending on your project. This project uses default and has a single button placed on the main form. The following code is the function created for the button's event handler:

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    handle: Thandle;
    maj: Integer;
    min: Integer;
    rev: Integer;
    bui: Integer;
    SM_GetVersion: procedure (Major:Pointer; Minor:Pointer; Revision:Pointer; Build:Pointer); stdcall;

begin
  handle := LoadLibrary('SeaMAX.dll');
  if handle >= 32 then { success }
  begin
```

```
    SM_GetVersion := GetProcAddress(handle, 'SM_Version');

    SM_GetVersion(@maj, @min, @rev, @bui);
    MessageDlg('Using SeaMAX Version ' + IntToStr(maj) + '.' + IntToStr(min) + '.' + IntToStr(rev)
          + '.' + IntToStr(bui), mtError, [mbOk], 0)
  end
else
  MessageDlg('Could not open the SeaMAX dll.', mtError, [mbOk], 0)
end;
```

4.6 Microsoft .NET Languages (C#, J#, VB .NET, C++ .NET)

For 32-bit .NET applications, SeaMAX includes a CLR component which encapsulates the SeaMAX API across any .NET project.

To include the .NET component into your project, open your Microsoft Visual Studio project and select Project on the menu bar. Select the 'Add Reference...' menu item, click the Browse tab. Browse to the SeaMAX installation folder ('C:\Program Files\Sealevel Systems\SeaMAX' by default) and choose the 'SeaMAX dot Net.dll' located in the 'Dll' folder.

Next, right click on the project and select Add->Existing Item. Select both the SeaMAX.dll and ftd2xx.dll from the 'Dll' folder. Select each of these files and in the Properties of the file set the Build Action property to either "Copy Always" or "Copy if Newer." Visual Studio will now automatically copy these necessary files to the output directory when you build your project.

The managed SeaMAX .NET component encapsulates SeaMAX into the Sealevel namespace. The functions provided in the .NET SeaMAX class conform to the SeaMAX API documented here, with the exception that the SeaMAX handle is concealed as a private variable within the class and is automatically created and destroyed for you.

The following is an example of how to call SM_Open() and SM_Close() within a C# .NET project after adding the Managed SeaMAX reference:

```
Sealevel.SeaMAX sm = new Sealevel.SeaMAX();

if (sm.SM_Open("COM1") < 0)
{
   // Error opening COM1
}

if (sm.IsSeaMAXOpen)
{
   sm.SM_Close();
}
```

New in version 1.0.3.0: .NET XML style documentation is now available. You may need to remove, then add the SeaMAX Dot Net.dll reference, shown above, in order for the documentation to be available inside Visual Studio. An example of the documentation is shown below.

Chapter 5

Legacy and Technical Documentation

The following documentation has been provided for legacy applications and diagnostics. If you are experiencing technical difficulties, please contact Sealevel technical support at support@sealevel.com or call +1 (864) 843.4343 Monday to Friday from 8:00 AM to 5:00 PM (United States Eastern Time).

5.1 Manual Conversion of A/D Values

To properly use the values returned from Analog Input channels, the application must convert the returned 16-bit value to a voltage. The function SM_AtoDConversion() has been provided to assist the programmer and eliminate error. At times, it may be desirable to manually convert the 16-bit values to voltages. The following is a brief tutorial on how to do so.

5.1.1 Analog to Digital Conversion in Positive Voltage Ranges

When an analog input has been configured to convert positive voltage ranges [0, Maximum VDC], the 16-bit values read from the device will correspond to the range 0 to MAX ADC ((2^ADC Resolution) - 1). The formula for converting the returned values to voltage is shown below:

```
{voltage} = ({value} / {2^ADC_RES - 1}) * {MAX_VDC}
```

For example, a returned value of 1319 (0x527) on a SeaI/O 470 module (12-bit ADC) configured for a 0-10V range equals a voltage input of 3.22V.

```
{voltage} = (1319 / 4095) * (10 V)

{voltage} = 3.22 V
```

5.1.2 Differential Voltage Range

When an analog input has been configured to convert bi-polar voltage ranges [-Maximum VDC, Maximum VDC], the most significant ADC bit (dependent on ADC resolution) determines whether the returned values is either positive (zero) or negative (one). The remaining bits provide the magnitude of the value. In other words, if the returned value is less than or equal to half the Maxiumum ADC value (i.e. 2^(ADC Resolution - 1)), then the resulting voltage is positive, otherwise it is negative. To calculate the voltage, continue the same as described above with 1 less bit of ADC resolution.

```
If the {value} is less than or equal to {MAX_ADC / 2},

{voltage} = {value} / {{2^ADC_RES} / 2} * {MAX_VDC}

Else

{voltage} = ({value} - {2^ADC_RES}) / {2^ADC_RES / 2} * {MAX_VDC}
```

For example, a returned value of 659 (0x293) on a SeaI/O 470 with a voltage range of +/-10V is less than 2048, so the calculated voltage is positive. Using the first formula above, the returned value results in a calculated voltage of 3.22V.

```
{voltage} = 659 / 2048 * 10V

{voltage} = 3.22V
```

If the polarity at the input is reversed, the returned value from SM_ReadInputRegisters() is 3434 (0xD6A). Since the returned value is greater than 2048, the calculated voltage must be negative and therefore the second formula is used.

```
{voltage} = (3434 - 4096) / 2048 * 10V

{voltage} = -3.23V
```

5.1.3 Current Loop Mode

In current loop mode, the returned value is the voltage drop measured across a built-in precision resistor. To calculate the current first use the appropriate formula above to determine the voltage, then use Ohm's law (I = V/R) to get the current by dividing by the resistor value. For example, if we use the 3.23V from the SeaI/O 470 shown above, the current would be calculated as follows:

```
{current} = {voltage} / {resistance}

{current} = 3.22V / 249 Ohms

{current} = 0.01293 A = 12.93 mA
```

5.2 SeaIO 462 / 463 Holding Register Set

The following register set for the 46x series modules has been provided for legacy reference only. To set either the PIO presets or direction, use SM_SetPIOPresets() or SM_SetPIODirection().

See SeaI/O 462 or SeaI/O 463 for more details.

SeaI/O 462 & 463 Holding Register Set

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Register 0	Bridge Type								Model Number							
Register 1	Baud Rate				Parity				Address							
Register 2 IO Direction	0	0	Port 4C	Port 4B	Port 4A	Port 3C	Port 3B	Port 3A	0	0	Port 2C	Port 2B	Port 2A	Port 1C	Port 1B	Port 1A
Register 3 Presets Port B1 and A1	B1 Bit 7	B1 Bit 6	B1 Bit 5	B1 Bit 4	B1 Bit 3	B1 Bit 2	B1 Bit 1	B1 Bit 0	A1 Bit 7	A1 Bit 6	A1 Bit 5	A1 Bit 4	A1 Bit 3	A1 Bit 2	A1 Bit 1	A1 Bit 0
Register 4 Presets Port A2 and C1	A2 Bit 7	A2 Bit 6	A2 Bit 5	A2 Bit 4	A2 Bit 3	A2 Bit 2	A2 Bit 1	A2 Bit 0	C1 Bit 7	C1 Bit 6	C1 Bit 5	C1 Bit 4	C1 Bit 3	C1 Bit 2	C1 Bit 1	C1 Bit 0
Register 5 Presets Port C2 and B2	C2 Bit 7	C2 Bit 6	C2 Bit 5	C2 Bit 4	C2 Bit 3	C2 Bit 2	C2 Bit 1	C2 Bit 0	B2 Bit 7	B2 Bit 6	B2 Bit 5	B2 Bit 4	B2 Bit 3	B2 Bit 2	B2 Bit 1	B2 Bit 0
Register 6 Presets Port B3 and A3	B3 Bit 7	B3 Bit 6	B3 Bit 5	B3 Bit 4	B3 Bit 3	B3 Bit 2	B3 Bit 1	B3 Bit 0	A3 Bit 7	A3 Bit 6	A3 Bit 5	A3 Bit 4	A3 Bit 3	A3 Bit 2	A3 Bit 1	A3 Bit 0
Register 7 Presets Port A4 and C3	A4 Bit 7	A4 Bit 6	A4 Bit 5	A4 Bit 4	A4 Bit 3	A4 Bit 2	A4 Bit 1	A4 Bit 0	C3 Bit 7	C3 Bit 6	C3 Bit 5	C3 Bit 4	C3 Bit 3	C3 Bit 2	C3 Bit 1	C3 Bit 0
Register 8 Presets Port C4 and B4	C4 Bit 7	C4 Bit 6	C4 Bit 5	C4 Bit 4	C4 Bit 3	C4 Bit 2	C4 Bit 1	C4 Bit 0	B4 Bit 7	B4 Bit 6	B4 Bit 5	B4 Bit 4	B4 Bit 3	B4 Bit 2	B4 Bit 1	B4 Bit 0

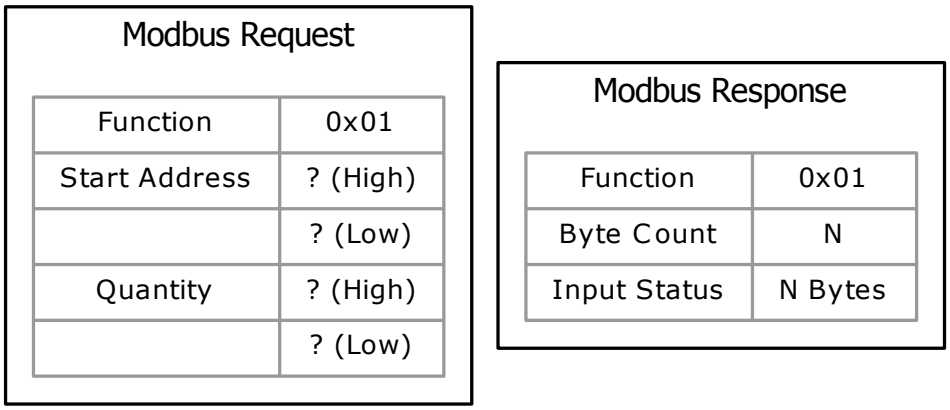
5.3 Communicating Via Modbus

The Sealevel SeaI/O series of digital I/O communicates serially using the Modbus protocol (http://www.modbus.org).

Enumerated below are the standard Modbus functions supported by the SeaI/O series of digital I/O. Following those are the Sealevel SeaI/O specific custom functions used for configuring and manipulating specific SeaI/O modules.

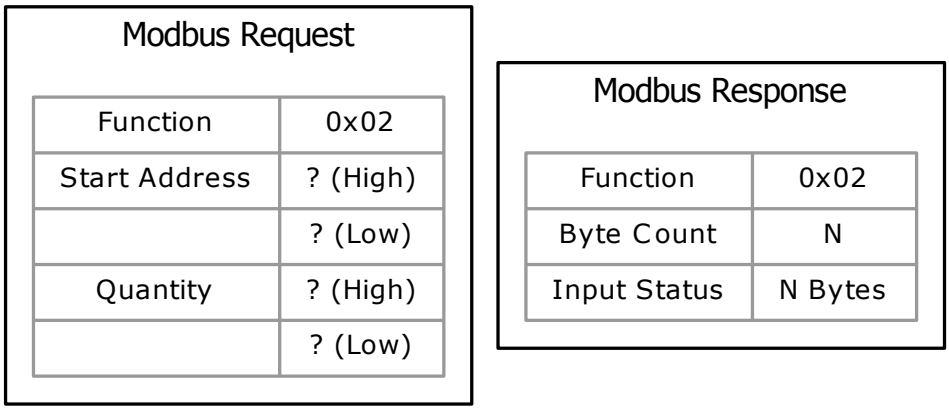
5.3.1 Standard Modbus Functions

5.3.1.1 (0x01) Read Coils



'Read Coils' Modbus PDU (0x01)

5.3.1.2 (0x02) Read Discrete Inputs



'Read Discrete Inputs' Modbus PDU (0x02)

5.3.1.3 (0x03) Read Holding Registers

Modbus Request

Function	0x03
Start Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)

Modbus Response

Function	0x03
Byte Count	N
Input Status	2*N Bytes

'Read Holding Registers' Modbus PDU (0x03)

5.3.1.4 (0x04) Read Input Registers

Modbus Request

Function	0x04
Start Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)

Modbus Response

Function	0x04
Byte Count	N
Input Status	2*N Bytes

'Read Input Registers' Modbus PDU (0x04)

5.3.1.5 (0x05) Write Single Coil

Modbus Request	
Function	0x05
Output Address	? (High)
	? (Low)
Output Value	0x00 or 0xFF (High)
	0x00 (Low)

Modbus Response	
Function	0x05
Output Address	? (High)
	? (Low)
Output Value	0x00 or 0xFF (High)
	0x00 (Low)

'Write Single Coil' Modbus PDU (0x05)

5.3.1.6 (0x06) Write Single Register

Modbus Request	
Function	0x06
Register Address	? (High)
	? (Low)
Register Value	? (High)
	? (Low)

Modbus Response	
Function	0x06
Register Address	? (High)
	? (Low)
Register Value	? (High)
	? (Low)

'Write Single Register' Modbus PDU (0x06)

5.3.1.7 (0x0F) Write Multiple Coils

Modbus Request

Function	0x0F
Starting Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)
Byte Count	N
Outputs Values	N Bytes

Modbus Response

Function	0x0F
Starting Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)

'Write Multiple Coils' Modbus PDU (0x0F)

5.3.1.8 (0x10) Write Multiple Registers

Modbus Request

Function	0x10
Starting Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)
Byte Count	N
Registers Values	2*N Bytes

Modbus Response

Function	0x10
Starting Address	? (High)
	? (Low)
Quantity	? (High)
	? (Low)

'Write Multiple Registers' Modbus PDU (0x10)

5.3.2 Sealevel Seal/O Specific Modbus Functions

5.3.2.1 (0x41) Read Programmable IO

Modbus Request

Function	0x41

Modbus Response

Function	0x41
Model	Model Low Byte
Config (Banks 6-11)	?
Config (Banks 0-5)	?
Bank 0	?
Bank 1	?
Bank 2	?
Bank 3	?
Bank 4	?
Bank 5	?
Bank 6	?
Bank 7	?
Bank 8	?
Bank 9	?
Bank 10	?
Bank 11	?

'Read Programmable IO' Modbus PDU (0x41)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Config (Banks 6-11) Byte	x	x	Bank 11	Bank 10	Bank 9	Bank 8	Bank 7	Bank 6
Config (Banks 0-5) Byte	x	x	Bank 5	Bank 4	Bank 3	Bank 2	Bank 1	Bank 0

Parameter 'config' Layout of IO Direction

Legend
'0' = Bank of 8 Outputs
'1' = Bank of 8 Inputs

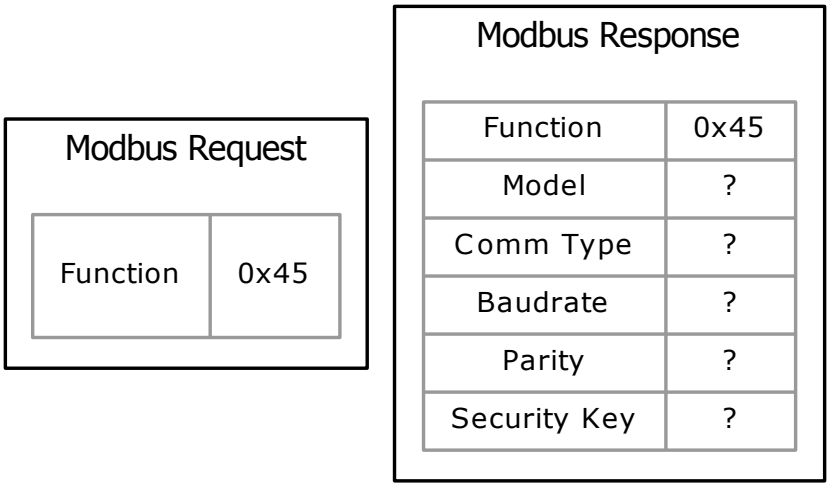
5.3.2.2 (0x42) Write Programmable IO

Modbus Request	
Function	0x42
Bank 0	?
Bank 1	?
Bank 2	?
Bank 3	?
Bank 4	?
Bank 5	?
Bank 6	?
Bank 7	?
Bank 8	?
Bank 9	?
Bank 10	?
Bank 11	?

Modbus Response	
Function	0x42
Bank 0	?
Bank 1	?
Bank 2	?
Bank 3	?
Bank 4	?
Bank 5	?
Bank 6	?
Bank 7	?
Bank 8	?
Bank 9	?
Bank 10	?
Bank 11	?

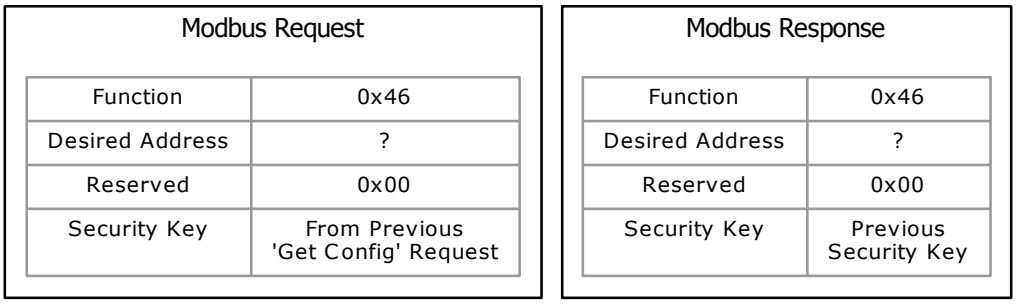
'Write Programmable IO' Modbus PDU (0x42)

5.3.2.3 (0x45) Get Module Configuration



'Get Module Configuration' Modbus PDU (0x45)

5.3.2.4 (0x46) Set Software Slave ID



'Set Software Slave ID' Modbus PDU (0x46)

5.3.2.5 (0x47) Set Communications

Modbus Request

Function	0x47
Baudrate	?
Parity	?
Security Key	From Previous 'Get Config' Request

Modbus Response

Function	0x47
Baudrate	?
Parity	?
Security Key	Previous Security Key

'Set Communications' Modbus PDU (0x47)

5.3.2.6 (0x64) Set A/D & D/A Configuration

Modbus Request

Function	0x64
Device Config	?
Channels 1-4	?
Channels 5-8	?
Channels 9-12	?
Channels 13-16	?

Modbus Response

Function	0x64
Error Code If Any	?

'Set A/D, D/A Configuration' Modbus PDU (0x64)

Device Config Bit Packing

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Config Byte	A/D Voltage Reference				A/D Channel Mode			

Channels Config Bit Packing

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Channels 1-4 Byte	Ch 1 Range		Ch 2 Range		Ch 3 Range		Ch 4 Range	
Channels 5-8 Byte	Ch 5 Range		Ch 6 Range		Ch 7 Range		Ch 8 Range	
Channels 9-12 Byte	Ch 9 Range		Ch 10 Range		Ch 11 Range		Ch 12 Range	
Channels 13-16 Byte	Ch 13 Range		Ch 14 Range		Ch 15 Range		Ch 16 Range	

Parameter 'config' Layout of IO Direction

5.3.2.7 (0x65) Get A/D & D/A Configuration

Modbus Request

Function	0x65

Modbus Response

Function	0x65
Device Config	?
Channels 1-4	?
Channels 5-8	?
Channels 9-12	?
Channels 13-16	?

'Get A/D, D/A Configuration' Modbus PDU (0x65)

Device Config Bit Packing

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Config Byte	A/D Voltage Reference				A/D Channel Mode			

Channels Config Bit Packing

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Channels 1-4 Byte	Ch 1 Range		Ch 2 Range		Ch 3 Range		Ch 4 Range	
Channels 5-8 Byte	Ch 5 Range		Ch 6 Range		Ch 7 Range		Ch 8 Range	
Channels 9-12 Byte	Ch 9 Range		Ch 10 Range		Ch 11 Range		Ch 12 Range	
Channels 13-16 Byte	Ch 13 Range		Ch 14 Range		Ch 15 Range		Ch 16 Range	

Parameter 'config' Layout of IO Direction

5.3.2.8 (0x66) Get Extended Module Configuration

Modbus Request

Function	0x66

Modbus Response

Function	0x66
Model	? (High)
	? (Low)
Firmware Ver	1 Byte
Reserved Bytes	13 Bytes

'Get Extended Module Configuration' Modbus PDU (0x66)

5.4 Calculating Modbus CRCs

5.4.1 Introduction

Modbus RTU (serial) requires a 16-bit CRC calculation following every Modbus RTU payload. To calculate that 16-bit CRC, the SeaMAX uses the following code:

```
unsigned short calc_crc(int n,unsigned char *outbound_message)
{
  unsigned char carry_flag;
```

```
    unsigned short crc=0xFFFF;

    for (int i = 0; i < n; i++)
    {
      crc = crc ^ outbound_message[i];

      for (int j = 0; j < 8; j++)
      {
        carry_flag = crc & 0x01;
        crc = crc >> 1;

        if (carry_flag == 1)
        {
          crc = crc ^ 0xA001;
        }
      }
    }

    return crc;
  }
```

5.5 SeaMAX Function Timing Data

Each model of Sealevel I/O device has been tested to determine the minimum response time from a high level software point of view. The following SeaMAX functions were exectued and timed using a high-precision software timer using Visual C# .NET.

Note

The system configuration used for testing is as follows:

- System Hardware : Intel Core i5 3.0 GHz w/ 4.0 GB RAM
- Operating System : Windows 7 Professional
- Testbed Software : SeaMAX Benchmark .NET, SeaMAX Version 3.4.0.1
- Test Hardware : All Sealevel I/O modules tested with factory default conditions

The timings presented below are from a modbus viewpoint only. Switching times for hardware components may vary. See your product's hardware manual for more information.

5.5.1 Seal/O Modules

Note

The function timing data listed above represent S-series (232) SeaI/O modules, operating at 9600 bps with no parity.

5.5.2 SeaDAC Modules

Note

The function timing data listed above represent factory default SeaDAC modules, operating at 9600 bps with no parity.

5.5.3 SeaDAC Lite Modules

Note

All timing data above represents SeaDAC Lite modules with factory default settings.

5.5.4 eI/O Modules

5.6 Modbus Function by Sealevel Model Number

5.6.1 Seal/O Modules

The following list enumerates all the relavent Modbus functions for each model of Seal/O module.

- Model 410
- Model 420
- Model 430
- Model 440
- Model 450
- Model 462
- Model 463
- Model 470
- Model 520
- Model 530
- Model 540
- Model 570
- Model 580

5.6.2 SeaDAC Modules

- Model 8221
- Model 8222
- Model 8223
- Model 8224
- Model 8225
- Model 8227
- Model 8232

5.6.3 SeaDAC Lite Modules

The SeaDAC Lite modules are USB modules and do not provide a serial interface. Therefore, they do not support any Modbus functionality.

5.6.4 eI/O Modules

- Model 110
- Model 120
- Model 130
- Model 140
- Model 150
- Model 160
- Model 170

5.6.5 SeaRAQ Modules

- Model 6510
- Model 6511
- Model 6512
- Model 6513
- Model 6520
- Model 6525
- Model 8510
- Model 8511
- Model 8512
- Model 8520
- Model 8521

5.6.6 SeaCONNECT Modules

- Model 370

5.6.7 SeaIO 410

Modbus Function	**Valid Range**	***Description***
(0x02) Read Discrete Inputs	0 - 15	Reads the opto-isolated inputs.

(0x01) Read Coils	0 - 15	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple reed relay outputs.

5.6.8 SealO 420

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple form C outputs.

5.6.9 SealO 430

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 31	Reads the opto-isolated inputs.

5.6.10 SealO 440

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 31	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 31	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 31	Writes multiple reed relay outputs.

5.6.11 SealO 450

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 15	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple form C outputs.

5.6.12 SealO 462

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 95	Reads the state of TTL outputs, skipping over inputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when reading 8 outputs. (ie. 16-23)
(0x02) Read Discrete Inputs	0 - 95	Reads the TTL inputs, skipping over outputs. Uses absolute addressing. Usable with 1 or 8 inputs only. Must start at a multiple of 8 when reading 8 inputs. (ie. 16-23)
(0x05) Write Single Coil	0 - 95	Writes to a single TTL output. Uses absolute addressing.
(0x0F) Write Multiple Coils	0 - 95	Writes multiple TTL outputs, skipping over inputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when writing 8 outputs. (ie. 16-23)

(0x41) Read Programmable IO	n/a	Reads the entire 96-bit I/O space.
(0x42) Write Programmable IO	n/a	Writes the entire 96-bit I/O space, masking out inputs.
(0x03) Read Holding Registers	0 - 8	Reads the state of the SeaIO 462 / 463 Holding Register Set.
(0x06) Write Single Register	1 - 8	Configures the module based on the SeaIO 462 / 463 Holding Register Set

5.6.13 SeaIO 463

Modbus Function	**Valid Range**	***Description***
(0x01) Read Coils	0 - 95	Reads the state of TTL outputs, skipping over inputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when reading 8 outputs. (ie. 16-23)
(0x02) Read Discrete Inputs	0 - 95	Reads the TTL inputs, skipping over outputs. Uses absolute addressing. Usable with 1 or 8 inputs only. Must start at a multiple of 8 when reading 8 inputs. (ie. 16-23)
(0x05) Write Single Coil	0 - 95	Writes to a single TTL output. Uses absolute addressing.
(0x0F) Write Multiple Coils	0 - 95	Writes multiple TTL outputs, skipping over inputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when writing 8 outputs. (ie. 16-23)

(0x41) Read Programmable IO	n/a	Reads the entire 96-bit I/O space.
(0x42) Write Programmable IO	n/a	Writes the entire 96-bit I/O space, masking out inputs.
(0x03) Read Holding Registers	0 - 8	Reads the state of the SeaIO 462 / 463 Holding Register Set.

(0x06) Write Single Register	1 - 8	Configures the module based on the SeaIO 462 / 463 Holding Register Set

5.6.14 SeaIO 470

Modbus Function	**Valid Range**	***Description***
(0x02) Read Discrete Inputs	0 - 7	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of open collector outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single open collector output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple open collector outputs.
(0x04) Read Input Registers	0 - 15/7	Reads the analog single-ended/differential inputs.
(0x06) Write Single Register	0 - 1	Writes to a single analog output.
(0x10) Write Multiple Registers	0 - 1	Writes multiple analog outputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D and D/A configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D and D/A configuration.
(0x66) Get Extended Module Configuration	n/a	Gets the hardware jumper configuration.

5.6.15 SeaIO 520

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 7	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple form C outputs.

5.6.16 SealO 530

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 15	Reads the state of open collector outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single open collector output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple open collector outputs.

5.6.17 SealO 540

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 31	Reads the state of open collector outputs.
(0x05) Write Single Coil	0 - 31	Writes to a single open collector output.
(0x0F) Write Multiple Coils	0 - 31	Writes multiple open collector outputs.

5.6.18 SealO 570

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 7	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple form C outputs.
(0x04) Read Input Registers	0 - 7	Reads the analog floating/ground referenced inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D configuration.
(0x66) Get Extended Module Configuration	n/a	Gets the hardware jumper configuration.

5.6.19 SealO 580

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 31	Reads the opto-isolated inputs.

5.6.20 SeaDAC 8221

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 15	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple reed relay outputs.

5.6.21 SeaDAC 8222

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple form C outputs.

5.6.22 SeaDAC 8223

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 31	Reads the opto-isolated inputs.

5.6.23 SeaDAC 8224

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 31	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 31	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 31	Writes multiple reed relay outputs.

5.6.24 SeaDAC 8225

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 15	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple form C outputs.

5.6.25 SeaDAC 8227

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 7	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of open collector outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single open collector output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple open collector outputs.
(0x04) Read Input Registers	0 - 15/7	Reads the analog single-ended/differential inputs.
(0x06) Write Single Register	0 - 1	Writes to a single analog output.
(0x10) Write Multiple Registers	0 - 7	Writes multiple analog outputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D and D/A configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D and D/A configuration.
(0x66) Get Extended Module Configuration	n/a	Gets the hardware jumper configuration.

5.6.26 SeaDAC 8232

Modbus Function	Valid Range	*Description*

(0x02) Read Discrete Inputs	0 - 7	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 7	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple form C outputs.

5.6.27 eIO 110

Modbus Function	**Valid Range**	***Description***
(0x02) Read Discrete Inputs	0 - 3	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 3	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 3	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 3	Writes multiple reed relay outputs.

5.6.28 eIO 120

Modbus Function	**Valid Range**	***Description***
(0x02) Read Discrete Inputs	0 - 3	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 3	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 3	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 3	Writes multiple form C outputs.

5.6.29 eIO 130

Modbus Function	**Valid Range**	***Description***
(0x02) Read Discrete Inputs	0 - 3	Reads the opto-isolated inputs.

5.6.30 eIO 140

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 3	Reads the state of reed relay outputs.
(0x05) Write Single Coil	0 - 3	Writes to a single reed relay output.
(0x0F) Write Multiple Coils	0 - 3	Writes multiple reed relay outputs.

5.6.31 eIO 150

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 3	Reads the state of form C outputs.
(0x05) Write Single Coil	0 - 3	Writes to a single form C output.
(0x0F) Write Multiple Coils	0 - 3	Writes multiple form C outputs.

5.6.32 eIO 160

Modbus Function	Valid Range	*Description*
(0x41) Read Programmable IO	n/a	Reads the entire 32-bit I/O space.
(0x42) Write Programmable IO	n/a	Writes the entire 32-bit I/O space, masking out inputs.
(0x03) Read Holding Registers	0 - 8	Reads the state of the SeaIO 462 / 463 Holding Register Set.
(0x06) Write Single Register	1 - 8	Configures the module based on the SeaIO 462 / 463 Holding Register Set. Note that registers 1 and 2 of the 160 may be used to read / write the TTL IO.

5.6.33 eIO 170

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 1	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 1	Reads the state of solid-state relay outputs.
(0x05) Write Single Coil	0 - 1	Writes to a single solid-state relay output.
(0x0F) Write Multiple Coils	0 - 1	Writes multiple solid-state relay outputs.
(0x04) Read Input Registers	0 - 7/3	Reads the analog single-ended/differential inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D configuration.

5.6.34 SeaRAQ 6510

Modbus Function	Valid Range	*Description*
(0x04) Read Input Registers	0 - 7	Reads the analog inputs.

5.6.35 SeaRAQ 6511

Modbus Function	Valid Range	*Description*
(0x04) Read Input Registers	0 - 5	Reads the analog inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D configuration.

5.6.36 SeaRAQ 6512

Modbus Function	Valid Range	*Description*
(0x04) Read Input Registers	0 - 5	Reads the analog inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D configuration.

5.6.37 SeaRAQ 6513

Modbus Function	Valid Range	*Description*
(0x04) Read Input Registers	0 - 7	Reads the analog inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D configuration. • Range = ch[n] + (ch[n + 8] << 2)
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D configuration. • Range = ch[n] + (ch[n + 8] << 2)

5.6.38 SeaRAQ 6520

Modbus Function	Valid Range	*Description*
(0x06) Write Single Register	0 - 7	Writes to a single analog output.
(0x10) Write Multiple Registers	0 - 7	Writes multiple analog outputs.

5.6.39 SeaRAQ 6525

Modbus Function	Valid Range	*Description*
(0x06) Write Single Register	0 - 3	Writes to a single analog output.
(0x10) Write Multiple Registers	0 - 3	Writes multiple analog outputs.

5.6.40 SeaRAQ 8510

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the digital inputs.

5.6.41 SeaRAQ 8511

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the digital inputs.

5.6.42 SeaRAQ 8512

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 15	Reads the digital inputs.
(0x01) Read Coils	0 - 3	Reads the state of the relay outputs.
(0x05) Write Single Coil	0 - 3	Writes to a single relay output.
(0x0F) Write Multiple Coils	0 - 3	Writes multiple relay outputs.

5.6.43 SeaRAQ 8520

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 15	Reads the state of the relay outputs.
(0x05) Write Single Coil	0 - 15	Writes to a single relay output.
(0x0F) Write Multiple Coils	0 - 15	Writes multiple relay outputs.

5.6.44 SeaRAQ 8521

Modbus Function	Valid Range	*Description*
(0x01) Read Coils	0 - 7	Reads the state of the relay outputs.
(0x05) Write Single Coil	0 - 7	Writes to a single relay output.
(0x0F) Write Multiple Coils	0 - 7	Writes multiple relay outputs.

5.6.45 SeaCONNECT 370

Modbus Function	Valid Range	*Description*
(0x02) Read Discrete Inputs	0 - 3	Reads the opto-isolated inputs.
(0x01) Read Coils	0 - 1	Reads the state of open collector outputs.
(0x05) Write Single Coil	0 - 1	Writes to a single open collector output.
(0x0F) Write Multiple Coils	0 - 1	Writes multiple open collector outputs.
(0x04) Read Input Registers	0 - 1	Reads the analog single-ended/differential inputs.
(0x64) Set A/D & D/A Configuration	n/a	∗ Sets the A/D and D/A configuration.
(0x65) Get A/D & D/A Configuration	n/a	∗ Gets the A/D and D/A configuration.
(0x66) Get Extended Module Configuration	n/a	Gets the hardware jumper configuration.

Chapter 6

SeaMAX by Sealevel Model Number

6.1 Seal/O Modules

The following list enumerates all the possible functions for each model of Seal/O module. For Seal/O modules with an Ethernet or wireless Ethernet interface, see the Ethernet Discovery & Configuration API document. Modules featuring a serial interface can use the SM_ConfigureSerialConnection() function to configure baudrate and parity.

- Model 410
- Model 420
- Model 430
- Model 440
- Model 450
- Model 462
- Model 463
- Model 470
- Model 520
- Model 530
- Model 540
- Model 570
- Model 580

6.2 SeaDAC Modules

- Model 8221
- Model 8222
- Model 8223
- Model 8224

- Model 8225
- Model 8227
- Model 8232

6.3 SeaDAC Lite Modules

The SeaDAC Lite Discovery API offers an easy way to locate and identify SeaDAC Lite modules physically connected to the host system.

- Model 8111
- Model 8112
- Model 8113
- Model 8114
- Model 8115
- Model 8123
- Model 8126

6.4 el/O Modules

- Model 110
- Model 120
- Model 130
- Model 140
- Model 150
- Model 160
- Model 170

6.5 SeaRAQ Modules

- Model 6510
- Model 6511
- Model 6512
- Model 6513
- Model 6520
- Model 6525
- Model 8510

- Model 8511
- Model 8512
- Model 8520
- Model 8521

6.6 SeaCONNECT Modules

- Model 370

6.7 SeaIO 410

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle

SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.8 SealO 420

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle

SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.9 SeaIO 430

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle

SM_ReadDigitalInputs()
Read digital inputs

6.10 SeaIO 440

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.11 SeaIO 450

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.12 SeaIO 462

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).

SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_GetPIODirection() & SM_SetPIODirection()
Control the IO direction (input vs. output)
SM_GetPIOPresets() & SM_SetPIOPresets()
Control the power-on defaults for banks directed as outputs
SM_ReadPIO() & SM_WritePIO()
Read and write the entire 96-bit IO space
SM_ReadDigitalOutputs()
Read only the digital outputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when reading 8 outputs. (ie. 16-23)
SM_WriteDigitalOutputs()
Write only the digital outputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when writing 8 outputs. (ie. 16-23)

SM_ReadDigitalInputs()
Read only the digital inputs. Uses absolute addressing. Usable with 1 or 8 inputs only. Must start at a multiple of 8 when reading 8 inputs. (ie. 16-23)

6.13 SeaIO 463

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_GetPIODirection() & SM_SetPIODirection()
Control the IO direction (input vs. output)
SM_GetPIOPresets() & SM_SetPIOPresets()
Control the power-on defaults for banks directed as outputs
SM_ReadPIO() & SM_WritePIO()
Read and write the entire 96-bit IO space
SM_ReadDigitalOutputs()
Read only the digital outputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when reading 8 outputs. (ie. 16-23)
SM_WriteDigitalOutputs()
Write only the digital outputs. Uses absolute addressing. Usable with 1 or 8 outputs only. Must start at a multiple of 8 when writing 8 outputs. (ie. 16-23)
SM_ReadDigitalInputs()
Read only the digital inputs. Uses absolute addressing. Usable with 1 or 8 inputs only. Must start at a multiple of 8 when reading 8 inputs. (ie. 16-23)

6.14 SeaIO 470

SM_Open()
Open the Module

SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_WriteAnalogOutputs()
Write digital-to-analog outputs
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration
SM_SetAnalogInputConfig() & SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration

SM_GetAnalogOutputRanges()
Control the analog-to-digital and digital-to-analog interface setup

6.15 SeaIO 520

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.16 SeaIO 530

SM_Open()
Open the Module

SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.17 SeaIO 540

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).

SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.18 SeaIO 570

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID

SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs
SM_ReadAnalogInputs()
Read analog-to-digital inputs Passing a Channel Range Value of 0-5 will return a value corresponding to +-5. Passing a Channel Range Value of 0-15 will return a value corresponding to +-15.
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration

SM_SetAnalogInputConfig() & SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration Valid Analog to Digital Voltage References are Ground (1) and Floating (3). Valid Analog to Digital Channel Mode is Single Ended (0). Valid Channel Range Values are +-5 (1) and +-15 (3).

6.19 SeaIO 580

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_SetSoftwareAddress()
Configures a SeaI/O module's software selectable Modbus slave ID (only with rotary switch set to zero).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.20 SeaDAC 8221

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.

SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.21 SeaDAC 8222

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle

SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.22 SeaDAC 8223

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.23 SeaDAC 8224

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.24 SeaDAC 8225

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.

SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.25 SeaDAC 8227

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.
SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_WriteAnalogOutputs()
Write digital-to-analog outputs
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration
SM_SetAnalogInputConfig() & SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration
SM_GetAnalogOutputRanges()
Control the analog-to-digital and digital-to-analog interface setup

6.26 SeaDAC 8232

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SetCommunications()
Configures a SeaI/O module's listening baudrate and parity.

SM_GetDeviceConfig()
Query the SeaDAC module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.27 SeaDAC Lite 8111

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs

SM_WriteDigitalOutputs()
Write the digital outputs

6.28 SeaDAC Lite 8112

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.29 SeaDAC Lite 8113

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info

SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.30 SeaDAC Lite 8114

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.31 SeaDAC Lite 8115

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info

SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.32 SeaDAC Lite 8123

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.33 SeaDAC Lite 8126

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaDAC Lite module for model info

SM_Close()
Close the SeaMAX handle
SM_ReadPIO() & SM_WritePIO()
Read the entire programmable IO space.
SM_SetPIODirection() & SM_GetPIODirection()
Sets or determines the input vs output direction of a bank of 8 IO.

Note

Unlike other PIO models, the 8126 does not use the SM_SetPIOPresets() function.

6.34 eIO 110

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.35 eIO 120

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.36 eIO 130

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.37 eIO 140

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID

SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.38 eIO 150

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.39 eIO 160

SM_Open()
Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadPIO() & SM_WritePIO()
Read/write the entire programmable IO space.
SM_SetPIODirection() & SM_GetPIODirection()
Sets or determines the input vs output direction of a bank of 8 IO.
SM_GetPIOPresets() & SM_SetPIOPresets()
Control the power-on defaults for banks directed as outputs

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.40 eIO 170

SM_Open()

Open the Module
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration
SM_SetAnalogInputConfig() & SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration Valid Analog to Digital Voltage Reference is Ground (1). Valid Analog to Digital Channel Modes are Single Ended (0) and Differential (1).

Note

All eI/O Modules use the Ethernet Discovery & Configuration API.

6.41 SeaRAQ 6510

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadAnalogInputs()
Read analog-to-digital inputs Raw value only. Built-in conversion of analog values is not supported at this time. Device channel range configuration is hardware controlled only.

6.42 SeaRAQ 6511

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info

SM_Close()
Close the SeaMAX handle
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_GetAnalogConfig()
Query the analog-to-digital interface configuration

6.43 SeaRAQ 6512

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration

6.44 SeaRAQ 6513

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadAnalogInputs()
Read analog-to-digital inputs Raw value only. Built-in conversion of analog values is not supported at this time.
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration

SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration Valid Channel Range Values are +-10 (3), +-5 (1), +-1 (4), +-0.5 (5), +-0.15 (6), and inline 249 ohm resistor (7).

6.45 SeaRAQ 6520

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).

SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_WriteAnalogOutputs()
Write digital-to-analog outputs
SM_Close()
Close the SeaMAX handle

6.46 SeaRAQ 6525

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_WriteAnalogOutputs()
Write digital-to-analog outputs
SM_Close()
Close the SeaMAX handle

6.47 SeaRAQ 8510

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.48 SeaRAQ 8511

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info

SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs

6.49 SeaRAQ 8512

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.50 SeaRAQ 8520

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.51 SeaRAQ 8521

SM_Open()
Open the Module
SM_ConfigureSerialConnection()
Configures the local PC's serial port baudrate and parity (for serial SeaI/O modules only).
SM_SelectDevice()
Target a particular Modbus Slave ID

SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs

6.52 SeaCONNECT 370

SM_Open()
Open the Module
SM_GetDeviceConfig()
Query the SeaI/O module for model and communications info
SM_Close()
Close the SeaMAX handle
SM_ReadDigitalInputs()
Read digital inputs
SM_ReadDigitalOutputs()
Read the digital outputs
SM_WriteDigitalOutputs()
Write the digital outputs
SM_ReadAnalogInputs()
Read analog-to-digital inputs
SM_GetAnalogInputConfig() & SM_GetAnalogInputRanges()
Query the analog-to-digital interface configuration

SM_SetAnalogInputConfig() & SM_SetAnalogInputRanges()
Control the analog-to-digital interface configuration

Note

All SeaCONNECT Modules use the Ethernet Discovery & Configuration API.

Chapter 7

Global Types and Enumerations

Listed here are the types, enumerations, and definitions used throughout the SeaMAX library.

7.1 SeaMAX Enumerations

7.1.1 DeviceConfig

```
typedef struct configuration
{
  int model;
  int commType;
  int baudrate;
  int parity;
  int firmware;
} DeviceConfig;
```

7.1.2 AnalogConfig

```
typedef struct analog_configuration
{
  short channelMode;      // Analog Channel Configuration Flags
  char  precision;        // Number of bits A/D or D/A precision
  char  exponent;         // Two's complement exponent for channel ranges
  short minValue;         // Minimum range  (minValue * (10 ^ exponent))
  short maxValue;         // Maximum range  (maxValue * (10 ^ exponent))
} AnalogConfig;
```

7.1.3 Baudrates

```
enum
{
  SM_BAUD_INVALID     = 0,
  SM_BAUD_1200        = 1,
  SM_BAUD_2400        = 2,
  SM_BAUD_4800        = 3,
  SM_BAUD_9600        = 4,
  SM_BAUD_14400       = 5,
  SM_BAUD_19200       = 6,
  SM_BAUD_28800       = 7,
  SM_BAUD_38400       = 8,
  SM_BAUD_57600       = 9,
  SM_BAUD_115200      = 10,
  SM_BAUD_230400      = 11,
  SM_BAUD_460800      = 12,
  SM_BAUD_921600      = 13,
};
```

7.1.4 Databits

```
enum
{
  SM_DATABITS_5   = 5,
  SM_DATABITS_6   = 6,
  SM_DATABITS_7   = 7,
  SM_DATABITS_8   = 8,
};
```

7.1.5 Parity

```
enum
{
  SM_PARITY_NONE      = 0,
  SM_PARITY_ODD       = 1,
  SM_PARITY_EVEN      = 2,
  SM_PARITY_MARK      = 3,
  SM_PARITY_SPACE     = 4,
};
```

7.1.6 Stopbits

```
enum
{
  SM_STOPBITS_1       = 0,
  SM_STOPBITS_1_5     = 1,
  SM_STOPBITS_2       = 2,
};
```

7.1.7 Communications Type

```
enum
{
  SM_COMTYPE_RS485        = 0,
  SM_COMTYPE_ETHERNET     = 1,
  SM_COMTYPE_USB          = 2,
  SM_COMTYPE_RS232        = 3,
  SM_COMTYPE_EXPANSION    = 4,
  SM_COMTYPE_RS422        = 5,
  SM_COMTYPE_WIRELESS     = 6,
  SM_COMTYPE_INTELLIGENT  = 7,
  SM_COMTYPE_P_O_ETH      = 8,
  SM_COMTYPE_P_O_USB      = 9,
};
```

7.1.8 Analog to Digital Voltage Reference

```
enum
{
  ANALOG          = 0,
  GROUND          = 1,
  AD_REFERENCE    = 2,
  FLOATING        = 3,
  DA_CHANNEL_1    = 4,
  DA_CHANNEL_2    = 8,
};
```

7.1.9 Analog to Digital Channel Modes

```
enum
{
  SINGLE_ENDED    = 0,
  DIFFERENTIAL    = 1,
  CURRENT_LOOP    = 2,
  MULTI_MODE      = 3,
```

```
};
```

7.1.10 Channel Range Values

```
enum
{
  ZERO_TO_FIVE            = 0,
  ZERO_TO_TWENTYFOUR_MA   = 0,
  PLUS_MINUS_FIVE         = 1,
  ZERO_TO_TEN             = 2,
  PLUS_MINUS_TEN          = 3,
  ZERO_TO_FIFTEEN         = 2,
  PLUS_MINUS_FIFTEEN      = 3,
  PLUS_MINUS_ONE          = 4,
  PLUS_MINUS_HALF         = 5,
  PLUS_MINUS_150MILLI     = 6,
  INLINE_RESISTOR_200     = 7,
};
```

7.1.11 Analog Channel Configuration Flags

```
enum
{
  IS_OUTPUT            = 0x0001,
  IS_INPUT             = 0x0002,
  IS_SINGLE_ENDED      = 0x0004,
  IS_DOUBLE_ENDED      = 0x0008,
  IS_DIFFERENTIAL      = 0x000C,
  IS_VOLTAGE           = 0x0010,
  IS_CURRENT           = 0x0020,
  IS_POWER             = 0x0040,
  IS_FREQUENCY         = 0x0080,
  USES_FLOATING_GND    = 0x0100,
  USES_EXTERNAL_GND    = 0x0200,
  USES_INTERNAL_GND    = 0x0400,

  AMP_MODE_0           = 0x0000,
  AMP_MODE_1           = 0x1000,
  AMP_MODE_2           = 0x2000,
  AMP_MODE_3           = 0x3000,
  AMP_MODE_4           = 0x4000,
  AMP_MODE_5           = 0x5000,
  AMP_MODE_6           = 0x6000,
  AMP_MODE_7           = 0x7000,
  AMP_MODE_8           = 0x8000,
  AMP_MODE_9           = 0x9000,
  AMP_MODE_10          = 0xA000,
  AMP_MODE_11          = 0xB000,
  AMP_MODE_12          = 0xC000,
  AMP_MODE_13          = 0xD000,
  AMP_MODE_14          = 0xE000,
  AMP_MODE_15          = 0xF000,
};
```

7.2 SeaMAX Ethernet Enumerations

7.2.1 Network Types

```
enum
{
  NETWORK_INFRASTRUCTURE  = 0,
  NETWORK_ADHOC           = 1,
};
```

7.2.2 Wireless Security Type

```
enum
```

```
{
  SECURITY_NONE             = 0,
  SECURITY_WEP_OPEN_64      = 1,
  SECURITY_WEP_SHARED_64    = 2,
  SECURITY_WEP_OPEN_128     = 3,
  SECURITY_WEP_SHARED_128   = 4,
  SECURITY_WPA_TKIP         = 5,
  SECURITY_WPA2_AES         = 6,

  SECURITY_UNKNOWN          = 0xFF,
};
```

7.2.3 Wireless Key Type

```
enum
{
  HEX_KEY             = 0,
  PASSPHRASE_KEY      = 1,
};
```

7.2.4 Security Type

```
enum
{
  UNSECURED    = 0,
  AES_128      = 1,
  AES_256      = 2,
};
```

7.2.5 Error Code

```
enum ErrorCodes
{
  SM_SUCCESS                          = 0,  // Success
  SM_HANDLE_INVALID_ERR               = 1,  // Invalid SeaMAX handle
  SM_WIN32_ERR                        = 2,  // Win32 Error. See GetLastWin32Error()
  SM_FTDI_ERR                         = 3,  // FTDI Error. See GetLastFTDI32Error()
  SM_PARAM_INVALID_ERR                = 4,  // A parameter was out of range.
  SM_NOT_CONNECTED_ERR                = 5,  // A connection is not established.
  SM_MODBUS_ILLEGAL_FUNC_01_ERR       = 6,  // Illegal Modbus Function (Modbus exception 0x01)
  SM_MODBUS_ILLEGAL_ADDR_02_ERR       = 7,  // Illegal Data Address (Modbus exception 0x02)
  SM_MODBUS_ILLEGAL_DATA_03_ERR       = 8,  // Illegal Data Value (Modbus exception 0x03)
  SM_MODBUS_UNKOWN_ERR                = 9,  // Unkown Modbus error
  SM_CRC_ERR                          = 10, // CRC Error, possible communications problem
  SM_WRITE_ERR                        = 11, // Error writing message
  SM_TIMEOUT_ERR                      = 12, // Read timeout occurred
  SM_CMD_NOT_SUPPORTED_ERR            = 13, // Command not supported by this device
  SM_MUTEX_ERR                        = 14, // Could not acquire a valid mutex
  SM_NO_MODULES_FOUND_ERR             = 15, // No modules have been found. See SME_SearchForModules() or
      SDL_SearchForDevices()
  SM_END_DISC_MODULES_ERR             = 16, // End of discovered modules
  SM_MODULE_NOT_FOUND_ERR             = 17, // Could not locate named module
  SM_SET_SERIAL_PARAM_ERR             = 18, // Could not set serial parameters
  SM_NON_SERIAL_ERR                   = 19, // Current connection is non-serial
  SM_INVALID_RESPONSE_ERR             = 20, // Invalid response from device
  SM_DEVICE_NOT_SUPPORTED_ERR         = 21, // Device not supported or unable to read device ID.
  SM_BIND_SOCKET_ERR                  = 22, // Error binding to socket
  SM_FREE_SOCKET_ERR                  = 23, // Could not acquire free socket
  SM_HOST_ADDRESS_ERR                 = 24, // Could not resolve host address
  SM_GET_CONFIG_ERR                   = 25, // Error getting current configuration
  SM_NET_INTERFACES_ERR               = 26, // Error obtaining a list of local network interfaces
  SM_BROADCAST_ERR                    = 27, // Error sending broadcast message over interface
  SM_SET_CONNECTION_PARAMS_ERR        = 28, // Error setting connection parameters
  SM_CBX_DLL_ERR                      = 29, // Unable to load cbx_enc_2_1.dll
  SM_THREAD_NOT_STARTED_ERR           = 30, // NotifyOnInputChange thread not started
  SM_THREAD_UNKNOWN_ERR               = 31, // Unknown error on NotifyOnInputChange thread
  SM_THREAD_CANCELLED_ERR             = 32, // NotifyOnInputChange thread has been cancelled
  SM_THREAD_PENDING_ERR               = 33, // NotifyOnInputChange request already pending
  SM_THREAD_INPUT_CHANGED             = 34, // NotifyOnInputChange input changed
  SM_THREAD_READ_FAILED               = 35, // NotifyOnInputChange read failed: non-terminal error
};
```

Chapter 8

Deprecated List

Member CCEthernet::Alloc (int number)

Version 2.x - Version 5.0

Member CCEthernet::find_devices (ceth_device_type type_to_find, int number_to_find, ceth_device_p list_↩ to_store_devices)

Version 2.x - Version 5.0

Member CCEthernet::Free (ceth_device ∗)

Version 2.x - Version 5.0

Member CCEthernet::GetSeaMAXVersion3Handle ()

Version 2.x - Version 5.0

Member CCEthernet::set_information (ceth_device ∗device, ceth_set_types command,...)

Version 2.x - Version 5.0

Member CSeaMaxW32::Close ()

Version 2.x - Version 5.0

Member CSeaMaxW32::getCommHandle (void)

Version 2.x - Version 5.0

Member CSeaMaxW32::GetSeaMAXVersion3Handle ()

Version 2.x - Version 5.0

Member CSeaMaxW32::Ioctl (slave_address_t, IOCTL_t, void ∗)

Version 2.x - Version 5.0

Member CSeaMaxW32::Open (char ∗file)

Version 2.x - Version 5.0

Member CSeaMaxW32::Read (slave_address_t slaveId, seaio_type_t type, address_loc_t address, address_↩ range_t range, void ∗values)

Version 2.x - Version 5.0

Member CSeaMaxW32::set_intermessage_delay (int new_delay_time)

Version 2.x - Version 5.0

Member CSeaMaxW32::Write (slave_address_t slaveId, seaio_type_t type, address_loc_t address, address_↩ range_t range, unsigned char ∗data)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Close (CSeaMaxW32 *SeaMaxPointer)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Create ()

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Destroy (CSeaMaxW32 *p)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32GetVersion3Handle (CSeaMaxW32 *SeaMaxPointer)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Ioctl (CSeaMaxW32 *SeaMaxPointer, slave_address_t slaveId, IOCTL_t which, void *data)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Open (CSeaMaxW32 *SeaMaxPointer, char *filename)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Read (CSeaMaxW32 *SeaMaxPointer, slave_address_t slaveId, seaio_type_t type, address_loc_t starting_address, address_range_t range, void *data)

Version 2.x - Version 5.0

globalScope> Member SeaMaxW32Write (CSeaMaxW32 *SeaMaxPointer, slave_address_t slaveId, seaio_type_t type, address_loc_t starting_address, address_range_t range, unsigned char *data)

Version 2.x - Version 5.0

globalScope> Member SM_AtoDConversion (SM_HANDLE handle, double *value, unsigned char *data, int channelRange)

Version 3.0.4 - Version 5.0

globalScope> Member SM_DtoAConversion (SM_HANDLE handle, double value, unsigned char *data, int channelRange)

Version 3.0.4 - Version 5.0

globalScope> Member SM_GetADDAConfig (SM_HANDLE handle, unsigned char *deviceConfig, unsigned char *channelsConfig)

Version 3.0.4 - Version 5.0

globalScope> Member SM_GetADDAExtendedConfig (SM_HANDLE handle, int *multiplierEnabled, unsigned char *daConfig)

Version 3.0.4 - Version 5.0

globalScope> Member SM_GetConfig (SM_HANDLE handle, int *model, int *commType, int *baudrate, int *parity)

Version 3.2.4 - Version 5.0

globalScope> Member SM_ReadCoils (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

globalScope> Member SM_ReadDiscreteInputs (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

globalScope> Member SM_ReadHoldingRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

globalScope> Member SM_ReadInputRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

globalScope> Member SM_SetADDAConfig (SM_HANDLE handle, unsigned char *deviceConfig, unsigned char *channelsConfig)

Version 3.0.4 - Version 5.0

globalScope> Member SM_WriteCoils (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

globalScope> Member SM_WriteHoldingRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

Version 3.0.4 - Version 5.0

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Chapter 10

Module Documentation

10.1 SeaMAX API

Modules

- Deprecated Functions

Functions

- int __stdcall SM_Version (int *major, int *minor, int *revision, int *build)

  Returns the SeaMAX library's version info as major.minor.revision.build.
- int __stdcall SM_Open (SM_HANDLE *handle, char *connection)

  Opens a connection to a Sealevel I/O module.
- int __stdcall SM_OpenSecure (SM_HANDLE *handle, char *connection, int securityMode, char *securityKey)

  Opens a secure connection to a Sealevel I/O module.
- int __stdcall SM_ConfigureSerialConnection (SM_HANDLE handle, int baudrate, int parity)

  Configures the local PC's serial port baudrate (For Serial Connections Only).
- int __stdcall SM_ConfigureSerialTimeouts (SM_HANDLE handle, int multiple, int constant, int interval)

  Configures the local PC's serial port timeout parameters (For Serial Connections Only).
- int __stdcall SM_Close (SM_HANDLE handle)

  Closes the SeaMAX handle and releases all allocated memory.
- int __stdcall SM_SelectDevice (SM_HANDLE handle, int slaveID)

  Targets a new Modbus device.
- int __stdcall SM_SetPolarity (SM_HANDLE handle, bool activeLow)

  Configures the polarity of all Digital IO reads and writes.
- int __stdcall SM_GetDeviceConfig (SM_HANDLE handle, DeviceConfig *config)

  Queries the Sealevel I/O module to determine the module model number, type, baudrate, parity, and firmware version number.
- int __stdcall SM_GetAnalogConfig (SM_HANDLE handle, int number, AnalogConfig *configuration)

  Gets the device's analog configuration.
- int __stdcall SM_GetAnalogInputConfig (SM_HANDLE handle, unsigned char *reference, unsigned char *mode)

  Gets the device's analog configuration.
- int __stdcall SM_GetAnalogInputRanges (SM_HANDLE handle, unsigned char *ranges)

  Gets the device's analog inputs range configuration.

- int __stdcall SM_SetAnalogConfig (SM_HANDLE handle, int number, AnalogConfig *configuration)

  Sets the device's analog configuration.
- int __stdcall SM_SetAnalogInputConfig (SM_HANDLE handle, unsigned char reference, unsigned char mode)

  Sets the device's analog configuration.
- int __stdcall SM_SetAnalogInputRanges (SM_HANDLE handle, unsigned char *ranges)

  Sets the device's analog inputs range configuration.
- int __stdcall SM_GetAnalogOutputRanges (SM_HANDLE handle, unsigned char *ranges)

  Polls the Sealevel I/O device for its analog hardware jumper configuration.
- int __stdcall SM_GetPIOPresets (SM_HANDLE handle, unsigned char *config)

  Retrieves a Sealevel I/O module's programmable IO bit presets.
- int __stdcall SM_SetPIOPresets (SM_HANDLE handle, unsigned char *config)

  Configures a Sealevel I/O module's programmable IO bit presets.
- int __stdcall SM_GetPIODirection (SM_HANDLE handle, unsigned char *config)

  Retrieves a Sealevel I/O module's programmable IO direction.
- int __stdcall SM_SetPIODirection (SM_HANDLE handle, unsigned char *config)

  Configures a Sealevel I/O module's programmable IO direction.
- int __stdcall SM_ReadPIO (SM_HANDLE handle, unsigned char *values)

  Reads the entire I/O space of a Sealevel programmable IO device.
- int __stdcall SM_WritePIO (SM_HANDLE handle, unsigned char *values)

  Writes the IO space of a programmable I/O Sealevel I/O module.
- int __stdcall SM_SetCommunications (SM_HANDLE handle, int baudrate, int parity)

  Configures a Sealevel I/O module's communication parameters.
- int __stdcall SM_SetSoftwareAddress (SM_HANDLE handle, int slaveID)

  Configure's a Sealevel I/O module's software selectable Modbus slave ID (Modbus devices only).
- int __stdcall SM_ReadDigitalOutputs (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads the current state of one or more digital outputs.
- int __stdcall SM_ReadDigitalInputs (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads the current state of one or more digital inputs.
- int __stdcall SM_ReadAnalogInputs (SM_HANDLE handle, int start, int number, double *analogValues, unsigned char *ranges, unsigned char *byteValues)

  Reads one or more Sealevel I/O device analog input(s).
- int __stdcall SM_WriteDigitalOutputs (SM_HANDLE handle, int start, int number, unsigned char *values)

  Writes the state of one or more digital outputs.
- int __stdcall SM_WriteAnalogOutputs (SM_HANDLE handle, int start, int number, double *analogValues, unsigned char *ranges, unsigned char *byteValues)

  Writes one or more analog outputs.
- int __stdcall SM_NotifyInputState (SM_HANDLE handle, int cancel)

  Checks or cancels the notify input state status.
- int __stdcall SM_NotifyOnInputChange (SM_HANDLE handle, int start, int number, unsigned char *values, int delay, int blocking)

  Continuously reads the discrete inputs until a change occurs.
- int __stdcall SM_GlobalCommsReset (SM_HANDLE handle)

  Resets a connected device to default address ID and baudrate.
- int __stdcall SM_CustomMessage (SM_HANDLE handle, unsigned char *message, int messageSize, int expectedBytes)

  Sends a custom modbus message to the selected device, and retrieves the response.
- int __stdcall SM_GetLastError ()

Returns the most recent SeaMAX Error and clears the error code.

- int __stdcall SM_GetLastWin32Error ()

  Returns the most recent Win32 Error and clears the error code.

- int __stdcall SM_GetLastFTDIError ()

  Returns the most recent FTDI Error and clears the error code.

10.1.1 Detailed Description

The SeaMAX API consists of the functions outline below, and provides an easy-to-use interface for Sealevel I/O device configuration and access. For more information, click a function name below for detailed information on function use, parameter types, and return codes.

Note

Not every function below may apply to your specific Sealevel I/O device. A complete list of of model specific SeaMAX API functions is available in the SeaMAX by Sealevel Model Number document.

10.1.2 Function Documentation

10.1.2.1 int __stdcall SM_Version (int * *major,* int * *minor,* int * *revision,* int * *build*)

Returns the SeaMAX library's version info as major.minor.revision.build.

Parameters

out	*major*	
out	*minor*	
out	*revision*	
out	*build*	

Return values

0	Success

10.1.2.2 int __stdcall SM_Open (SM_HANDLE * *handle,* char * *connection*)

Opens a connection to a Sealevel I/O module.

Parameters

out	*handle*	SeaMAX handle. This integer will be filled with a valid handle for future SeaMAX operations after a successful open.
in	*connection*	String representing the connection to open. For Modbus TCP, the ":502" does not have to be specified.

Return values

0	Success.
-1	Parameter 'connection' is null.

-2	Could not determine connection type.
-3	Invalid connection string.
-10	Serial: Invalid or unavailable serial connection.
-11	Serial: Unable to acquire a valid mutex.
-12	Serial: Unable to set serial timeouts.
-13	Serial: Unable to set serial parameters (e.g. baudrate, parity, etc.).
-14	Serial: Invalid serial name parameter.
-20	Ethernet: Could not resolve host address.
-21	Ethernet: Host refused or unavailable.
-22	Ethernet: Could not acquire free socket.
-23	Ethernet: Could not acquire a valid mutex.
-30	SeaDAC Lite: Invalid or unavailable port
-31	SeaDAC Lite: Unable to acquire a mutex handle
-32	SeaDAC Lite: Invalid device number (should be zero or greater). Object invalid.
-33	SeaDAC Lite: Could not read Vendor ID
-34	SeaDAC Lite: Could not read Product ID
-40	Could not read USB device product or vendor ID.
-41	Non-Sealevel USB device.
-42	SeaMAX does not support this Sealevel USB device.

The connection parameter can be in many different forms. For instance, a COM based product such as a SeaI/O module can be opened with a connection string of the form "COMx" or "\\.\COMx". For an Ethernet connection, such as the SeaI/O E or W series module, a connection string formatted as "x.x.x.x:y" for a dotted notation socket and port connection, or "x:y" for a host name and port will suffice. If the 'y' (TCP/IP port) part of the connection is not specified, the default Modbus TCP port of 502 will be selected as default.

For example, "COM3" or "\\.\COM47" is appropriate for a serial connection, while "10.0.0.1" is appropriate for a socket-based connection.

Note

For Modbus compliant devices (such as the SeaI/O modules), a default slave ID of 247 is used for all SeaMAX functions. To read, write, or configure a module at an address other than 247, please refer to the SM_Select↩Device() function for more details.
A default local COM port baudrate of 9600 bps is chosen for any serial connection. If your I/O device communicates at a rate other than this, please use the SM_ConfigureSerialConnection().

For a SeaDAC Lite model device, the connection string should be formatted as "SeaDAC Lite x" where x is a number. Care should be taken to check to ensure that the device could be opened. For instance, assuming a PC has a SeaDAC Lite device installed, "SeaDAC Lite 0" would open the first one available.

Care should be taken to check to ensure that the device could be opened. For instance, assuming a PC has a SeaDAC Lite device installed, "SeaDAC Lite 0" would open the first one available.

Note

The connection parameter string is case sensitive. Some attempt at parameter checking is made, but it is probable that a malformed string will be successfully added as an TCP/IP connection. Therefore, care should be taken to ensure proper string formatting when opening connections.

Warning

The handle returned by SM_Open is not compatible with any of the other SeaMAX modules such as the SME (Ethernet Configuration) module or the SeaDAC SDL module. Only functions beginning with SM_ should use the handle returned by SM_Open().

References SM_ConfigureSerialTimeouts().

Referenced by CSeaMaxW32::Open().

10.1.2.3 int __stdcall SM_OpenSecure (SM_HANDLE * *handle*, char * *connection*, int *securityMode*, char * *securityKey*)

Opens a secure connection to a Sealevel I/O module.

Parameters

out	*handle*	SeaMAX handle. This integer will be filled with a valid handle for future SeaMAX operations after a successful open.
in	*connection*	String representing the connection to open. For Modbus TCP, the ":502" does not have to be specified.
in	*securityMode*	Type of encryption to use.
in	*securityKey*	C String representing the encryption key in hexadecimal characters

Return values

0	Success.
-1	Parameter 'connection' is null.
-2	Could not determine connection type.
-3	Invalid connection string.
-20	Ethernet: Could not resolve host address.
-21	Ethernet: Host refused or unavailable.
-22	Ethernet: Could not acquire free socket.
-23	Ethernet: Could not acquire a valid mutex.
-24	Ethernet: Unknown error establishing connection.
-25	Ethernet: Invalid key.
-26	Ethernet: Unable to load cbx_enc_2_1.dll.

This method establishes an Ethernet connection, such as the SeaI/O E or W series module, and a connection string formatted as "x.x.x.x:y" for a dotted notation socket and port connection, or "x:y" for a host name and port is required for the connection parameter. If the 'y' (TCP/IP port) part of the connection is not specified, the default Modbus TCP port of 502 will be selected as default.

For example, the connection string "10.0.0.1" will establish a socket-based connection on port 502 with a device having the IP Address "10.0.0.1".

For securityMode values, see Security Type.

The securityKey parameter must include a C String (null terminated) containing two hexadecimal characters per byte of encryption. For example, an AES 256 key would be represented as 64 hexadecimal characters: "0102030405060708090A0B0C0D0E0F10111213141516171819 1A1B1C1D1E1F20".

Note

For Modbus compliant devices (such as the SeaI/O modules), a default slave ID of 247 is used for all SeaMAX functions. To read, write, or configure a module at an address other than 247, please refer to the SM_SelectDevice() function for more details.

Warning

The handle returned by SM_OpenSecure is not compatible with any of the other SeaMAX modules such as the SME (Ethernet Configuration) module or the SeaDAC SDL module. Only functions beginning with SM_ should use the handle returned by SM_OpenSecure().

10.1.2.4 int __stdcall SM_ConfigureSerialConnection (SM_HANDLE *handle,* int *baudrate,* int *parity*)

Configures the local PC's serial port baudrate (For Serial Connections Only).

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*baudrate*	
in	*parity*	

Return values

0	Successful.
-1	Invalid SeaMAX handle.
-2	Connection not established.
-3	Current connection is non-serial.
-4	Could not configure serial connection, possible invalid parameter.

This function configures the PC's local serial port connection speed and parity - it does not configure any communication rates or parameters on the module itself.

In order to configure a module's communication parameters, see SM_SetCommunications().

See also

Baudrate Values, Parity Values

10.1.2.5 int __stdcall SM_ConfigureSerialTimeouts (SM_HANDLE *handle,* int *multiple,* int *constant,* int *interval*)

Configures the local PC's serial port timeout parameters (For Serial Connections Only).

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*multiple*	
in	*constant*	
in	*interval*	

Return values

0	Successful.
-1	Invalid SeaMAX handle.
-2	Connection not established.
-3	Current connection is non-serial.
-4	Could not configure serial timeouts, possible invalid parameter.

This function configures the PC's local serial port connection timeouts.

multiple: The multiplier parameter is used to calculate the total timeout for a read or write operation. For instance, if a serial operations requires sending 10 bytes and the multiple parameter provided is 15 (ms), the total timeout for that read or write will be 150 ms. A zero (0) indicates the multiple timeout should not be used.

constant: A constant may be added to the total timeout for read or write operations. For instance, in the multiple example above, a constant parameter of 200 would cause the total timeout to be 150 ms + 200 = 350 ms. A zero (0) indicates the constant timeout should not be used.

interval: An interval timeout can be used to timeout if the time between incoming characters exceeds the interval parameter (ms). For instance, an interval parameter of 10 would cause a timeout if the incoming characters have more than 10 ms delay between them. A zero (0) indicates the interval timeout should not be used.

Note

The interval parameter does not take effect until the first character is received. It is often prudent to provide a large constant timeout (e.g. 10000) as well to ensure that serial operations will return provided no data returns from a serial read.

Referenced by SM_Open().

10.1.2.6 int __stdcall SM_Close (SM_HANDLE *handle*)

Closes the SeaMAX handle and releases all allocated memory.

Parameters

in	*handle*	Valid handle returned by SM_Open().

Return values

0	Successful closure and cleanup.
-1	Invalid SeaMAX handle.

Warning

SM_Close() must be called in order to free all associated system memory associated with the SeaMAX library.

Referenced by CSeaMaxW32::Close().

10.1.2.7 int __stdcall SM_SelectDevice (SM_HANDLE *handle,* int *slaveID*)

Targets a new Modbus device.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*slaveID*	New Modbus slave ID to be used in future SeaMAX operations. Valid values are 0 - 247.

Return values

0	Success.
-1	Invalid SeaMAX handle.
-2	Invalid slave ID.

After calling SM_SelectDevice, all future SeaMAX operations will be directed at the indicated slave ID parameter. For instance, if COM3 has been opened with SM_Open() the default slave ID is 247, all reads and writes through SM_↩ ReadDigitalInputs() and SM_WriteDigitalOutputs() will be directed at the SeaI/O module with slave ID 247.

However, if there is another module chained to the first module, and it has a slave ID of 4, that module can be read from and written to by calling SelectDevice with a value of 4. Afterward, all operations will directed at the second module (slave ID 4) until the port is closed or SelectDevice is called again.

Note

Calling this function will terminate any outstanding notify on input change requests.

References SM_NotifyInputState().

Referenced by CSeaMaxW32::Ioctl(), CSeaMaxW32::Read(), and CSeaMaxW32::Write().

10.1.2.8 int __stdcall SM_SetPolarity (SM_HANDLE *handle,* bool *activeLow*)

Configures the polarity of all Digital IO reads and writes.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*activeLow*	Use active-low polarity instead of typical active-high.

Return values

0	Success.
-1	Invalid SeaMAX handle.
-2	No module detected.

Calling this modifier will cause all read and write Digital IO requests to be mutated to use the indicated polarity. Set the boolean value FALSE to use typical active-high logic states i.e. a logic high is representative of a physical high signal. Likewise set the boolean value TRUE to use the reverse logic i.e. a logic low is representative of a physical low or grounded signal.

When using Form C relays this will have the effect of reversing the NC and NO contacts.

Note

If a different device is selected at a future time, it is possible that the polarity could change again depending on the configuration of the selected device. Therefore, it is necessary to re-issue this call whenever selecting a new device to ensure consistency.

References SM_GetConfig().

10.1.2.9 int __stdcall SM_GetDeviceConfig (SM_HANDLE *handle,* DeviceConfig * *config*)

Queries the Sealevel I/O module to determine the module model number, type, baudrate, parity, and firmware version number.

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*config*	Valid DeviceConfig struct.

Return values

0	Success.
-1	Invalid SeaMAX handle.
-2	Connection not established.
-3	Error reading or writing to device.
-4	Invalid config parameter.

Each module contains an internally stored model number, communications type (USB, Ethernet, etc.), baudrate, parity, and firmware version number. This function returns those stored parameters as a way to identify what module is available on a particular slave ID, and the functionality of the firmware.

Note

For SeaDAC Lite modules, the model parameter will be the only valid parameter returned. All other values will be 0.

See also

DeviceConfig

Referenced by CSeaMaxW32::Ioctl().

10.1.2.10 int __stdcall SM_GetAnalogConfig (SM_HANDLE *handle,* int *number,* AnalogConfig * *configuration*)

Gets the device's analog configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*number*	Number of Analog IO points.
in	*configuration*	Array containing space for returned configurations of Analog IO points.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Invalid reference or mode parameter. May not be null.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).
-8	Modbus: CRC was invalid. Possible communications problem.

Gets the I/O device's Analog Channel settings.

For more information on the returned parameter values, see AnalogConfig and Analog Channel Configuration Flags.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

Warning

'Configuration' must contain at least six bytes for each analog input configuration requested in the 'number' field.

10.1.2.11 int __stdcall SM_GetAnalogInputConfig (SM_HANDLE *handle*, unsigned char * *reference*, unsigned char * *mode*)

Gets the device's analog configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*reference*	Analog to digital reference point.
in	*mode*	Device input mode.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Invalid reference or mode parameter. May not be null.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).

-8	Modbus: CRC was invalid. Possible communications problem.

Gets the I/O device's offset reference and A/D channel mode. The reference parameter represents which source each analog input is compared against. The mode parameter is the overall configuration of the channels and whether they are singled ended or paired.

For more information on the returned parameter values, see A/D Voltage Reference values and A/D Channel Modes.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

10.1.2.12 int __stdcall SM_GetAnalogInputRanges (SM_HANDLE *handle,* unsigned char * *ranges*)

Gets the device's analog inputs range configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*ranges*	Array of desired channel ranges.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Invalid parameter. Parameters may not be null.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).
-8	Modbus: CRC was invalid. Possible communications problem.

Gets the I/O device's analog input ranges and places them in the 'ranges' array parameter. For more information on the returned range values, see A/D Channel Range values.

The 'ranges' parameter must contain 16 bytes. Only the first X bytes will be valid, where X is the number of analog inputs on the device. The first byte corresponds to a desired range for analog input 1, the second byte for analog input 2, and so on.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

10.1.2.13 int __stdcall SM_SetAnalogConfig (SM_HANDLE *handle,* int *number,* AnalogConfig * *configuration*)

Sets the device's analog configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*number*	Number of Analog IO points.

in	*configuration*	Array containing configurations for each Analog IO point.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Invalid reference or mode parameter. May not be null.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).
-8	Modbus: CRC was invalid. Possible communications problem.

Sets the I/O device's Analog Channel settings.

For more information on the returned parameter values, see Analog Configuration and Analog Channel Configuration Flags.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

10.1.2.14 int __stdcall SM_SetAnalogInputConfig (SM_HANDLE *handle,* unsigned char *reference,* unsigned char *mode*)

Sets the device's analog configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*reference*	Analog to digital reference point.
in	*mode*	Device input mode.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Error retrieving the mode configuration.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).
-8	Modbus: CRC was invalid. Possible communications problem.
-9	Invalid device configuration.

Sets the I/O device's offset reference and A/D channel mode. The reference parameter determines which source each analog input is compared against. The mode parameter determines the overall configuration of the channels and whether they are singled ended or paired.

Note

The reference parameter for almost all purposes should be set to ANALOG. This is the normal reference point for single ended, differential, and current mode applications. Other reference points have been included as diagnostic tools or references for calibration.

For more information on the appropriate values, see A/D Voltage Reference values and A/D Channel Modes.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

10.1.2.15 int __stdcall SM_SetAnalogInputRanges (SM_HANDLE *handle,* unsigned char * *ranges*)

Sets the device's analog inputs range configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*ranges*	Array of desired channel ranges.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Error retrieving the mode configuration.
-3	Connection is not established. Check the provided Connection object state.
-4	Modbus: Read error waiting for response. Unknown Modbus exception.
-5	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-6	Modbus: Illegal Data Address (Modbus Exception 0x02).
-7	Modbus: Illegal Data Value (Modbus Exception 0x03).
-8	Modbus: CRC was invalid. Possible communications problem.
-9	Invalid device configuration.

Sets the I/O device's analog input ranges according to the 'ranges' array parameter. For more information on the appropriate values, see A/D Channel Range values.

The 'ranges' parameter must contain 16 bytes. Only one byte for each analog input will contain valid data, with the first byte corresponding to a desired range for analog input 1, the second byte for analog input 2, and so on.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

10.1.2.16 int __stdcall SM_GetAnalogOutputRanges (SM_HANDLE *handle,* unsigned char * *ranges*)

Polls the Sealevel I/O device for its analog hardware jumper configuration.

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*ranges*	Array of analog output ranges.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Error retrieving the Sealevel I/O device analog configuration.

-4	Error setting temporary analog configuration.
-5	Error writing to digital to analog output.
-6	Error reading the analog inputs.
-7	Invalid voltage returned by analog to digital converter.
-8	Error restoring the original I/O device settings.

Gets the analog I/O device's analog output ranges and places them in the ranges array parameter. For more information on the returned range values, see Channel Range Values.

The 'ranges' parameter will contain one byte for each analog output, with the first byte corresponding to the first analog output, and so on.

Note

This method only applies to certain models. See SeaMAX by Sealevel Model Number for details.

Warning

'Ranges' must contain at least one byte for each analog output.
The digital to analog channel outputs may be altered by calling this function. The ranges parameter should contain one byte for each analog output present on your device.

10.1.2.17 int __stdcall SM_GetPIOPresets (SM_HANDLE *handle,* unsigned char * *config*)

Retrieves a Sealevel I/O module's programmable IO bit presets.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*config*	Contains the presets ('on' or 'off') of the programmable IO.

Return values

>=0	Successful completion. Bytes successfully read.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.

The Seal/O 462 & 463 modules offer 96-bits of programmable IO in 12 banks, each of which may be configured as a bank of 8 inputs or 8 outputs. For those banks configured as outputs, the module must know the state of the IO on power-up or on direction change (from input to output).

The format of the returned config parameter is twelve (12) bytes, with each bit representing a bit in a PIO bank. The first byte configures the bit presets for PIO 1-8 (LSB to MSB) of bank 1, the second byte configures PIO 1-8 of bank 2, etc. In each byte's case, a '0' bit indicates the corresponding IO should be preset as 'off', and a '1' indicates an 'on' state.

The following illustrates the format:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
config[0]	IO 8 (IO 8)	IO 7 (IO 7)	IO 6 (IO 6)	IO 5 (IO 5)	IO 4 (IO 4)	IO 3 (IO 3)	IO 2 (IO 2)	IO 1 (IO 1)
config[1]	IO 8 (IO 16)	IO 7 (IO 15)	IO 6 (IO 14)	IO 5 (IO 13)	IO 4 (IO 12)	IO 3 (IO 11)	IO 2 (IO 10)	IO 1 (IO 9)
				...				
config[11]	IO 8 (IO 96)	IO 7 (IO 95)	IO 6 (IO 94)	IO 5 (IO 93)	IO 4 (IO 92)	IO 3 (IO 91)	IO 2 (IO 90)	IO 1 (IO 89)

Legend
'0' = Off
'1' = On

Parameter 'config' Layout of IO Direction

Warning

The parameter config should have at least 12 bytes allocated before calling GetProgrammableIOBitPresets().

10.1.2.18 int __stdcall SM_SetPIOPresets (SM_HANDLE *handle,* unsigned char * *config*)

Configures a Sealevel I/O module's programmable IO bit presets.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*config*	Contains the presets ('on' or 'off') of the programmable IO.

Return values

>0	Successful completion. Bytes successfully written.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.

This method provides a simpler, easier way to interface to the programmable IO Sealevel I/O modules. The SeaI/O 462 & 463 modules offer 96-bits of programmable IO in 12 banks, each of which may be configured as a bank of 8 inputs or 8 outputs. For those banks configured as outputs, the module must know the state of the IO on power-up or on direction change (from input to output).

The format of the config parameter should be a packed set of twelve (12) bytes, with each bit representing a bit in a bank of 12 8-bit programmable IO points. The configuration bytes are arranged such that the first byte configures the bit presets for IO 1-8 (LSB to MSB) of bank 1, the second byte configures IO 1-8 of bank 2, etc. In each bytes case, a bitwise '0' indicates the corresponding IO should be preset as 'off', and a bitwise '1' indicates an 'on' state. The following illustrates the format:

Note

This function has currently no effect on SeaDAC Lite modules.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
config[0]	IO 8 (IO 8)	IO 7 (IO 7)	IO 6 (IO 6)	IO 5 (IO 5)	IO 4 (IO 4)	IO 3 (IO 3)	IO 2 (IO 2)	IO 1 (IO 1)
config[1]	IO 8 (IO 16)	IO 7 (IO 15)	IO 6 (IO 14)	IO 5 (IO 13)	IO 4 (IO 12)	IO 3 (IO 11)	IO 2 (IO 10)	IO 1 (IO 9)
...								
config[11]	IO 8 (IO 96)	IO 7 (IO 95)	IO 6 (IO 94)	IO 5 (IO 93)	IO 4 (IO 92)	IO 3 (IO 91)	IO 2 (IO 90)	IO 1 (IO 89)

Legend

'0' = Off
'1' = On

Parameter 'config' Layout of IO Direction

10.1.2.19 int __stdcall SM_GetPIODirection (SM_HANDLE *handle,* unsigned char * *config*)

Retrieves a Sealevel I/O module's programmable IO direction.

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*config*	Contains the direction (input or output) of the programmable IO.

Return values

>=0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-10	SeaDAC Lite: Invalid model number.
-11	SeaDAC Lite: Unknown connection type.
-12	SeaDAC Lite: Error communicating with device.

The SeaI/O 462 & 463 modules offer 96-bits of programmable IO in 12 banks, each of which may be configured as a bank of 8 inputs or 8 outputs.

The format of the config parameter should be one byte for every 8 bits of programmable I/O, each byte representing banks 1 - 12. A non-zero byte indicates that the corresponding bank of PIO should be a bank of 8 inputs, zero indicating 8 outputs. For example, a non-zero value in config[3] would indicate that bank 4 should be a bank of inputs.

Warning

The parameter, config, must have at least one byte of space allocated for every 8 bits of I/O before calling this function.

Referenced by CSeaMaxW32::Ioctl(), and CSeaMaxW32::Read().

10.1.2.20 int __stdcall SM_SetPIODirection (SM_HANDLE *handle,* unsigned char * *config*)

Configures a Sealevel I/O module's programmable IO direction.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*config*	Contains the direction (input or output) of the programmable IO.

Return values

>=0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-10	SeaDAC Lite: Invalid model number.
-11	SeaDAC Lite: Unknown connection type.
-12	SeaDAC Lite: Error communicating with device.

This method provides a simpler, easier way to interface to the programmable IO Sealevel I/O modules. The SeaI/O 462 & 463 modules offer 96-bits of programmable IO in 12 banks, each of which may be configured as a bank of 8 inputs or 8 outputs. The most basic way of configuring the programmable IO module is by subsequent reads and writes to it's holding register map. This method masks many of those reads and writes and provides one simple interface to set IO direction.

The format of the config parameter should be one byte for every 8 bits of programmable I/O, each byte representing banks 1 - 12. A non-zero byte indicates that the corresponding bank of PIO should be a bank of 8 inputs, zero indicating 8 outputs. For example, a non-zero value in config[3] would indicate that bank 4 should be a bank of inputs.

Warning

When changing the direction of any PIO bank from input to output, or vice versa, it is worth mentioning that all banks set as output will reset to their previously saved, default preset state (if applicable - see SM_SetPIO↩ Presets()). For devices that do not support presets, their output banks will reset to an inactive state.

Referenced by CSeaMaxW32::Ioctl().

10.1.2.21 int __stdcall SM_ReadPIO (SM_HANDLE *handle*, unsigned char * *values*)

Reads the entire I/O space of a Sealevel programmable IO device.

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*values*	After completion, contains the state of the Sealevel I/O device's inputs and outputs.

Return values

>0	Successful completion. Number of bytes of valid data in pioValues.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.

-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-10	SeaDAC Lite: Invalid model number.
-11	SeaDAC Lite: Unknown connection type.
-12	SeaDAC Lite: Error communicating with device.

This method attempts to read the state of the programmable IO for any applicable Sealevel I/O module. The data is returned as a packed array of bytes reflecting the state of ports A1 to C4, MSB to LSB. For instance, a 96-bit PIO module would return 12 bytes of packed data as such:

```
values[0] = Port A1 (PIO 7 to 0 - MSB to LSB)
values[1] = Port B1 (PIO 15 to 8)
values[2] = Port C1      ...
values[3] = Port A2
values[4] = Port B2
         ...
values[11] = Port C4
```

Warning

The parameter, values, must have at least one bytes of space allocated for each bank of 8 I/O points before calling this function. For instance, a SeaDAC Lite module with 32 PIO would require 4 bytes of pre-allocated space.

Referenced by CSeaMaxW32::Read().

10.1.2.22 int __stdcall SM_WritePIO (SM_HANDLE *handle,* unsigned char * *values*)

Writes the IO space of a programmable I/O Sealevel I/O module.

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*values*	Desired state of the Sealevel I/O module's inputs and outputs.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-10	SeaDAC Lite: Invalid model number.
-11	SeaDAC Lite: Unknown connection type.

-12	SeaDAC Lite: Error communicating with device.

This method attempts to configure the state of all programmable IO. The data should be a packed array of bytes, in order of ports A1 to C4, MSB to LSB. For instance, a 96-bit PIO module would require a 12-byte array, packed as such:

```
values[0] = Port A1 (PIO 7 to 0 - MSB to LSB)
values[1] = Port B1 (PIO 15 to 8)
values[2] = Port C1     ...
values[3] = Port A2
        ...
values[11] = Port C4
```

Note

There is no need to segregate inputs and outputs within the values array. If any particular port is set as an input, rather than output, the corresponding byte in values will be ignored. Only ports which have been configured as outputs will be affected by this function. This function expects an array size of (the number of PIO / 8) bytes (e.g. 96 PIO would require 12 bytes).

Warning

This function requires an array size of (PIO / 8) bytes. Therefore, for a 96 programmable I/O module, 12 bytes of data would be required, and for a 32 I/O module only four is needed.

Referenced by CSeaMaxW32::Write().

10.1.2.23 int __stdcall SM_SetCommunications (SM_HANDLE *handle*, int *baudrate*, int *parity*)

Configures a Sealevel I/O module's communication parameters.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*baudrate*	Desired baudrate of the Sealevel I/O module.
in	*parity*	Desired parity.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Error reading or writing to Modbus device.
-4	Failed to perform implicit 'Get Module Configuration (0x45)

This method attempts to set the communications parameters of a Sealevel I/O module. The communications parameters of the host PC are not affected, and must be changed using the SM_ConfigureSerialConnection method.

Note

This method may not be applicable to all I/O devices. SeaDAC Lite modules are not affected by this function.

See also

Baudrate Values, Parity Values

Referenced by CSeaMaxW32::Ioctl().

10.1.2.24 int __stdcall SM_SetSoftwareAddress (SM_HANDLE *handle,* int *slaveID*)

Configure's a Sealevel I/O module's software selectable Modbus slave ID (Modbus devices only).

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*slaveID*	Desired slave ID of the Modbus compliant Sealevel I/O module.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Error writing to Modbus device.
-4	Failed to perform implicit 'Get Module Configuration (0x45)

This method attempts to set the software selectable address (slave ID) of a Sealevel I/O module, whose side rotary switch is set to the '0' position. By default, setting the side switch to zero will result in a Modbus slave ID address of 247.

The 'Set Software Slave ID' Modbus function requires a security key byte, received by performing a 'Get Module Configuration (0x45)'. If the security key has not previously been explicitly requested , the SetSoftwareAddress() method will automatically perform the 'Get Module Configuration' function prior to executing the 'Set Software Slave ID'

Note

If the side rotary switch is not in the zero position, this Modbus function will fail.

Referenced by CSeaMaxW32::Ioctl().

10.1.2.25 int __stdcall SM_ReadDigitalOutputs (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Reads the current state of one or more digital outputs.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Starting output to read (zero-indexed).
in	*number*	Quantity of outputs to read.
out	*values*	Output state(s).

Return values

>=0	Number of bytes successfully returned in result array.
-1	Invalid SeaMAX handle.
-2	Modbus: Connection is not established. Check the provided Connection object state.
-3	Modbus: Read error waiting for response. Unknown Modbus exception.
-4	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-5	Modbus: Illegal Data Address (Modbus Exception 0x02).
-6	Modbus: Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-20	SeaDAC Lite: Invalid model number.
-21	SeaDAC Lite: Invalid addressing.
-22	SeaDAC Lite: Error reading the device.

Reads the state of one or more digital outputs. The output state(s) parameter will be an array of bytes where each byte represents 8 outputs and their states. The LSB of the first byte (values[0]) will contain the first 8 output states and will be ordered from LSB to MSB.

For instance, a read of 18 outputs (starting at output 0) will be an array of three bytes as such:

```
values[0] = Outputs 7 to 0 (MSB to LSB)
```

```
values[1] = Outputs 15 to 8 (MSB to LSB)
values[2] = 6 padded bits followed by outputs 17 - 16.
```

Warning

The result array parameter should have enough allocated space, before calling this method, to hold 1 bit for every coil requested.

Referenced by SM_ReadCoils().

10.1.2.26 int __stdcall SM_ReadDigitalInputs (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Reads the current state of one or more digital inputs.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Starting input (zero-indexed).
in	*number*	Quantity of inputs to read.
out	*values*	Discrete input values.

Return values

>=0	Number of bytes successfully returned in result array.
-1	Invalid SeaMAX handle.
-2	Modbus: Connection is not established. Check the provided Connection object state.
-3	Modbus: Read error waiting for response. Unknown Modbus exception.
-4	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-5	Modbus: Illegal Data Address (Modbus Exception 0x02).
-6	Modbus: Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-20	SeaDAC Lite: Invalid model number.
-21	SeaDAC Lite: Invalid addressing.
-22	SeaDAC Lite: Error reading the device.

Reads the state of one or more digital inputs. The digital input values in the values array (after a successful read) are packed as one input per bit. The LSB of the first byte (values[0]) contains the first addressed input. The next inputs follow toward the high order end of this byte, and then from low order to high order in subsequent bytes.

For instance, a read of 18 inputs would result in a three byte values array:

```
values[0] = Input 7 to Input 0 (MSB to LSB)
values[1] = Input 15 to Input 8
values[2] = 6 Padding bits followed by Input 17 to Input 16
```

Warning

The result array parameter should have enough allocated space, before calling this method, to hold 1 bit for every input requested.

Referenced by SM_ReadDiscreteInputs().

10.1.2.27 int __stdcall SM_ReadAnalogInputs (SM_HANDLE *handle*, int *start*, int *number*, double * *analogValues*, unsigned char * *ranges*, unsigned char * *byteValues*)

Reads one or more Sealevel I/O device analog input(s).

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Starting input.
in	*number*	Quantity of analog inputs to read.
out	*analogValues*	Input values as floating point values (voltages).
in	*ranges*	Array of channel ranges that correspond directly to each requested input.
out	*byteValues*	Register state values as 16-bit byte pairs.

Return values

>=0	Number of bytes successfully returned in result array.
-1	Invalid SeaMAX handle.
-2	Both 'values' or 'doubleValues' are null. Supply either one or both.
-3	If an array of doubles is supplied, an similar array of analog input ranges must also be supplied.
-4	One or more of the analog input ranges in the 'ranges' array is invalid.
-5	Connection is not established. Check the provided Connection object state.
-6	Read error waiting for response. Unknown Modbus exception.
-7	Illegal Modbus Function (Modbus Exception 0x01).
-8	Illegal Data Address (Modbus Exception 0x02).
-9	Illegal Data Value (Modbus Exception 0x03).
-10	Modbus CRC was invalid. Possible communications problem.

Reads one or more analog inputs. There are two ways that the analog input values can be returned: an array of 16-bit byte values, or as an array of double (floating-point) values.

Note

This function will return, via parameters, either an array of bytes, an array of doubles, or both. If you pass either the byte array or double array as null, that type of data will **not** be returned.

Double Array - Voltage Values

If an array of doubles is supplied, the array must contain at least one double for each analog input requested. The array will be filled with floating point voltage values after a successful completion. An array of analog input ranges is also required to convert the device's response to appropriate floating point values. The ranges array must have one byte corresponding to an appropriate channel range for each input requested. For instance, if three analog inputs are requested starting at input 2, the ranges array should contain:

```
ranges[0] = Input 2 Channel Range
ranges[1] = Input 3 Channel Range
ranges[2] = Input 4 Channel Range
```

The channel ranges for any analog I/O device can be retrieved using the SM_GetAnalogInputRanges() function.

Byte Array - 16-bit Byte Values

If an array of bytes is supplied, the array must contain at least two bytes for every analog input request. If only a byte array is supplied, the ranges parameter is not required. The byte array will be filled with the two byte values read directly from the analog I/O device. The byte order is such that the high-byte always comes first. For instance, if two analog inputs are read starting at input 4, the byte array returned will contain:

```
byteValues[0] = Input 4 (High Byte)
byteValues[1] = Input 4 (Low Byte)
byteValues[2] = Input 5 (High Byte)
byteValues[3] = Input 5 (Low Byte)
```

Warning

If a doubles array is supplied, it should contain space allocated for one double for each of the analog inputs requested. If a byte array is supplied, it should contain two bytes of space allocated for each input.

Referenced by SM_ReadInputRegisters().

10.1.2.28 int __stdcall SM_WriteDigitalOutputs (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Writes the state of one or more digital outputs.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Starting output (zero-indexed).
in	*number*	Quantity of outputs to write.
in	*values*	Desired output state(s).

Return values

>=0	Number of bytes successfully written.
-1	Invalid SeaMAX handle.
-2	Modbus: Connection is not established. Check the provided Connection object state.
-3	Modbus: Read error waiting for response. Unknown Modbus exception.
-4	Modbus: Illegal Modbus Function (Modbus Exception 0x01).
-5	Modbus: Illegal Data Address (Modbus Exception 0x02).
-6	Modbus: Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-20	SeaDAC Lite: Invalid model number.
-21	SeaDAC Lite: Invalid addressing.
-22	SeaDAC Lite: Error writing to the device.

The desired output state(s) parameter ('values') should be an array of bytes, where each byte represents 8 output states. The LSB of the first byte (values[0]) should contain the first 8 output states and should be ordered from LSB to MSB.

For instance, a write of 18 outputs should be an array of 3 bytes as such:

```
values[0] = Outputs 7 to 0 (MSB to LSB)
values[1] = Outputs 15 to Outputs 8
values[2] = 6 padded bits followed by outputs 17 - 16.
```

Referenced by SM_WriteCoils().

10.1.2.29 int __stdcall SM_WriteAnalogOutputs (SM_HANDLE *handle*, int *start*, int *number*, double * *analogValues*, unsigned char * *ranges*, unsigned char * *byteValues*)

Writes one or more analog outputs.

Parameters

in	*handle*	Valid handle returned by SM_Open().

in	*start*	Starting output (zero-indexed).
in	*number*	Quantity of analog outputs to write.
in	*analogValues*	Desired output values as floating point values (voltages).
in	*ranges*	Array of analog output ranges.
in	*byteValues*	Desired output values as 16-bit byte values.

Return values

>=0	Number of bytes successfully written.
-1	Invalid SeaMAX handle.
-2	Either 'byteValues' or 'doubleValues' must be null. Supply either one, but not both.
-3	A doubles array was provided without an array of channel ranges.
-4	One or more of the analog output channel ranges was an invalid range.
-5	Connection is not established. Check the provided Connection object state.
-6	Read error waiting for response. Unknown Modbus exception.
-7	Illegal Modbus Function (Modbus Exception 0x01).
-8	Illegal Data Address (Modbus Exception 0x02).
-9	Illegal Data Value (Modbus Exception 0x03).
-10	Modbus CRC was invalid. Possible communications problem.

Writes one or more analog output values. This function may be called one of two ways: either as with an array of floating point voltage values (with corresponding ranges), or with an array of 16-bit byte values.

Note

This function will either take an array of doubles (and it's corresponding ranges), or an array of bytes - but **not** both.

Double Array - Output Voltages

If an array of doubles is supplied, the array must contain at least one double for each analog output to be written. Each floating point value corresponds to the desired voltage output for each channel. Along with an array of floating point values, an array of analog output channel ranges must also be supplied - which map directly to the target analog outputs.

For instance, if two analog outputs are to be written the values 2.56 V and 5.64 V consecutively starting at output 3, the following arrays and values would need to be supplied:

```
analogValues[0] = 2.56
analogValues[1] = 5.64

ranges[0] = Analog Output 3's Range
ranges[1] = Analog Output 4's Range
```

The analog output ranges can be found by calling the SM_GetAnalogHardwareInfo() function.

Byte Array - 16-bit Byte Values

If an array of bytes is supplied, the array must contain at least two bytes for every analog output to be written. The byte array should be filled with two-byte values to be written directly to the analog I/O device's registers. The byte order is such that the high-byte is always first. For instance, if two anaog outputs are to be written at input 4 with the values 0x0F73 and 0x011A, the byte array should contain:

```
byteValues[0] = 0x0F
byteValues[1] = 0x73
byteValues[2] = 0x01
byteValues[3] = 0x1A
```

Referenced by SM_WriteHoldingRegisters().

10.1.2.30 int __stdcall SM_NotifyInputState (SM_HANDLE *handle,* int *cancel*)

Checks or cancels the notify input state status.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*cancel*	Non zero value indicates the current notify operation should be canceled.

Return values

2	Last input read failed. Will try again (non-terminal).
1	Input state has changed. 'Values' parameter to SM_NotifyOnInputChange() now contains valid input state.
0	No change in inputs detected.
-1	Invalid SeaMAX handle.
-2	There is no currently outstanding NotifyOnInputChange request started. You must call SM_NotifyOnInputChange() first.
-3	Could not read inputs. Parameters may be incorrect or the device may not be reachable, present, or may be busy.
-4	Read request has been canceled.

This function allows the caller to check on the status of a notify request that was previously issued. Also, SM_Notify↩ InputState allows the caller to cancel an outstanding request. See SM_NotifyOnInputChange() for more details.

Note

One or more of the return values are non-terminal. Non-terminal error codes indicates that there was a problem which did not force a cancelation of the request, and that the request will continue until completed or until canceled by the caller.

See also

SM_NotifyOnInputChange()

Referenced by SM_SelectDevice().

10.1.2.31 int __stdcall SM_NotifyOnInputChange (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*, int *delay*, int *blocking*)

Continuously reads the discrete inputs until a change occurs.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Modbus address to begin the read (zero-indexed).
in	*number*	Quantity of inputs to read.
out	*values*	On input change, this buffer will be populated with the input state.
in	*delay*	Time to delay between issuing reads (ms).
in	*blocking*	Indicates whether to wait for a change (non-zero), or immediately return control to the caller (zero).

Return values

1	Input state has changed. 'Values' parameter now contains valid input state. (Blocking mode only)

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	There is already a pending NotifyOnInputChange request. See SM_NotifyInput↩ State() to determine it's state or to cancel it.
-10	Could not read inputs. Parameters may be incorrect or the device may not be reachable, present, or may be busy.

SM_NotifyOnInputChange will continuously read the inputs until a change is detected, similar to consecutive SM↩ _ReadDigitalInputs() calls. The delay between reads is configurable based on the delay parameter. After an input change, the values parameter will be populated with the current input state.

Note

Only one notify request can be outstanding at a time. SM_ReadDigitalInputs() has more information on parameter values.

There are two modes of operation for this function: blocking and non-blocking. If the blocking parameter is supplied as non-zero, SM_NotifyOnInputChange will not return until a input change occurs.

Since blocking mode may be dangerous in instances where a logic change can not be guaranteed, a non-blocking notify request may be issued that will begin reading the inputs and terminate once canceled or a change occurs. Execution returns immediately to the caller, whereby the SM_NotifyInputState() function provides an interface to cancel the request or check on it's status.

Note

See the SM_NotifyInputState() function for more information on cancelling or checking the status of an outstanding request.

Warning

The 'values' parameter should be pre-allocated with enough bytes of space to hold 'number' bits of input states. For instance, if 12 inputs are being read, the values parameter should have 2 bytes of space allocated before calling this function.
The 'values' parameter should not be accessed (read or written) until SM_NotifyInputState() indicates that the request has completed (either canceled by the user or completed due to input change). Accessing the values buffer may prevent the notify request from completing, resulting in data loss.

See also

SM_NotifyInputState()

10.1.2.32 int __stdcall SM_GlobalCommsReset (SM_HANDLE *handle*)

Resets a connected device to default address ID and baudrate.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.

Note

After issuing this command, the connected device will be changed to factory default settings. The address ID will be set to the position of the rotary switch and the baudrate will be set to 9600. If the hardware position is on 0 the address ID will be set to 247 and 9600 baud.

Warning

This API function only applies to SeaIO devices with a an address rotary switch.

See also

Communication Type Values

Referenced by CSeaMaxW32::Write().

10.1.2.33 int __stdcall SM_CustomMessage (SM_HANDLE *handle*, unsigned char * *message*, int *messageSize*, int *expectedBytes*)

Sends a custom modbus message to the selected device, and retrieves the response.

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*message*	Message to send beginning with the function code. It will also contain the response on success.
in	*messageSize*	Size in bytes of the outgoing message. Must be less than 1017 bytes.
in	*expectedBytes*	Size in bytes of the expected response. Must be less than 1017 bytes.

Return values

>=0	Number of bytes successfully read.
-1	Invalid SeaMAX handle or connection type or invalid parameter.
-2	Modbus: Connection is not established. Check the provided Connection object state.
-3	Modbus: Read error waiting for response. Unknown Modbus exception.
-7	Modbus CRC was invalid. Possible communications problem.

The message parameter should contain the function code and remaining bytes in the message. The header, the Slave ID and the CRC are not required because SeaMAX handles these elements of the message.

Note

The message parameter must contain an array large enough to hold the return message.

See also

http://www.modbus.org

10.1.2.34 int __stdcall SM_GetLastError ( )

Returns the most recent SeaMAX Error and clears the error code.

Return values

ErrorCode	value

When an error occurs, an internal error value is set. Calling this method will return the stored error and reset the stored value. Calling SM_GetLastError() after an error has occurred is not required.

See also

ErrorCodes

10.1.2.35 int __stdcall SM_GetLastWin32Error ( )

Returns the most recent Win32 Error and clears the error code.

Return values

Win32	error value

When an error occurs, an internal error value is set. Calling this method will return the stored Win32 error and reset the stored value. Calling SM_GetLastWin32Error() after an error has occurred is not required. For more information on Win32 errors, please see the Win32 API documentation.

10.1.2.36 int __stdcall SM_GetLastFTDIError ( )

Returns the most recent FTDI Error and clears the error code.

Return values

FTDI	error value

When an FTDI error occurs, an internal error value is set. Calling this method will return the stored FTDI error and reset the stored value. Calling SM_GetLastFTDIError() after an error has occurred is not required. For more information on FTDI errors, please see the FTDI API documentation.

10.2 Deprecated Functions

Functions

- int __stdcall SM_GetConfig (SM_HANDLE handle, int *model, int *commType, int *baudrate, int *parity)

  Queries the Sealevel I/O module to determine the module model number, type, baudrate, and parity.
- int __stdcall SM_SetADDAConfig (SM_HANDLE handle, unsigned char *deviceConfig, unsigned char *channelsConfig)

  Sets the A/D configuration for a Sealevel I/O module.
- int __stdcall SM_GetADDAConfig (SM_HANDLE handle, unsigned char *deviceConfig, unsigned char *channelsConfig)

  Retrieves the A/D configuration for a Sealevel I/O module.
- int __stdcall SM_GetADDAExtendedConfig (SM_HANDLE handle, int *multiplierEnabled, unsigned char *da↩ Config)

  Polls the Sealevel I/O device hardware for analog specific jumper settings.
- int __stdcall SM_ReadCoils (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads one or more digital outputs.
- int __stdcall SM_ReadDiscreteInputs (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads one or more digital inputs.
- int __stdcall SM_ReadHoldingRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads a 16-bit value from the Sealevel I/O device, depending on the model.
- int __stdcall SM_WriteCoils (SM_HANDLE handle, int start, int number, unsigned char *values)

  Writes one or more digital outputs.
- int __stdcall SM_WriteHoldingRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

  Writes one or more Modbus Holding Registers.
- int __stdcall SM_ReadInputRegisters (SM_HANDLE handle, int start, int number, unsigned char *values)

  Reads a 16-bit value from the Sealevel I/O device, depending on the model.
- int __stdcall SM_AtoDConversion (SM_HANDLE handle, double *value, unsigned char *data, int channelRange)

  Converts a two-byte input register value to a double type voltage.
- int __stdcall SM_DtoAConversion (SM_HANDLE handle, double value, unsigned char *data, int channelRange)

  Converts a given floating point voltage value to an appropriate 16-bit D/A holding register value.

10.2.1 Detailed Description

Some functions have been deprecated due to organizational changes in SeaMAX or to provide clarity of use and function. Standard SeaMAX functions (where available) have been identified in the function descriptions and should be used if possible.

10.2.2 Function Documentation

10.2.2.1 int __stdcall SM_GetConfig (SM_HANDLE *handle*, int * *model*, int * *commType*, int * *baudrate*, int * *parity*)

Queries the Sealevel I/O module to determine the module model number, type, baudrate, and parity.

Deprecated Version 3.2.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*model*	Model number.
out	*commType*	Type of communications interface.
out	*baudrate*	
out	*parity*	

Return values

0	Success.
-1	Invalid SeaMAX handle.
-2	Connection not established.
-3	Error reading or writing to device.

Note

This function has been replaced by SM_GetDeviceConfig(). See the updated function for parameter and return values.

Each module contains an internally stored model number, communications type (USB, Ethernet, etc.), baudrate, and parity. This function returns those stored paramters as a way to identify what module is available on a particular slave ID.

Note

For SeaDAC Lite modules, the model parameter will be the only vaild parameter returned. CommType, baudrate, and parity may be supplied as null parameters (zero). Null or zero will be returned for any parameters supplied other than model.

See also

Communication Type Values, Baudrate Values, Parity Values

Referenced by CSeaMaxW32::Read(), and SM_SetPolarity().

10.2.2.2 int __stdcall SM_SetADDAConfig (SM_HANDLE *handle,* unsigned char * *deviceConfig,* unsigned char * *channelsConfig*)

Sets the A/D configuration for a Sealevel I/O module.

Deprecated Version 3.0.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*deviceConfig*	Desired configuration of the module.
in	*channelsConfig*	Channels' voltage ranges.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unkown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.
-8	Invalid device configuration.

Note

This function has been replaced by SM_SetAnalogInputConfig() and SM_SetAnalogInputRanges(). See the updated function for parameter and return values.

Sets the module's offset reference, A/D mode, and channel ranges. The deviceConfig parameter should contain two bytes: deviceConfig[0] should contain the A/D reference (zero for most applications), and deviceConfig[1] should contain the A/D mode. ChannelsConfig should contain one byte for each analog input, with each byte corresponding to that inputs channel range (e.g. channelsConfig[3] would contain the intended channel range for analog input 4).

Layout of deviceConfig

daConfig[0]	Analog Reference
daConfig[1]	AD Channel Mode

Layout of channelsConfig

channelsConfig[0]	A/D Channel 1 Range
channelsConfig[1]	A/D Channel 2 Range
channelsConfig[2]	A/D Channel 3 Range
...	...

Layout of Parameters

For more information on the appropriate bit values, see A/D Voltage Reference values, A/D Channel Modes, and A/D Channel Range values.

Referenced by CSeaMaxW32::Ioctl().

10.2.2.3 int __stdcall SM_GetADDAConfig (SM_HANDLE *handle,* unsigned char * *deviceConfig,* unsigned char * *channelsConfig*)

Retrieves the A/D configuration for a Sealevel I/O module.

Deprecated Version 3.0.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*deviceConfig*	Configuration of the module.
out	*channelsConfig*	A/D channels' voltage ranges.

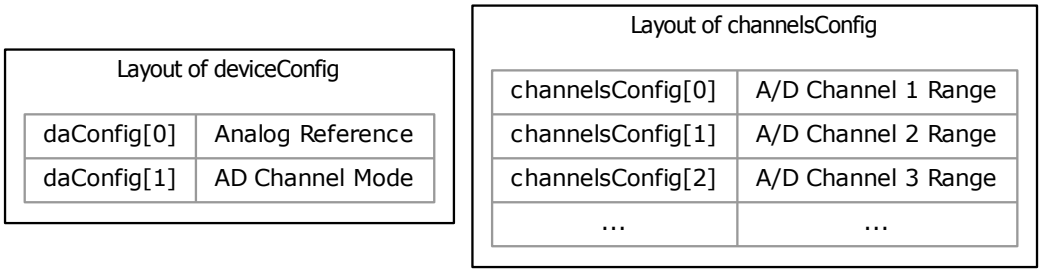
Return values

0	Success.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unkown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.

Note

This function has been replaced by SM_GetAnalogInputConfig() and SM_GetAnalogInputRanges(). See the updated function for parameter and return values.

Retrieves the module's offset reference, A/D mode, and channel ranges. The deviceConfig parameter will contain two bytes: deviceConfig[0] containing the analog reference point (should be zero for most applications), and deviceConfig[1] containing the A/D channel mode. The channelsConfig parameter will contain one byte for every analog input, indicating that input's channel range (e.g. channelsConfig[4] indicates analog input 5's channel range).



Layout of Parameters

For more information on the appropriate bit values, see A/D Voltage Reference values, A/D Channel Modes, and A/D Channel Range values.

Warning

The parameter deviceConfig should have at least 2 bytes, and channelsConfig should have at least 16 bytes, allocated before calling SM_GetADDAConfig().

Referenced by CSeaMaxW32::Ioctl().

10.2.2.4 int __stdcall SM_GetADDAExtendedConfig (SM_HANDLE *handle,* int * *multiplierEnabled,* unsigned char * *daConfig*)

Polls the Sealevel I/O device hardware for analog specific jumper settings.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_GetAnalogOutputRanges(). See the updated function for parameter and return values.

Referenced by CSeaMaxW32::Ioctl().

10.2.2.5 int __stdcall SM_ReadCoils (SM_HANDLE *handle,* int *start,* int *number,* unsigned char * *values*)

Reads one or more digital outputs.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_ReadDigitalOutputs(). See the updated function for parameter and return values.

References SM_ReadDigitalOutputs().

Referenced by CSeaMaxW32::Read().

10.2.2.6 int __stdcall SM_ReadDiscreteInputs (SM_HANDLE *handle,* int *start,* int *number,* unsigned char * *values*)

Reads one or more digital inputs.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_ReadDigitalInputs(). See the updated function for parameter and return values.

References SM_ReadDigitalInputs().

Referenced by CSeaMaxW32::Read().

10.2.2.7 int __stdcall SM_ReadHoldingRegisters (SM_HANDLE *handle,* int *start,* int *number,* unsigned char * *values*)

Reads a 16-bit value from the Sealevel I/O device, depending on the model.

Deprecated Version 3.0.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*start*	Modbus address to begin the read (zero-indexed).
in	*number*	Quantity of holding registers to read.

out	*values*	Register state values.

Return values

>=0	Number of bytes successfully returned in result array.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unkown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus Exception 0x02).
-6	Illegal Data Value (Modbus Exception 0x03).
-7	Modbus CRC was invalid. Possible communications problem.

This function performs different tasks depending on the model of Sealevel I/O device. For more information, see

Note

Modbus holding registers are 16-bit wide registers. Therefore, each read will result in 2 bytes in the result array. For multi-register reads, the high-order byte will reside in the first byte (result[0]) with the lower-order byte residing thereafter (result[1]). Other register bytes subsequently follow the same pattern.

Warning

The result array parameter should have enough allocated space, before calling this method, to hold 2 bytes for every register requested.

Referenced by CSeaMaxW32::Read().

10.2.2.8 int __stdcall SM_WriteCoils (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Writes one or more digital outputs.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_WriteDigitalOutputs(). See the updated function for parameter and return values.

References SM_WriteDigitalOutputs().

Referenced by CSeaMaxW32::Write().

10.2.2.9 int __stdcall SM_WriteHoldingRegisters (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Writes one or more Modbus Holding Registers.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_WriteAnalogOutputs(). See the updated function for parameter and return values.

References SM_WriteAnalogOutputs().

Referenced by CSeaMaxW32::Write().

10.2.2.10 int __stdcall SM_ReadInputRegisters (SM_HANDLE *handle*, int *start*, int *number*, unsigned char * *values*)

Reads a 16-bit value from the Sealevel I/O device, depending on the model.

Deprecated Version 3.0.4 - Version 5.0

This function has been replaced by SM_ReadAnalogInputs(). See the updated function for parameter and return values.

References SM_ReadAnalogInputs().

Referenced by CSeaMaxW32::Read(), and CSeaMaxW32::Write().

10.2.2.11 int __stdcall SM_AtoDConversion (SM_HANDLE *handle*, double * *value*, unsigned char * *data*, int *channelRange*)

Converts a two-byte input register value to a double type voltage.

Deprecated Version 3.0.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
out	*value*	After completion, contains a double floating-point voltage representation of 'data'.
in	*data*	Two bytes of data - usually returned by SM_ReadInputRegisters().
in	*channelRange*	A/D channel range of the Sealevel I/O module.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Unknown voltage range.

This function has been provided as a way to simply for calculations required to convert the 16-bit data returned by SM_ReadInputRegisters() into a more usuable double-precision floating point form. This function converts a two-byte array into a single floating point given a valid A/D channel range.

Note

If your particular Sealevel I/O device supports current mode conversions, call this function to return the voltage reading across the internal precision resistor. To convert to a current reading, divide that value by 249.0(Ohms).

See also

Channel Range values, Manual Conversion of A/D Values

10.2.2.12 int __stdcall SM_DtoAConversion (SM_HANDLE *handle*, double *value*, unsigned char * *data*, int *channelRange*)

Converts a given floating point voltage value to an appropriate 16-bit D/A holding register value.

Deprecated Version 3.0.4 - Version 5.0

Parameters

in	*handle*	Valid handle returned by SM_Open().
in	*value*	Floating point desired D/A voltage value.
out	*data*	After completion, contains two bytes of data - usefull for calling SM_WriteHoldingRegisters().
in	*channelRange*	D/A channel range.

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.
-2	Unknown or invalid voltage range.

This function has been provided to simplify the converstion between a desired voltage output value and an appropriate 16-bit holding register value required by SM_WriteHoldingRegisters().

Note

Appropriate ranges are limited to ZERO_TO_TEN and ZERO_TO_FIVE.

See also

Channel Range values and SM_GetADDAExtendedConfig()

10.3 SeaMAX Ethernet Discovery / Configuration API

Functions

- int __stdcall SME_Initialize (SME_HANDLE *handle)

  Initializes the interface and allocates internal memory.
- int __stdcall SME_Cleanup (SME_HANDLE handle)

  De-allocates memory used by the SME interface.
- int __stdcall SME_SearchForModules (SME_HANDLE handle)

  Attempts to discover all reachable Sealevel I/O Ethernet modules via a UDP broadcast.
- int __stdcall SME_FirstModule (SME_HANDLE handle)

  Sets the internal pointer to the first discovered module.
- int __stdcall SME_NextModule (SME_HANDLE handle)

  Advances the internal pointer to the next discovered module.
- int __stdcall SME_ModuleByName (SME_HANDLE handle, char *moduleName)

  Advances the internal pointer to the first discovered module whose name matches the parameter.
- int __stdcall SME_Ping (SME_HANDLE handle)

  Contacts the currently selected module to ensure it's powered and accessible.
- int __stdcall SME_ModuleByIP (SME_HANDLE handle, char *ipAddress)

  Advances the internal pointer to the first discovered module whose IP address matches the parameter.
- int __stdcall SME_ModuleByMAC (SME_HANDLE handle, char *mac)

  Advances the internal pointer to the first discovered module whose MAC address matches the parameter.
- int __stdcall SME_ModuleCount (SME_HANDLE handle)

  Returns the number of modules discovered on the last call to SME_SearchForModules.
- int __stdcall SME_GetName (SME_HANDLE handle, char *moduleName)

  Retrieve the currently selected Sealevel I/O module's name.
- int __stdcall SME_GetMACAddress (SME_HANDLE handle, char *address)

  Retrieve the currently selected Sealevel I/O module's hardware MAC address.
- int __stdcall SME_GetNetworkConfig (SME_HANDLE handle, char *ipAddress, char *netmask, char *gateway)

  Retrieve's the current Sealevel I/O module's IP address, netmask, and gateway as strings.
- int __stdcall SME_GetNetworkConfigBytes (SME_HANDLE handle, unsigned char *ipAddress, unsigned char *netmask, unsigned char *gateway)

  Retrieve's the current module's IP address, netmask, and gateway as arrays of four bytes.
- int __stdcall SME_GetWirelessConfig (SME_HANDLE handle, int *network, char *SSID, int *channel, int *security, int *keyType)

  Gets the wireless module's configuration.
- int __stdcall SME_SetWirelessConfig (SME_HANDLE handle, int network, char *SSID, int channel, int security, int keyType, char *key)

  Configures a wireless enabled device's network settings.
- int __stdcall SME_IsReachable (SME_HANDLE handle)

  Sets the internal pointer to the first discovered module.
- int __stdcall SME_GetDHCPConfig (SME_HANDLE handle, int *status)

  Gets the currently selected Sealevel I/O module's DHCP status (on or off).
- int __stdcall SME_GetType (SME_HANDLE handle, char *type)

  Gets the current Sealevel I/O module's interface type.
- int __stdcall SME_SetName (SME_HANDLE handle, char *name)

  Sets the currently selected Sealevel I/O module's name.
- int __stdcall SME_SetNetworkConfig (SME_HANDLE handle, char *ipAddress, char *netmask, char *gateway)

Sets the currently selected Sealevel I/O module's network configuration.

- int __stdcall SME_SetNetworkConfigBytes (SME_HANDLE handle, unsigned char *ipAddress, unsigned char *netmask, unsigned char *gateway)

  Sets the currently selected module's network configuration.
- int __stdcall SME_SetDHCPConfig (SME_HANDLE handle, int status)

  Enables or disables the currently selected Sealevel I/O module's use of DHCP configuration.
- int __stdcall SME_SetNetworkSerialParams (SME_HANDLE handle, int baudrate, int parity)

  Sets the Ethernet to serial translation speed (serial output baudrate).
- int __stdcall SME_GetNetworkSerialParams (SME_HANDLE handle, int *baudrate, int *parity)

  Gets the Ethernet to serial translation speed (serial output baudrate).
- int __stdcall SME_ConfigureDeviceSecurity (SME_HANDLE handle, int securityMode, char *securityKey)

  Configures security mode for an Ethernet device.
- int __stdcall SME_RemoveDeviceSecurity (char *connection, int securityMode, char *securityKey)

  Disables security mode for an Ethernet device.

10.3.1 Detailed Description

The SeaMAX Ethernet Discover API includes functions used for configuration and discovery of Ethernet enabled Sealevel I/O devices. For specifics, click any of the function names below to view function use, parameter, and return code information.

10.3.2 Function Documentation

10.3.2.1 int __stdcall SME_Initialize (SME_HANDLE * *handle*)

Initializes the interface and allocates internal memory.

Parameters

out	*handle*	SME Integer handle required by all other SME functions.

Return values

0	Success.

SME_Initialize allocates internal memory and prepares the SME interface. After using the SME interface, it is necessary to explicitly call SME_Cleanup() to avoid memory leaks.

Warning

The SME handle returned by SME_Initialize is compatible with only the functions beginning with SME_. The SME API was written only to locate and configure Ethernet enabled Sealevel I/O modules. In order to read or write to a particular SeaI/O module, the standard SeaMAX SM_ functions should be used, after opening the module with SM_Open() and acquiring a SM handle.

10.3.2.2 int __stdcall SME_Cleanup (SME_HANDLE *handle*)

De-allocates memory used by the SME interface.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

0	Successful completion.
-1	Invalid SeaMAX handle.

10.3.2.3 int __stdcall SME_SearchForModules (SME_HANDLE *handle*)

Attempts to discover all reachable Sealevel I/O Ethernet modules via a UDP broadcast.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

>=0	Success. Number of found modules.
-1	Invalid SME_Initialize handle.
-2	Error acquiring a free socket.
-3	Error obtaining a list of local network interfaces.
-4	Error binding to socket.
-5	Error setting broadcast option.
-6	Error setting receive timeouts.
-7	Error sending broadcast message over interface.

Referenced by CCEthernet::find_devices().

10.3.2.4 int __stdcall SME_FirstModule (SME_HANDLE *handle*)

Sets the internal pointer to the first discovered module.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No modules have been found. See SME_SearchForModules().

10.3.2.5 int __stdcall SME_NextModule (SME_HANDLE *handle*)

Advances the internal pointer to the next discovered module.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No modules have been found. See SME_SearchForModules().
-3	End of discovered modules.

Referenced by CCEthernet::find_devices().

10.3.2.6 int __stdcall SME_ModuleByName (SME_HANDLE *handle*, char * *moduleName*)

Advances the internal pointer to the first discovered module whose name matches the parameter.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*moduleName*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No modules have been found. See SME_SearchForModules().
-3	Could not locate named module.
-4	Invalid name parameter.

Attempts to search for the first module whose name matches the 'moduleName' parameter. If SME_ModuleByName fails, the currently select module (if any) remains selected.

Note

The moduleName parameter should be between 0 and 16 characters, null terminated.

10.3.2.7 int __stdcall SME_Ping (SME_HANDLE *handle*)

Contacts the currently selected module to ensure it's powered and accessible.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

1	Module responded to request. Module is powered and accessible.
0	Module did not respond to request. Module may not be powered, may be unplugged, or misconfigured.
-1	Invalid SME_Initialize handle.

Because most of the SME functions work based off of an internal list compiled when SME_SearchForModules() is called, this function has been provided to check the real status of the currently selected module.

10.3.2.8 int __stdcall SME_ModuleByIP (SME_HANDLE *handle*, char * *ipAddress*)

Advances the internal pointer to the first discovered module whose IP address matches the parameter.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*ipAddress*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No modules have been found. See SME_SearchForModules().
-3	Could not locate named module.
-4	Invalid IP address parameter.

Attempts to search for the first module whose IP address matches the 'ipAddress' parameter. If SME_ModuleByIP fails, the currently select module (if any) remains selected.

Note

The ipAddress parameter should be null terminated.

Referenced by CCEthernet::set_information().

10.3.2.9 int __stdcall SME_ModuleByMAC (SME_HANDLE *handle*, char * *mac*)

Advances the internal pointer to the first discovered module whose MAC address matches the parameter.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*mac*	In the form "xx-xx-xx-xx-xx-xx"

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No modules have been found. See SME_SearchForModules().
-3	Could not locate named module.
-4	Invalid MAC address parameter.

Attempts to search for the first module whose MAC address matches the 'mac' string parameter. If SME_ModuleByMAC fails, the currently select module (if any) remains selected.

Note

The mac parameter should be null terminated.

10.3.2.10 int __stdcall SME_ModuleCount (SME_HANDLE *handle*)

Returns the number of modules discovered on the last call to SME_SearchForModules.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().

Return values

>=0	Number of Sealevel I/O Ethernet enabled modules found.
-1	Invalid SME_Initialize handle.

10.3.2.11 int __stdcall SME_GetName (SME_HANDLE *handle*, char * *moduleName*)

Retrieve the currently selected Sealevel I/O module's name.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*moduleName*	Buffer in which to store the name.

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	Invalid module selected. Call SME_SearchForModules() first.

Warning

The parameter moduleName should have at least 16 bytes of space allocated before calling SME_GetName.

Referenced by CCEthernet::find_devices().

10.3.2.12 int __stdcall SME_GetMACAddress (SME_HANDLE *handle*, char * *address*)

Retrieve the currently selected Sealevel I/O module's hardware MAC address.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*address*	Buffer in which to store the MAC address.

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	Invalid module selected. Call SME_SearchForModules() first.
-3	Address parameter may not be zero (null).

The returned string will be in the format of "XX-XX-XX-XX-XX-XX" where the XX's represent two-byte hexadecimal representations of the six MAC address octets.

Warning

The buffer to store the MAC address much contain at least 18 bytes of allocated space.

Referenced by CCEthernet::find_devices().

10.3.2.13 int __stdcall SME_GetNetworkConfig (SME_HANDLE *handle*, char * *ipAddress*, char * *netmask*, char * *gateway*)

Retrieve's the current Sealevel I/O module's IP address, netmask, and gateway as strings.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*ipAddress*	
out	*netmask*	
out	*gateway*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.

The currently selected Sealevel I/O module's IP address, netmask, and gateway will be placed in the parameter buffers in standard IP form (i.e. "x.x.x.x").

Note

If any parameter is NULL, that parameter is ignored. For instance, to retrieve only the current module's netmask, supply a valid string pointer for the second parameter, and NULL for the first and third parameters.

Warning

The supplied string buffers should each be allocated with at least 16 bytes of data before calling GetIPAddress().

Referenced by CCEthernet::find_devices().

10.3.2.14 int __stdcall SME_GetNetworkConfigBytes (SME_HANDLE *handle,* unsigned char * *ipAddress,* unsigned char * *netmask,* unsigned char * *gateway*)

Retrieve's the current module's IP address, netmask, and gateway as arrays of four bytes.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*ipAddress*	
out	*netmask*	
out	*gateway*	

Return values

0	Success.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SearchForModules() first.

The currently selected module's IP address, netmask, and gateway will be placed in the parameter buffers in the form {a,b,c,d}. Instead of populating the three parameters with string versions of the network configuration, the three parameters will each contain four byte equivalents of the network configuration.

Note

If any parameter is NULL, that parameter is ignored. For instance, to retrieve only the current module's netmask, supply a valid byte buffer pointer for the second parameter, and NULL for the first and third parameters.

Warning

The supplied byte buffers should each be allocated with at least 4 bytes of data before calling GetIPAddress().

10.3.2.15 int __stdcall SME_GetWirelessConfig (SME_HANDLE *handle*, int * *network*, char * *SSID*, int * *channel*, int * *security*, int * *keyType*)

Gets the wireless module's configuration.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*network*	Type of network connection
out	*SSID*	Network name
out	*channel*	Ad hoc channel number. If not an ad-hoc connection, channel is ignored
out	*security*	Type of network security enabled.
out	*keyType*	Key type used.

Return values

0	Success.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Could not get the module's wireless configuration.

Retrieves the current wireless configuration. If the wireless device exhibits a configuration other than those specified in this documentation, then the security type of SECURITY_UNKNOWN will be used.

Warning

SSID should be pre-allocated with 33 bytes of space before calling this function.

See also

Network Types, Security Type, Wireless Key Type

10.3.2.16 int __stdcall SME_SetWirelessConfig (SME_HANDLE *handle*, int *network*, char * *SSID*, int *channel*, int *security*, int *keyType*, char * *key*)

Configures a wireless enabled device's network settings.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*network*	Type of network connection
in	*SSID*	Network ID.
in	*channel*	Only valid for ad-hoc networks. Use zero otherwise.
in	*security*	Type of network security to enable
in	*keyType*	Type of key provided
in	*key*	Encryption key

Return values

0	Success.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Invalid network parameter. See Network Types.
-4	Invalid SSID. SSID must be null-terminated and must not exceed 32 characters.
-5	Invalid channel number. Channel should be either zero (infrastructure), or between 1 and 14.

-6	Invalid security type. See Security Type.
-7	Invalid key type. See Wireless Key Type.
-8	Invalid key. Key must be null-terminated, contain at least one character, and must not exceed 64 characters.
-9	Invalid security type for ad-hoc. Use NONE or a WEP type encryption only.
-10	The supplied security configuration does not allow the use of hex keys (passphrase only).
-11	Invalid key. 64-bit hex keys require 10 characters; 128-bit hex keys require 26 characters.
-12	Hexadecimal key contains non-hexadecimal characters.
-13	Could not set device's wireless configuration due to network error.
-14	Invalid SSID parameter.

Configures a wireless device's network settings and configuration. Following a successful upload of the new wireless configuration, the wireless hardware will momentarily reboot and will therefore be unavailable on the network for several seconds.

Note

Both of the SSID and key parameters must be null terminated strings.

Warning

Setting the wireless configuration causes the module to reboot momentarily.

See also

Network Types, Security Type, Wireless Key Type

10.3.2.17 int __stdcall SME_IsReachable (SME_HANDLE *handle*)

Sets the internal pointer to the first discovered module.

Return values

1	Module is reachable (on a valid subnet).
0	Module is not reachable.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.

10.3.2.18 int __stdcall SME_GetDHCPConfig (SME_HANDLE *handle,* int * *status*)

Gets the currently selected Sealevel I/O module's DHCP status (on or off).

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*status*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Invalid parameter.

If DHCP is enabled for the module, then status will be returned as non-zero.

Referenced by CCEthernet::find_devices().

10.3.2.19 int __stdcall SME_GetType (SME_HANDLE *handle*, char * *type*)

Gets the current Sealevel I/O module's interface type.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*type*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Invalid parameter.

Currently, there are only two types of Ethernet enabled SeaI/O modules, wired and wireless. SME_GetType() will return 'X' for wired SeaI/O modules and 'W' for wireless.

Referenced by CCEthernet::find_devices().

10.3.2.20 int __stdcall SME_SetName (SME_HANDLE *handle*, char * *name*)

Sets the currently selected Sealevel I/O module's name.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*name*	Null terminated name buffer

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	Supplied name is invalid. Name should be between 0 and 16 characters and NULL terminated.
-3	No module selected. Call SME_SearchForModules() first.
-4	Error acquiring free socket.
-5	Error connecting to module.
-6	Error getting current configuration.
-7	Error writing new configuration.

Referenced by CCEthernet::set_information().

10.3.2.21 int __stdcall SME_SetNetworkConfig (SME_HANDLE *handle,* char * *ipAddress,* char * *netmask,* char * *gateway*)

Sets the currently selected Sealevel I/O module's network configuration.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*ipAddress*	
in	*netmask*	
in	*gateway*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Invalid IP address string format. String should be in dotted notation.
-4	Invalid netmask string format. String should be in dotted notation.
-5	Invalid gateway string format. String should be in dotted notation.
-6	Invalid netmask.
-7	Invalid IP address. IP address must not be a home or broadcast address.
-8	Error getting current configuration.
-9	Error writing new configuration.

Any of the three parameters may be NULL if that part of the network configuration should not be changed. For instance, it is possible to only adjust the netmask of the currently selected module by supplying a valid second parameter, and NULL for the first and third parameters. Likewise, it is possible to set the IP address and netmask, without affecting the gateway.

Note

Setting a network configuration will automatically disable DHCP configuration, if it has been enabled. Calling SM_SetDHCPConfig() prior to this function is not required.
This function will delete the internal list of found modules and reset the internal current module pointer, resulting in "No module selected" errors on subsequent calls to SME_ functions. SME_SearchForModules() should be called immediately after calling this function if you wish to find or configure other modules.

Warning

Care should be taken when setting the IP address and netmask. This method attempts to deny invalid combinations such as home and broadcast addresses, but it does not check for out of subnet addresses.
This function will cause the Ethernet to serial bridge to reset for a short time while the new parameters are enabled. For bridges with statically assigned IP addresses, the reset time is only momentary. However, for DHCP assigned IP addresses, the bridge will not respond until it's IP address is re-leased from the DHCP server.

Referenced by CCEthernet::set_information().

10.3.2.22 int __stdcall SME_SetNetworkConfigBytes (SME_HANDLE *handle,* unsigned char * *ipAddress,* unsigned char * *netmask,* unsigned char * *gateway*)

Sets the currently selected module's network configuration.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*ipAddress*	
in	*netmask*	
in	*gateway*	

Return values

0	Success.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SearchForModules() first.
-3	Invalid netmask.
-4	Invalid IP address. IP address must not be a home or broadcast address.
-5	Error getting current configuration.
-6	Error writing new configuration.

Any of the three parameters may be NULL if that part of the network configuration should not be changed. For instance, it is possible to only adjust the netmask of the currently selected module by supplying a valid second parameter, and NULL for the first and third parameters. Likewise, it is possible to set the IP address and netmask, without affecting the gateway.

Instead of populating the three parameters with string versions of the network configuration, the three parameters should each contain four byte equivalents of the network configuration.

Note

Setting a network configuration will automatically disable DHCP configuration, if it has been enabled. Calling SM_SetDHCPConfig() prior to this function is not required.
This function will delete the internal list of found modules and reset the internal current module pointer, resulting in "No module selected" errors on subsequent calls to SME_ functions. SME_SearchForModules() should be called immediately after calling this function if you wish to find or configure other modules.

Warning

Care should be taken when setting the IP address and netmask. This method attempts to deny invalid combinations such as home and broadcast addresses, but it does not check for out of subnet addresses.
This function will cause the Ethernet to serial bridge to reset for a short time while the new parameters are enabled. For bridges with statically assigned IP addresses, the reset time is only momentary. However, for DHCP assigned IP addresses, the bridge will not respond until it's IP address is re-leased from the DHCP server.

10.3.2.23 int __stdcall SME_SetDHCPConfig (SME_HANDLE *handle,* int *status*)

Enables or disables the currently selected Sealevel I/O module's use of DHCP configuration.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*status*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Error getting current configuration.
-4	Error setting new configuration.

If the input parameter status is non-zero, DHCP configuration will be enabled for the selected module.

Warning

This function will cause the Ethernet to serial bridge to reset for a short time while the new parameters are enabled. For bridges with statically assigned IP addresses, the reset time is only momentary. However, for DHCP assigned IP addresses, the bridge will not respond until it's IP address is re-leased from the DHCP server.

Referenced by CCEthernet::set_information().

10.3.2.24 int __stdcall SME_SetNetworkSerialParams (SME_HANDLE *handle*, int *baudrate*, int *parity*)

Sets the Ethernet to serial translation speed (serial output baudrate).

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*baudrate*	
in	*parity*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Could not get the current module's serial parameters.
-4	Invalid baudrate parameter.
-5	Invalid parity parameter.
-6	Could not set the new serial parameters.

Note

The translation baudrate set here does not affect the rate at which any SeaI/O module sends or receives Modbus commands. It only affects the rate at which Ethernet data is translated from Modbus TCP to RTU, and back again. In order for a SeaI/O module to receive and respond to Modbus commands, the translation baudrate must match the SeaI/O's baudrate (see SM_SetCommunications()).
The Ethernet to serial bridge only supports baudrates 1200, 2400, 4800, 9600, 19200, 38400, and 115200 with parity values none, odd, or even. Any other values than these will result in an invalid parameter return value.

Warning

After a successful set of the serial parameters, this function will cause the Ethernet to serial bridge to reset for a short time while the new parameters are enabled. For bridges with statically assigned IP addresses, the reset time is only momentary. However, for DHCP assigned IP addresses, the bridge will not respond until it's IP address is re-leased from the DHCP server.

See also

Baudrate Values, Parity Values

10.3.2.25 int __stdcall SME_GetNetworkSerialParams (SME_HANDLE *handle*, int * *baudrate*, int * *parity*)

Gets the Ethernet to serial translation speed (serial output baudrate).

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
out	*baudrate*	
out	*parity*	

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	No module selected. Call SME_SearchForModules() first.
-3	Could not get the current module's serial parameters.

Retrieves the current setting of the Ethernet to serial bridge. A value of -1 returned in either parameter indicates an invalid or unknown value was returned.

Note

The Ethernet to serial bridge only supports baudrates 1200, 2400, 4800, 9600, 19200, 38400, and 115200 with parity values none, odd, or even.

See also

Baudrate Values, Parity Values

10.3.2.26 int __stdcall SME_ConfigureDeviceSecurity (SME_HANDLE *handle*, int *securityMode*, char * *securityKey*)

Configures security mode for an Ethernet device.

Parameters

in	*handle*	Valid handle returned by SME_Initialize().
in	*securityMode*	Type of encryption.
in	*securityKey*	Hexadecimal encryption key in ASCII format.

Return values

0	Successful completion.
-1	Invalid SME_Initialize handle.
-2	Invalid security mode.
-3	Invalid encryption key.
-4	No module selected. Call SME_SearchForModules() first.
-5	Could not set device's configuration. Possible network error.

For securityMode values, see Security Type.

The securityKey parameter must include a C String containing two hexadecimal characters per byte of encryption. For example, an AES 256 key would be represented as 64 hexadecimal characters: "0102030405060708090A0B0C0D0↩E0F10111213141516171819 1A1B1C1D1E1F20".

Warning

Devices with encryption enabled will not respond to broadcasts. Secured devices will not be discoverable with SME_SearchForModules() and will not be configurable with other SME methods. In order to reconfigure the device, first disable security on the device with SME_DisableDeviceSecurity().

10.3.2.27 int __stdcall SME_RemoveDeviceSecurity (char * *connection,* int *securityMode,* char * *securityKey*)

Disables security mode for an Ethernet device.

Parameters

in	*connection*	Valid IP Address of the device.
in	*securityMode*	Type of encryption.
in	*securityKey*	Hexadecimal encryption key in ASCII format.

Return values

-1	Parameter 'connection' is null.
-2	Invalid connection string.
-3	Ethernet: Could not resolve host address.
-4	Ethernet: Host refused or unavailable.
-5	Ethernet: Could not acquire free socket.
-6	Ethernet: Could not acquire a valid mutex.
-7	Ethernet: Unknown error establishing connection.
-8	Ethernet: Invalid key.
-9	Ethernet: Unable to load cbx_enc_2_1.dll.
-10	Ethernet: Unknown error when sending command to device.

For securityMode values, see Security Type.

The securityKey parameter must include a C String containing two hexadecimal characters per byte of encryption. For example, an AES 256 key would be represented as 64 hexadecimal characters: "0102030405060708090A0B0C0D0↩E0F10111213141516171819 1A1B1C1D1E1F20".

10.4 SeaDAC Lite Discovery API

Functions

- int __stdcall SDL_Initialize (SDL_HANDLE *handle)

  Initializes the interface and allocates internal memory.
- int __stdcall SDL_Cleanup (SDL_HANDLE handle)

  De-allocates memory used by the SDL interface.
- int __stdcall SDL_SearchForDevices (SDL_HANDLE handle)

  Attempts to enumerate all connected SeaDAC Lite modules and stores them as an internal list.
- int __stdcall SDL_DeviceCount (SDL_HANDLE handle)

  Returns the number of discovered SeaDAC Lite devices presently on the system.
- int __stdcall SDL_FirstDevice (SDL_HANDLE handle)

  Sets the internal pointer to the first discovered SeaDAC Lite device.
- int __stdcall SDL_NextDevice (SDL_HANDLE handle)

  Advances the internal pointer to the next discovered SeaDAC Lite device.
- int __stdcall SDL_GetName (SDL_HANDLE handle, char *deviceName)

  Retrieve's the current SeaDAC Lite device's SeaMAX compatible name.
- int __stdcall SDL_GetModel (SDL_HANDLE handle, int *model)

  Retrieve's the current SeaDAC Lite device's model number.
- int __stdcall SDL_GetDeviceID (SDL_HANDLE handle, int *id)

  Retrieve's the current SeaDAC Lite device's USB ID.
- int __stdcall SDL_GetSerial (SDL_HANDLE handle, char *serialNumber)

  Retrieve the current SeaDAC Lite device's Serial Number.
- int __stdcall SDL_GetNameBySerial (SDL_HANDLE handle, char *serialNumber, char *deviceName)

  Searches current device list for the device denoted by a Serial Number.

10.4.1 Detailed Description

SeaDAC Lite modules are discovered using the following SeaDAC Lite Discovery API. For more details, click any of the function names below to learn more about function use, parameter, and return code information.

10.4.2 Function Documentation

10.4.2.1 int __stdcall SDL_Initialize (SDL_HANDLE * *handle*)

Initializes the interface and allocates internal memory.

Parameters

out	*handle*	SDL handle required by all other SDL functions.

Return values

0	Success.

SDL_Initialize allocates internal memory and prepares the SDL interface. After using the SDL interface, it is necessary to explicitly call SDL_Cleanup() to avoid memory leaks.

Warning

The SDL handle returned by SDL_Initialize is compatible with only the functions beginning with SDL_. The SDL API was written only to locate SeaDAC Lite modules. In order to read or write to a particular SeaDAC Lite module, the standard SeaMAX SM_ functions should be used, after opening the SeaDAC Lite module with SM_Open() and acquiring a SM handle.

10.4.2.2 int __stdcall SDL_Cleanup (SDL_HANDLE *handle*)

De-allocates memory used by the SDL interface.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().

Return values

0	Successful completion.
-1	Invalid SDL_Initialize handle.

10.4.2.3 int __stdcall SDL_SearchForDevices (SDL_HANDLE *handle*)

Attempts to enumerate all connected SeaDAC Lite modules and stores them as an internal list.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().

Return values

>=0	Success. Number of found modules.
-1	Invalid SDL_Initialize handle.
-2	Could not determine the number of connected USB devices.

10.4.2.4 int __stdcall SDL_DeviceCount (SDL_HANDLE *handle*)

Returns the number of discovered SeaDAC Lite devices presently on the system.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().

Return values

>=0	Success. Number of found devices by previous call to SDL_SearchForDevices().
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.

10.4.2.5 int __stdcall SDL_FirstDevice (SDL_HANDLE *handle*)

Sets the internal pointer to the first discovered SeaDAC Lite device.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().

Return values

0	Success.
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.

10.4.2.6 int __stdcall SDL_NextDevice (SDL_HANDLE *handle*)

Advances the internal pointer to the next discovered SeaDAC Lite device.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().

Return values

0	Success.
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.
-3	End of discovered SeaDAC Lite devices.

10.4.2.7 int __stdcall SDL_GetName (SDL_HANDLE *handle,* char * *deviceName*)

Retrieve's the current SeaDAC Lite device's SeaMAX compatible name.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().
out	*deviceName*	

Return values

0	Success.
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.
-3	Parameter must not be zero.

The name returned by SDL_GetName is compatible with SM_Open() in the main SeaMAX API.

Note

SeaDAC Lite modules may not return consecutive numbers such as SeaDAC Lite 0, 1, 2, etc. It is not uncommon, depending on the number and type of SeaDAC Lite modules being used, to see numbering that 'skips' or is non-consecutive.

Warning

The parameter 'deviceName' should have at least 20 bytes of allocated space before calling SDL_GetName.

10.4.2.8 int __stdcall SDL_GetModel (SDL_HANDLE *handle*, int * *model*)

Retrieve's the current SeaDAC Lite device's model number.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().
out	*model*	

Return values

0	Success.
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.
-3	Parameter must not be zero.

10.4.2.9 int __stdcall SDL_GetDeviceID (SDL_HANDLE *handle*, int * *id*)

Retrieve's the current SeaDAC Lite device's USB ID.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().
out	*id*	

Return values

0	Success.
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. Call SDL_SearchForDevices() first.
-3	Parameter must not be zero.

10.4.2.10 int __stdcall SDL_GetSerial (SDL_HANDLE *handle*, char * *serialNumber*)

Retrieve the current SeaDAC Lite device's Serial Number.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().
out	*serialNumber*	

Return values

0	Success
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. See SearchForDevices().
-3	Parameter must not be zero.

Warning

The parameter 'serialNumber' should have at least 20 bytes of allocated space before calling SDL_GetSerial.

10.4.2.11 int __stdcall SDL_GetNameBySerial (SDL_HANDLE *handle*, char * *serialNumber*, char * *deviceName*)

Searches current device list for the device denoted by a Serial Number.

Parameters

in	*handle*	Valid handle returned by SDL_Initialize().
in	*serialNumber*	A valid character pointer
out	*deviceName*	

Return values

0	Success
-1	Invalid SDL_Initialize handle.
-2	No devices have been found. See SearchForDevices().
-3	serialNumber must not be zero.
-4	deviceName must not be zero.
-5	serialNumber not found in connected devices.

Warning

The parameter 'deviceName' should have at least 20 bytes of allocated space before calling SDL_GetNameBy↩ Serial.

10.5 Version 2.x Deprecated Interface

Modules

- Object-Oriented SeaMAX 2.x Interface
- Functional SeaMAX 2.x Interface
- CEthernet Interface

10.5.1 Detailed Description

A deprecated interface has been provided for backwards-compatability with older projects using SeaMAX version 2.x.

Note

This collection of deprecated functions is intended as a replacement for third party applications using SeaMAX v2. Sealevel can not guarantee that this API as a drop-in replacement, but will make every effort to provide support for applications upgrading from SeaMAX v2.

Because these functions are deprecated, Sealevel I/O products outside the SeaI/O module line (SeaI/O 410, 420, 430, 440, 450, 462, 463, 470, and 520) may not be supported by these functions. If you own Sealevel I/O products that are not covered by this deprecated API, please see the standard SeaMAX API for information on how to access those products.

Warning

These functions should not be used in new projects, as they scheduled for removal from future versions of the SeaMAX API. They have been added for backward support only.

10.6 Object-Oriented SeaMAX 2.x Interface

Functions

- SM_HANDLE CSeaMaxW32::GetSeaMAXVersion3Handle ()

  Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SM functions.
- int CSeaMaxW32::Read (slave_address_t slaveId, seaio_type_t type, address_loc_t address, address_range_t range, void ∗values)

  Performs a Modbus read on an opened SeaI/O module and returns the data by reference.
- int CSeaMaxW32::Write (slave_address_t slaveId, seaio_type_t type, address_loc_t address, address_range_t range, unsigned char ∗data)

  Performs a Modbus write on an opened SeaI/O module and returns the data by reference.
- int CSeaMaxW32::Open (char ∗file)

  Opens the specified resource.
- int CSeaMaxW32::Close ()

  Closes a previously opened SeaI/O module resource.
- HANDLE CSeaMaxW32::getCommHandle (void)

  Returns the HANDLE type data used by SeaMAX as the communications resource.
- int CSeaMaxW32::Ioctl (slave_address_t, IOCTL_t, void ∗)

  Multi-function tool to configure and access non-standard Modbus features of the SeaI/O module.
- int CSeaMaxW32::set_intermessage_delay (int new_delay_time)

  Sets the intermessage delay time.

10.6.1 Detailed Description

A deprecated version of the SeaMAX 2 interface has been provided for backwards compatability.

Warning

These functions should not be used in new projects, as they scheduled for removal from future versions of the SeaMAX API. They have been added for backward support only.

10.6.2 Function Documentation

10.6.2.1 SM_HANDLE CSeaMaxW32::GetSeaMAXVersion3Handle ( )

Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SM functions.

Deprecated Version 2.x - Version 5.0

Return values

>0	Success - the SeaMAX version 3.x compatable handle for use with non-deprecated functions.

0	No SeaMAX handle available - see CSeaMaxW32::Open()

This deprecated function was not originally part of SeaMAX version 2.x, but a later addition to provide support for newer features of SeaMAX version 3.x. The handle returned can be used by any of SeaMAX version 3.x SM_ functions, but should not be used for any deprecated SeaMAX version 2.x functions.

Warning

The SeaMAX handle returned by this function is bound by the CSeaMaxW32 object and it's scope. Therefore, any attempt to use the handle returned by this method after destroying the CSeaMaxW32 object will result in unforsable errors.

10.6.2.2 int CSeaMaxW32::Read (slave_address_t *slave_id,* seaio_type_t *type,* address_loc_t *starting_address,* address_range_t *range,* void * *data*)

Performs a Modbus read on an opened SeaI/O module and returns the data by reference.

Deprecated Version 2.x - Version 5.0

Parameters

in	*slave_id*	Slave ID of the target module
in	*type*	Type of read to complete
in	*starting_address*	Address to begin reading
in	*range*	Number of I/O points to read
out	*data*	Data pointer

Return values

>=0	Success - the number of valid data bytes in the buffer.
-14	A Modbus read exception has occured - the first byte of the data buffer contains the exception.
-22	A null pointer has been supplied where a pointer to a data buffer was expected.
-9	The specified communication is not current established.
-19	The SeaI/O module did not respond.

Read performs a Modbus read on an opened SeaI/O module and returns the data by reference.

The actual Modbus read type is specified by the 'type' argument. Since Modbus is one-indexed, the starting location should be no less than one. Likewise, the range indicates how many consecutive inputs should be read.

The final parameter, 'data', is a pointer to a buffer in which to store the returned data - most often, an array of bytes. Consult the provided code examples for model specific information.

Note

The following are deviations from SeaMAX v2:

- The SETUPREG and COMMREG options are not supported and will return -1.

References SM_GetConfig(), SM_GetPIODirection(), SM_ReadCoils(), SM_ReadDiscreteInputs(), SM_ReadHolding↩Registers(), SM_ReadInputRegisters(), SM_ReadPIO(), and SM_SelectDevice().

10.6.2.3 int CSeaMaxW32::Write (slave_address_t *slave_id,* seaio_type_t *type,* address_loc_t *starting_address,* address_range_t *range,* unsigned char * *data*)

Performs a Modbus write on an opened SeaI/O module and returns the data by reference.

Deprecated Version 2.x - Version 5.0

Parameters

in	*slave_id*	Slave ID of the target module
in	*type*	Type of read to complete
in	*starting_address*	Address to begin reading
in	*range*	Number of I/O points to read
out	*data*	Data pointer

Return values

>=0	Success - the number of valid data bytes in the buffer.
-14	A Modbus read exception has occured - the first byte of the data buffer contains the exception.
-22	A null pointer has been supplied where a pointer to a data buffer was expected.
-9	The specified communication is not current established.
-18	Can not write data to the outbound connection - connection error.
-19	The SeaI/O module did not respond.
-34	Attempting to write to more IO points than are available.
-27	Illegal data value.

Write performs a Modbus write on an opened SeaI/O module and returns the data by reference.

The actual Modbus write type is specified by the 'type' argument. Since Modbus is actually base one, the starting location should be no less than one. Likewise, the range indicates how many consecutive inputs should be read.

The final parameter, 'data', is a pointer to a buffer in which to store the returned data - specifically, an array of bytes. Consult the provided code examples for model specific information.

References SM_GlobalCommsReset(), SM_ReadInputRegisters(), SM_SelectDevice(), SM_WriteCoils(), SM_Write↩HoldingRegisters(), and SM_WritePIO().

10.6.2.4 int CSeaMaxW32::Open (char * *filename*)

Opens the specified resource.

Deprecated Version 2.x - Version 5.0

Parameters

in	*filename*	

Return values

0	Success.
-38	The length of the filename is greater than 256 characters.
-22	The specified protocol is invalid, or, the module could not be contacted.
-9	The specified location is invalid.

Opens the specified resource. The filename parameter can be one of two different types, Modbus RTU or Modbus TCP.

To specifiy a Modbus RTU connection (used for all non-Ethernet enabled modules) the string "sealevel_rtu://COMx" should be used, where x is a valid COM port number.

To specify a Modbus TCP connection (used only for Ethernet enabled modules) the string "sealevel_tcp://x.x.x.x" should be used, where x's represent the valid IP address of the module to be addressed.

References SM_Open().

10.6.2.5 int CSeaMaxW32::Close ( )

Closes a previously opened SeaI/O module resource.

Deprecated Version 2.x - Version 5.0

Return values

0	Success.

References SM_Close().

10.6.2.6 HANDLE CSeaMaxW32::getCommHandle (void)

Returns the HANDLE type data used by SeaMAX as the communications resource.

Deprecated Version 2.x - Version 5.0

10.6.2.7 int CSeaMaxW32::Ioctl (slave_address_t *slave_id*, IOCTL_t *which*, void * *data*)

Multi-function tool to configure and access non-standard Modbus features of the SeaI/O module.

Deprecated Version 2.x - Version 5.0

Parameters

in	*slave_id*	Slave ID of the module
in	*which*	IOCTL function to perform
in,out	*data*	Appropriate data structure (depending on IOCTL function)

Return values

>=0	Success.
-22	A null pointer has been supplied where a pointer to a data buffer was expected.
(*)	Also valid are return values for CSeaMaxW32::Read() and CSeaMaxW32::Write().

Ioctl has been provided as a multi-function tool to configure and access non-standard Modbus features of the SeaI/O module.

The actual Modbus transaction is specified by the 'which' argument. Ioctl can be used to set module parameters, configure state, and access data. Consult the provided example source for model specific information.

The final parameter, data, is a pointer to a datatype which is appropriate for the particular IOCTL call.

Note

The following are deviations from SeaMAX v2:

- An invalid IOCTL will return -1.
- For IOCTL_READ_COMM_PARAM, the ioctl_struct will not be populated with a magic cookie value.

References SM_GetADDAConfig(), SM_GetADDAExtendedConfig(), SM_GetDeviceConfig(), SM_GetPIODirection(), SM_SelectDevice(), SM_SetADDAConfig(), SM_SetCommunications(), SM_SetPIODirection(), and SM_SetSoftware↩Address().

10.6.2.8 int CSeaMaxW32::set_intermessage_delay (int *delay*)

Sets the intermessage delay time.

Deprecated Version 2.x - Version 5.0

Parameters

in	*delay*	

Return values

>=0	Success.

This function is used to set the intermessage delay time in effort to optimize the number of messages that can be sent. This time must be at least 4 character times long. The default is a function of the baud rate and should not be changed unless necessary.

The parameter 'delay' is the delay (in milliseconds) to wait between sending back to back messages.

10.7 Functional SeaMAX 2.x Interface

Functions

- long SeaMaxW32Create ()

  Creates an instance of the SeaMAX object in the global memory space.
- SM_HANDLE SeaMaxW32GetVersion3Handle (CSeaMaxW32 ∗SeaMaxPointer)

  Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SM functions.
- long SeaMaxW32Destroy (CSeaMaxW32 ∗p)

  Releases the globally allocated SeaMAX object.
- long SeaMaxW32Open (CSeaMaxW32 ∗SeaMaxPointer, char ∗filename)
- long SeaMaxW32Read (CSeaMaxW32 ∗SeaMaxPointer, slave_address_t slaveId, seaio_type_t type, address↩ _loc_t starting_address, address_range_t range, void ∗data)
- long SeaMaxW32Write (CSeaMaxW32 ∗SeaMaxPointer, slave_address_t slaveId, seaio_type_t type, address↩ _loc_t starting_address, address_range_t range, unsigned char ∗data)
- long SeaMaxW32Ioctl (CSeaMaxW32 ∗SeaMaxPointer, slave_address_t slaveId, IOCTL_t which, void ∗data)
- long SeaMaxW32Close (CSeaMaxW32 ∗SeaMaxPointer)

10.7.1 Detailed Description

A deprecated version of the SeaMAX 2 interface has been provided for backwards compatability.

Warning

These functions should not be used in new projects, as they scheduled for removal from future versions of the SeaMAX API. They have been added for backward support only.

10.7.2 Function Documentation

10.7.2.1 long SeaMaxW32Create ( )

Creates an instance of the SeaMAX object in the global memory space.

Deprecated Version 2.x - Version 5.0

Return values

>0	Success - a long pointer to the SeaMAX object.

10.7.2.2 SM_HANDLE SeaMaxW32GetVersion3Handle (CSeaMaxW32 ∗ *SeaMaxPointer*)

Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SM functions.

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	

Return values

>0	Success - the SeaMAX version 3.x compatable handle for use with non-deprecated functions.
0	SeaMaxPointer may not be null.

This deprecated function was not originally part of SeaMAX version 2.x, but a later addition to provide support for newer features of SeaMAX version 3.x. The handle returned can be used by any of SeaMAX version 3.x SM_ functions, but should not be used for any deprecated SeaMAX version 2.x functions.

Warning

The SeaMAX handle returned by this function will no longer be valid after calling SeaMaxW32Destroy(). Therefore, any attempt to use the handle returned by this method after destroying the SeaMAX object will result in unforsable errors.

10.7.2.3 long SeaMaxW32Destroy (CSeaMaxW32 * *p*)

Releases the globally allocated SeaMAX object.

Deprecated Version 2.x - Version 5.0

Return values

0	Success.

10.7.2.4 long SeaMaxW32Open (CSeaMaxW32 * *SeaMaxPointer,* char * *filename*)

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	
in	*filename*	

See also

CSeaMaxW32::Open()

10.7.2.5 long SeaMaxW32Read (CSeaMaxW32 * *SeaMaxPointer,* slave_address_t *slaveId,* seaio_type_t *type,* address_loc_t *starting_address,* address_range_t *range,* void * *data*)

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	
in	*slaveId*	
in	*type*	
in	*starting_address*	
in	*range*	
in,out	*data*	

See also

CSeaMaxW32::Read()

10.7.2.6 long SeaMaxW32Write (CSeaMaxW32 * *SeaMaxPointer*, slave_address_t *slaveId*, seaio_type_t *type*, address_loc_t *starting_address*, address_range_t *range*, unsigned char * *data*)

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	
in	*slaveId*	
in	*type*	
in	*starting_address*	
in	*range*	
in,out	*data*	

See also

CSeaMaxW32::Write()

10.7.2.7 long SeaMaxW32Ioctl (CSeaMaxW32 * *SeaMaxPointer*, slave_address_t *slaveId*, IOCTL_t *which*, void * *data*)

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	
in	*slaveId*	
in	*which*	
in,out	*data*	

See also

CSeaMaxW32::Ioctl()

10.7.2.8 long SeaMaxW32Close (CSeaMaxW32 * *SeaMaxPointer*)

Deprecated Version 2.x - Version 5.0

Parameters

in	*SeaMaxPointer*	

See also

CSeaMaxW32::Close()

10.8 CEthernet Interface

Functions

- ceth_device * CCEthernet::Alloc (int number)

  Allocates a list of CEthernet devices to be populated using find_devices()
- void CCEthernet::Free (ceth_device *)

  Frees a previously allocated list.
- SME_HANDLE CCEthernet::GetSeaMAXVersion3Handle ()

  Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SME functions.
- int CCEthernet::find_devices (ceth_device_type type_to_find, int number_to_find, ceth_device_p list_to_store_↩ devices)

  Locates all SeaIO Ethernet modules on the local subnet.
- int CCEthernet::set_information (ceth_device *device, ceth_set_types command,...)

  Sets the communications parameters for a particular module.

10.8.1 Detailed Description

A deprecated version of the CEthernet interface has been provided for backwards compatability.

Warning

These functions should not be used in new projects, as they scheduled for removal from future versions of the SeaMAX API. They have been added for backward support only.

10.8.2 Function Documentation

10.8.2.1 ceth_device * CCEthernet::Alloc (int *number*)

Allocates a list of CEthernet devices to be populated using find_devices()

Deprecated Version 2.x - Version 5.0

Parameters

in	*number*	Number of devices to allocate in the list

Return values

>0	Success. Integer pointer to a valid device list.
-ENOMEM	Error allocating memory.

10.8.2.2 void CCEthernet::Free (ceth_device * *list*)

Frees a previously allocated list.

Deprecated Version 2.x - Version 5.0

Parameters

in	*list*	Integer pointer to the list allocaed using Alloc()

10.8.2.3 SME_HANDLE CCEthernet::GetSeaMAXVersion3Handle ( )

Returns a SeaMAX Version 3.x compatable handle for use with non-deprecated (newer) SME functions.

Deprecated Version 2.x - Version 5.0

Return values

>0	Success - the SeaMAX version 3.x compatable handle for use with non-deprecated functions.
0	No SeaMAX handle available - see CCEthernet::Open()

This deprecated function was not originally part of SeaMAX version 2.x, but a later addition to provide support for newer features of SeaMAX version 3.x. The handle returned can be used by any of SeaMAX version 3.x SM_ functions, but should not be used for any deprecated SeaMAX version 2.x functions.

Warning

The SeaMAX handle returned by this function is bound by the CCEthernet object and it's scope. Therefore, any attempt to use the handle returned by this method after destroying the CCEthernet object will result in unforsable errors.

10.8.2.4 int CCEthernet::find_devices (ceth_device_type *type_to_find,* int *number_to_find,* ceth_device_p *list_to_store_devices*)

Locates all SeaIO Ethernet modules on the local subnet.

Deprecated Version 2.x - Version 5.0

Parameters

in	*type_to_find*	Can be only be the values SeaIO_Ethernet or Sealevel_All_Devices.
in	*number_to_find*	Maximum number of devices to find.
in,out	*list_to_store_devices*	Integer pointer provided by Alloc().

Return values

>0	Number of devices found.
-100XX	Socket error. Refer to Winsock documentation for error code.

Note

The number_to_find parameter must be less than or equal to the parameter provided to Alloc() when first creating the device list.

Warning

Alloc() must be explicitly called before calling find_devices().

Note

The following are deviations from SeaMAX v2:

- Ignores the type_to_find parameter and locates only Sealevel I/O devices (SeaIO_Ethernet).
- Returns (-10000 + SME_SearchForModules() result) on failure to search for modules.

References SME_GetDHCPConfig(), SME_GetMACAddress(), SME_GetName(), SME_GetNetworkConfig(), SME_GetType(), SME_NextModule(), and SME_SearchForModules().

10.8.2.5 int CCEthernet::set_information (ceth_device * *device*, ceth_set_types *command*, ...)

Sets the communications parameters for a particular module.

Deprecated Version 2.x - Version 5.0

Parameters

in	*device*	Device to modify the parameters of.
in	*command*	Which parameters to modify (i.e. SetIPAddress, SetDHCP, SetName).
in	...	This parameter list is based off of the command parameter above.

Return values

-100XX	Socket error. Refer to the Winsock documentation for error code.

The variable argument list is defined by the command parameter passed in. The commands can be logically OR'd together to make the sequence easier. First each command individually:

SetIPAddress - Parameters: New IP Address, New Netmask, New Gateway SetDHCP - Parameters: None SetName - Parameters: New Name

If the IP configuration (DHCP or Static) is logically OR'd with the SetName option, the parameters for the IP address must come first, and the last parameter will be the name.

References SME_ModuleByIP(), SME_SetDHCPConfig(), SME_SetName(), and SME_SetNetworkConfig().