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↩ : "0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20".

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.

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.

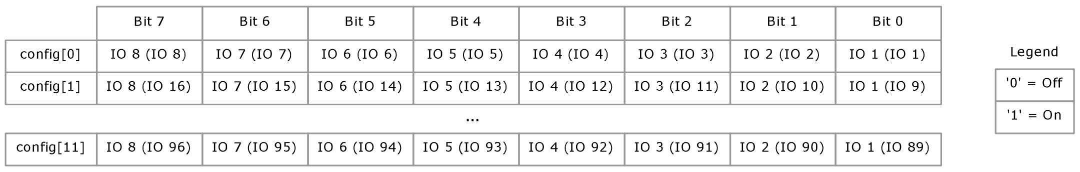
Return values

>=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 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 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

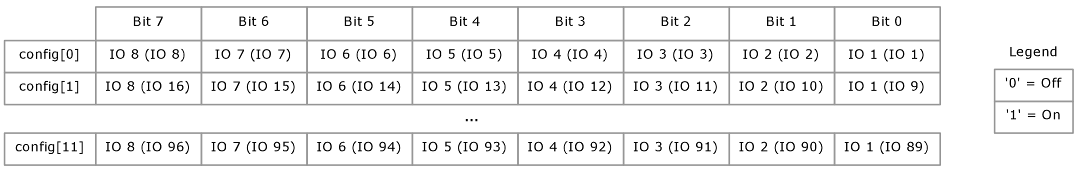
>0	Successful completion. Bytes successfully written.
-1	Invalid SeaMAX handle.
-2	Connection is not established. Check the provided Connection object state.
-3	Read error waiting for response. Unknown Modbus exception.
-4	Illegal Modbus Function (Modbus Exception 0x01).
-5	Illegal Data Address (Modbus 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 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 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 SealO 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.

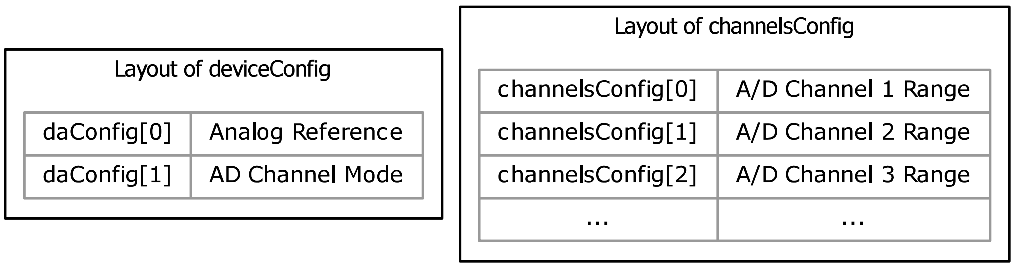
Return values

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.

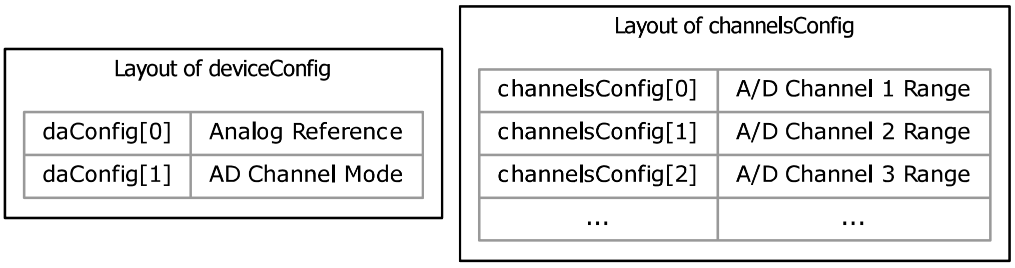
Return values

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_Write↩ HoldingRegisters().
`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)