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 Seal/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 Seal/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 Seal/O module to receive and respond to Modbus commands, the translation baudrate must match the Seal/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↩E0F101112131415161718191A1B1C1D1E1F20".

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↩
E0F101112131415161718191A1B1C1D1E1F20".