ALIEN TECHNOLOGY®

ALH-900x/901x DEVELOPER'S GUIDE

December, 2013

ALH-9000
ALH-9001
ALH-9010
ALH-9011

Legal Notices

Copyright ©2013 Alien Technology Corporation. All rights reserved.
Alien Technology Corporation has intellectual property rights relating to technology embodied in the products described in this document, including without limitation certain patents or patent pending applications in the U.S. or other countries.

This document and the products to which it pertains are distributed under licenses restricting their use, copying, distribution and decompilation. No part of this product documentation may be reproduced in any form or by any means without the prior written consent of Alien Technology Corporation and its licensors, if any. Third party software is copyrighted and licensed from Licensors. Alien, Alien Technology, the Alien logo, Nanoblock, Fluidic Self Assembly, FSA, Gen2Ready, Squiggle, Nanoscanner and other graphics, logos, and service names used in this document are trademarks of Alien Technology Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners. U.S. Government approval required when exporting the product described in this documentation.

Federal Acquisitions: Commercial Software -- Government Users Subject to Standard License Terms and Conditions. U.S. Government: If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARANTEES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGMENT ARE HEREBY DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Alien Technology®

ALH-900x/901x Developer's Guide

All Alien Handheld RFID Readers

Table of Contents

1 Introduction

The Alien ALH-900x/901x Software Developer's Kit (SDK) provides libraries and sample code for programmatically controlling Alien ALH-900x/901x handheld readers, using the Microsoft .NET Framework v2.0. Alien provides class libraries and sample applications to help you get up-and-running developing your custom applications to run directly on the device.

1.1 Audience

We assume that the readers of this guide:

- are proficient embedded Windows developers,
- have minimal previous knowledge of RFID, GPS, and other relevant technologies.

1.2 Type Conventions

- Regular text appears in a plain, sans-serif font.
- External files and documents appear in *italic text*.
- Class names appear in a `fixed-width serif font`.
- Things you type in, and sample code appear:

```
            indented, in a fixed-width serif font.
```

- Longer blocks of sample code appear:

```
within a shaded, outlined block
in a smaller, fixed-width serif font
```

1.3 Overview

This document focuses on controlling the various modules of the ALH-900x/901x handheld reader using the Alien .NET API supplied in the SDK. There are separate class libraries for the .NET Compact Framework for each hardware module. They are summarized in the following table by the Alien handheld model type.

.NET Library	Hardware Module	ALH-9000 (Windows CE)	ALH-9001 (Windows CE)	ALH-9010 (Windows Mobile)	ALH-9011 (Windows Mobile)
NRfidApi.dll	RFID Reader	✓	✓	✓	✓
NBarcodeApi.dll	1D/2D Barcode Scanner	✓ 1D	✓ 1D & 2D	✓ 1D	✓ 1D & 2D
NGpsApi.dll	GPS Receiver	-	✓	-	use Windows Mobile .NET system library
NCameraApi.dll	Built-in Camera	-	✓	-	use Windows Mobile .NET system library
NBluetoothApi.dll	Bluetooth	-	✓	-	use Windows Mobile .NET system library
NSysSvcApi.dll	System Services, including WLAN	✓	✓	✓	✓

1.4 Documentation and Sample Code

This Developer's Guide provides complete documentation of all API elements. Additional usage tips can be gleaned by examining and modifying the source code samples provided by Alien. Those sample applications, developed in the .NET environment (C#), each demonstrate most of the major features of each hardware module. The class libraries contained within the Alien API provide type structures and classes that constitute discrete functional groups for controlling various aspects of the handheld.

1.5 System Requirements

You will need Visual Studio 2005 or 2008 and .NET Compact Framework 2.0 in order to develop applications to run on Alien handhelds.

In order to interface directly with the handheld over USB, you will need Windows ActiveSync 3.7 or above. Windows 7 users do not need ActiveSync, since Windows Mobile Device Center handles that function.

The Windows Mobile Professional SDK is also required in order to develop software for ALH-9010/9011 Windows Mobile-based handhelds.

The following is a summary of system requirements for handheld software development.

- **Development Environment:** Visual Studio 2005 or 2008
- **.NET Compact Framework:** Version 2.0 or later
- **SDK (for ALH-9010/9011 only):** Windows Mobile Professional SDK 6.5 or later
- **ActiveSync (for Windows XP SP2 or earlier):** Version 3.7 or later

2 RFID

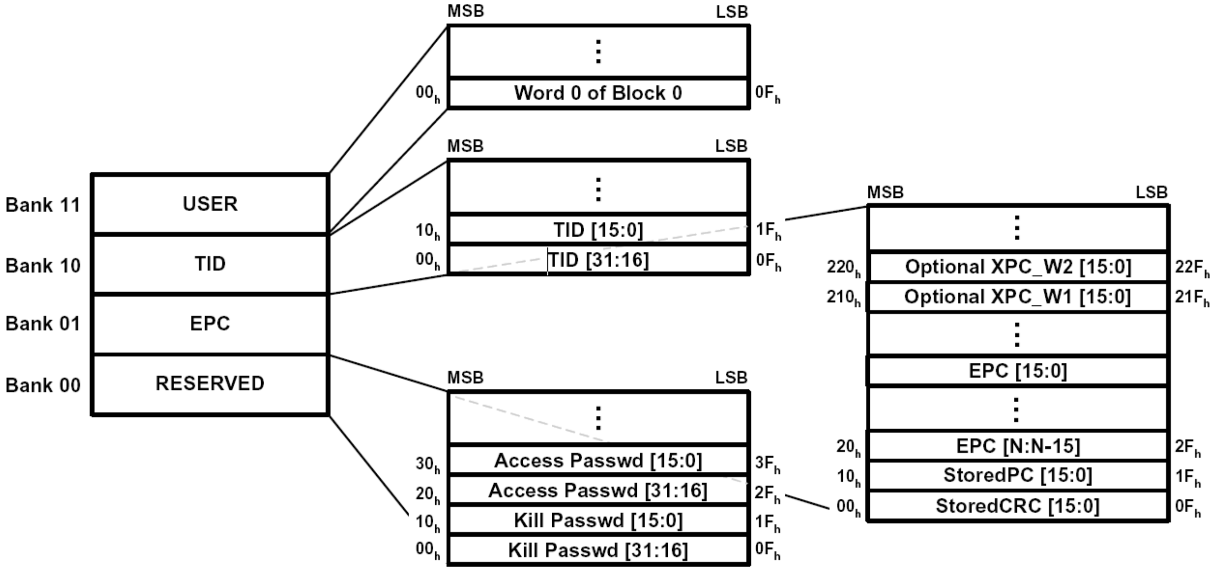
2.1 Introduction

The `NRfidApi.dll` library contains all of the classes you will need to control and communicate with the RFID module in your Alien handheld reader. You first create an RfidApi object, then use methods on it to power up and open a communication channel with the reader, and then send and receive commands. The API supports both synchronous (send-receive) communication, and asynchronous (send, wait for callback) communications.

You have control over most aspects of the Gen2 protocol, including access operations like write, lock, and kill, using tag masks, and Return Signal Strength Indication (RSSI) data on each tag read. The API can perform most operations continuously or one tag at a time.

To take full advantage of the RFID module and the Gen2 protocol, you are encouraged to read and understand the relevant portions of the EPCglobal Gen2 specification. One thing you must be aware of when accessing individual portions of memory in the tag is the layout of tag memory. As shown in the diagram below, all Gen 2 tags have four banks of memory (RESERVER, EPC, TID, USER) and each of those banks can be broken down into fields (Kill Password and Access Password within the RESERVED bank).

When reading and writing tag memory, all data operations are performed in one-word (2 bytes, 16 bits) increments, and only on word boundaries within a single bank. Some masking operations allow addressing memory down to the nibble (1/2 byte, 4 bits) or bit level.



2.2 RFID Result Codes

2.2.1 RFID_RESULT

Commands that return an immediate result (using SyncMode) will return one of the RFID_RESULT codes:

Type	Item	Description
RFID_RESULT	RFID_RESULT_SUCCESS	operation successfully performed
	RFID_RESULT_INVALIDARG	invalid parameter
	RFID_RESULT_OUTOFMEMORY	failed to assign a memory resource
	RFID_RESULT_UNEXPECTED	undefined error
	RFID_RESULT_FAILURE	failed to perform operation
	RFID_RESULT_ALREADY_OPENED	RFID device has already been opened
	RFID_RESULT_INVALID_DEVICE	RFID device not installed
	RFID_RESULT_NOT_CONNECTED	opened RFID device stopped working
	RFID_RESULT_NOT_OPENED	function called with calling Open first
	RFID_RESULT_NOT_DETECT	no tag detected
	RFID_RESULT_ACCESS_ERROR	incorrect access password, or nonexistent memory
	RFID_RESULT_COMMAND_ERROR	error executing a command, or received a command before finishing another
	RFID_RESULT_LOW_BATTERY	module failed, due to a low battery
	RFID_RESULT_UNKNOWN	unknown error
	RFID_RESULT_NOT_SUPPORTED	command currently not supported
	RFID_RESULT_STOPPED	
	RFID_RESULT_POWER_OFF	

2.2.2 RFIDCALLBACKTYPE_REPLY

When using the asynchronous (callback) method of accessing tags, the registered RfidCallbackProc function is called, with a data structure containing one of the following RFIDCALLBACKTYPE_REPLY values:

Type	Reply	Description
RFIDCALLBACKTYPE_REPLY	“Other Error”	
	“Memory Overrun”	Tried to access nonexistent memory.
	“Memory Locked”	Tag is locked.
	“Insufficient Tag Power”	Cannot write to the tag because it has insufficient power.
	“Non-specific Error”	
	“Check Antenna”	Cannot connect to antenna.
	“Try after cooling”	The RFID module is overheating.
	“Insufficient PDA Power”	Module won’t work because of low battery.
	“OK”	Function completed successfully.
	“Not Supported”	Current function is unsupported.
	“Not Connected”	Module failed after opening connection.
	“Not Opened”	Function called before Open().
	“Bad Access Password”	Used wrong tag access password.
	“Invalid Parameter”	Invalid parameter.
	“Command Error”	Error executing a command, or interrupted by another command.
	“Success”	Command successfully executed.
	“Not Detect”	No tag detected during operation.
	“Multi Read Stop”	Multi-read/continuous mode stopped.
	“EAS”	Detected EAS of NXP tag.

2.3 RFID Enumerations

2.3.1 MEM_BANK

The MEM_BANK enumeration is used to specify a particular memory bank, for locking, reading, and writing. See the Gen2 memory diagram at the start of this chapter for details of each bank.

```
public enum MEM_BANK{
  RESERVED,
  EPC,
  TID,
  USER
}
```

2.3.2 RFID_READ_TYPE

Designates whether an inventory operation should apply to a single tag, or many tags.

```
public enum RFID_READ_TYPE {
  EPC_GEN2_ONE_TAG,
  EPC_GEN2_MULTI_TAG
}
```

2.3.3 PERMALOCK_FIELD

Allows you to specify tag memory fields when permalocking.

```
public enum PERMALOCK_FIELD {
  ACCESS_PASSWORD,
  KILL_PASSWORD,
  EPC,
  TID,
  USER
}
```

2.3.4 RFID_CALLBACK_TYPE

When your callback function is executed, it passes you one of these RFID_CALLBACK_TYPEs.

```
public enum RFID_CALLBACK_TYPE {
  RFIDCALLBACKTYPE_DATA,
  RFIDCALLBACKTYPE_REPLY
}
```

- RFIDCALLBACKTYPE_DATA: contains requested memory data from the tag
- RFIDCALLBACKTYPE_REPLY: contains the results of the specific command just executed

2.3.5 INVENTORIED_STATE

Controls which Gen2 A/B state is used to inventory tags.

```
public enum INVENTORIED_STATE {
  STATE_A,
  STATE_B,
  STATE_AB
}
```

2.3.6 SESSION_TYPE

Controls which Gen2 session is used to inventory tags.

```
public enum SESSION_TYPE {
  SESSION_0,
  SESSION_1,
  SESSION_2,
  SESSION_3
}
```

2.3.7 HOPPING_MODE

The HOPPING_MODE enumeration is used to indicate which RF hop table is in use.

```
public enum HOPPING_MODE {
  ANONYMOUS,
  CHINA_FHSS,
  EURO_LBT,
  JAPAN_LBT,
  JAPAN_NO_LBT,
  KOREA_FHSS,
  KOREA_KCCh,
  KOREA_LBT,
  USA_FHSS
}
```

2.3.8 SELECT_ACTION

Used when specifying specific Gen2 actions to take during masking.

```
public enum SELECT_ACTION {
  ACTION_0
  ACTION_1
  ACTION_2
  ACTION_3
  ACTION_4
  ACTION_5
  ACTION_6
  ACTION_7
}
```

	If SELECT_TARGET is SL		If SELECT_TARGET are between S0 ~ S3	
	Matching	Non-matching	Matching	Non-matching
ACTION_0	Assert SL	De-assert SL	→ A	→ B
ACTION_1	Assert SL	No Action	→ A	No Action
ACTION_2	No Action	De-assert SL	No Action	→ B
ACTION_3	Negate SL	No Action	Negate	No Action
ACTION_4	De-assert SL	Assert SL	→ B	→ A
ACTION_5	De-assert SL	No Action	→ B	No Action
ACTION_6	No Action	Assert SL	No Action	→ A
ACTION_7	No Action	Negate SL	No Action	Negate

2.3.9 SELECT_TARGET

Set Target Flag of Tag which will be applied to the Select command.

```
public enum SELECT_TARGET {
  S0
  S1
  S2
  S3
  SL
}
```

- **S0**: Inventoried Flag of S0
- **S1**: Inventoried Flag of S1
- **S2**: Inventoried Flag of S2
- **S3**: Inventoried Flag of S3
- **SL**: Selected Flag

2.4 RFID Data Types

2.4.1 LOCKUNLOCKFIELD

Stores information that controls which fields/banks to lock or unlock in a tag.
Lock or unlock only applies to fields whose members in this struct are true.

```
public struct LOCKUNLOCKFIELD {
  bAccessPassword,
  bEPC,
  bKillPassword,
  bTID,
  bUSER
}
```

For example, in order to lock the Access password and EPC, but leave the other fields alone, set the structure like this:

```
LOCKUNLOCKFIELD lock   = new LOCKUNLOCKFIELD();
lock.bAccessPassword   = true;
lock.bEPC              = true;
lock.bKillPassword     = false;
lock.bTID              = false;
lock.bUSER             = false;
```

2.4.2 RFIDMASKPARAMS

The RFIDMASKPARAMS type defines a tag mask, including bank, offset, and data. You use masks to filter out desired tags from the general tag population. The RFIDMASKPARAMS is an optional argument for many of the tag access and inventory-related commands.

```
public struct RFIDMASKPARAMS {
  public MEM_BANK  MemBank;
  public uint      nOffSet
  public string    MaskPattern;
}
```

- **MemBank**: Memory Bank of tag which is used as Mask. (Refer to MEM_BANK)
- **nOffSet**: *nibble* offset into the selected memory bank.
- **MaskPattern**: mask pattern string.

For example, if a tag's TID bank is "E2006004015F325A", and you would like to use the "2006" portion as a mask, the structure would be:

```
RFIDMASKPARAMS Mask    = new RFIDMASKPARAMS();
Mask.MemBank           = MEM_BANK.TID;
Mask.nOffSet           = 1;
Mask.MaskPattern       = ”2006”;
```

Or, to mask on the "F325" portion:

```
RFIDMASKPARAMS Mask    = new RFIDMASKPARAMS();
Mask.MemBank           = MEM_BANK.TID;
Mask.nOffSet           = 11;
Mask.MaskPattern       = ”F325”;
```

2.4.3 RFIDMASKPARAMS_EX

The RFIDMASKPARAMS_EX type defines a tag mask, including bank, bit offset, number of bits to mask, and data. This version of the mask params includes bit-level control of the mask offset and length, and the mask data is supplied as a byte array, instead of a string. You use masks to filter out desired tags from the general tag population. The RFIDMASKPARAMS_EX is an optional argument for many of the tag access and inventory-related commands.

```
public struct RFIDMASKPARAMS_EX {
  public MEM_BANK  MemBank
  public uint      nOffSet
  public uint      nBits
  public byte[]    MaskPattern
}
```

- **MemBank**: Memory Bank of tag which is used as Mask. (Refer to MEM_BANK)
- **nOffSet**: *bit* offset into the selected memory bank
- **nBits**: number of bits of the MaskPattern which will be used as the masking pattern (0-255).
- **MaskPattern**: byte array to be used as the masking pattern.

2.4.4 RFIDSELMASKPARAMS_EX

The RFIDSELMASKPARAMS_EX type defines a tag mask, including the mask action, bank, bit offset, number of bits to mask, data, and Select Target. This version of the mask params includes bit-level control of the mask offset and length, and the mask data is supplied as a byte array (instead of a string), and you also have control over the mask action and session/SL target. You use masks to filter out desired tags from the general tag population. The RFIDSELMASKPARAMS_EX is an optional argument for many of the tag access and inventory-related commands.

The MaskPattern byte array must be an array of length `RfidApi.MAX_MASK_BYTES` (currently 32 bytes - big enough for any mask supported by the Gen2 protocol).

```
public struct RFIDSELMASKPARAMS_EX {
  public SELECT_ACTION ActionCode
  public uint          nBits
  public byte[]        MaskPattern
  public MEM_BANK      MemBank
  public uint          nOffSet
  public SELECT_TARGET SelectTarget
}
```

- **ActionCode**: The Gen2 action to apply during masking (Refer to SELECT_ACTION)
- **nBits**: number of bits of the MaskPattern which will be used as the masking pattern (0-255).
- **MaskPattern**: 32-byte array to be used as the masking pattern.
- **MemBank**: Memory Bank of tag which is used as Mask. (Refer to MEM_BANK)
- **nOffSet**: *bit* offset into the selected memory bank
- **SelectTarget**: The session or SL flag to target during masking Select command. (Refer to SELECT_TARGET)

2.4.5 RFIDCALLBACKDATA

When your callback delegate is called, you are given a RFIDCALLBACKDATA structure. Within the RFIDCALLBACKDATA is a RFID_CALLBACK_TYPE, which specifies the type of response. If the response type is data, you use the two additional parameters, wParam and lParam to fetch the actual data with a call to GetResult().

```
public struct RFIDCALLBACKDATA
{
  public RFID_CALLBACK_TYPE CallbackType;
  public IntPtr    wParam;
  public IntPtr    lParam;
}
```

- **CallbackType**: the reason for the callback – data or a result code
- **wParam**: the first parameter used to read response data from the module
- **lParam**: the second parameter used to read response data from module

2.4.6 RfidCallbackProc

This is the main RFID callback delegate that will receive and process data from the RFID module. You are provided a RFIDCALLBACKDATA structure, from which you determine the type of response and request any fetched data.

```
public delegate void RfidCallbackProc(RFIDCALLBACKDATA CallbackData);
```

2.5 RFID Methods

The Alien SDK provides APIs which can control the RFID reader. Using the RFID API, the application sets up the connection with RFID radio module, adjusts radio parameters, and executes tag operations (inventory, read, write, kill, lock) according to the ISO 18000 (EPC Class 1 Generation 2) tag protocol.

2.5.1 RfidApi()

```
RfidApi rfid = new RfidApi();
```

This is the constructor method for the main RFID class. All subsequent API operations are performed by calling the methods of the RfidApi object.

Parameters:
None

Returns:
None

2.5.2 PowerOn()

```
RFID_RESULT  PowerOn();
```

Turns on and initializes the RFID RFID reader module. This should be called once, at the beginning of your program.

Parameters:
None

Returns:
`RFID_RESULT_SUCCESS` if the module initialization performed normally, or was already on
`RFID_RESULT_FAILURE` if failed

2.5.3 PowerOff()

```
void       PowerOff();
```

Turns off power to the RFID reader module.

Parameters:
None

Returns:
None

2.5.4 Open()

```
RFID_RESULT  Open();
```

Creates a message queue, a thread for receiving messages from the RFID module driver, and opens the communication port with the RFID reader module.

Parameters:
None

Returns:
`RFID_RESULT_SUCCESS` if all the processes performed normally
`RFID_RESULT_ALREADY_OPENED` if the port was already opened (remains open)

2.5.5 IsOpen()

```
bool       IsOpen();
```

Checks whether the communication port with the reader module is already open or not.

Parameters:
None

Returns:
TRUE if the port is open
FALSE if the port is close

2.5.6 Close()

```
void       Close();
```

Releases the message queue, stops the communication thread, and closes the communication port with the RFID reader module.

Parameters:
None

Returns:
None

2.5.7 Stop()

```
RFID_RESULT  Stop();

RFID_RESULT  Stop(bool Block);
```

Stops the operation currently being performed by the RFID reader module in asynchronous mode. The results of the stop operation can be received by the registered Callback function. If the boolean argument, Block is given and is true, then Stop() will wait until the module is done stopping, and then return, otherwise Stop() returns immediately and you should wait for an event message in the callback indicating the RFID module is fully stopped.

If you call Stop() during a long tag inventory, and then try to start a new tag inventory before it is fully stopped, you may receive NOT_CONNECTED messages in the callback.

Parameters:
`Block`
`[IN] Whether to block until completely stopped or not (optional).`

Returns:
`RFID_RESULT_SUCCESS` if the current operation stopped normally
`RFID_RESULT_NOT_OPENED` if Stop() was called before the reader was opened

2.5.8 IsRunning

```
bool IsRunning()
```

Checks whether the RFID module is busy with a tag inventory or not. If RFID module is busy, true will be returned. If not, false will be returned.

Parameters :

None

Returns:

TRUE if the module is busy
FALSE if the module is not busy

2.5.9 SetCallback()

```
RFID_RESULT  SetCallback(RfidCallbackProc CallbackProc);
```

Specifies which of your functions to call back when the reader module has data available for consumption, for instance to receive tag data, or various end-of-operation messages.

Parameters:

CallbackProc
  [IN] the callback delegate to receive data from the module

Returns:

RFID_RESULT_SUCCESS if callback delegate was registered successfully
RFID_RESULT_NOT_OPENED if the connection to the RFID reader module hasn't been opened yet
RFID_RESULT_INVALID_ARGS if CallbackProc is NULL

2.5.10 GetResult()

```
RFID_RESULT  GetResult(string Result, RFID_CALLBACK_TYPE CallbackType,
           IntPtr wParam, IntPtr lParam);
```

When the the reader completes an operation it calls the registered callback delegate function and hands it a RFIDCALLBACKDATA structure. You then pass the information in that RFIDCALLBACKDATA structure to GetResult(), which copies the data to your Result string variable.

Note: This function must only be called within the delegate function that was registered by the application!

Parameters:

Result
  [OUT] String parameter that will receive the RFID data.
CallbackType
  [IN] the CallbackType as it was handed to your RfidCallbackProc.
wParam
  [IN] the wParam as it was handed to your RfidCallbackProc.
lParam
  [IN] the lParam as it was handed to your RfidCallbackProc.

Returns:

RFID_RESULT_SUCCESS if data successfully retrieved

2.5.11 ReadEPC()

```
RFID_RESULT  ReadEPC(BOOL bSyncMode, RFID_READ_TYPE ReadType, string EpcData);

RFID_RESULT  ReadEPC(BOOL bSyncMode, RFID_READ_TYPE ReadType,
           ref RFIDMASKPARAMS Mask, string EpcData);
```

Reads EPC data from a tag. You can read a single tag, synchronously or asynchronously, or tell the reader read tags on its own, asynchronously, until told to stop. You can optionally pass in a reference to a RFIDMASKPARAMS structure to selectively read tags.

Parameters:

bSyncMode

[IN] if TRUE, synchronously reads EPC data from one tag (ReadType must be EPC_GEN2_ONE_TAG). If FALSE, the callback delegate is called when each tag is read.

ReadType

[IN] indicates whether to read a single tag, or many tags.

pMask

[IN] an optional reference to the tag mask used while reading a tag.

EpcData

[OUT] if the bSyncMode is TRUE, the EpcData variable passed in will store the EPC data, without calling the callback delegate.

Returns:

RFID_RESULT_SUCCESS if successfully read EPC data while in SyncMode, or start to read EPC data in asynchronous mode

2.5.12 ReadEpcEx()

```
RFID_RESULT ReadEpcEx(BOOL bSyncMode, READ_TYPE ReadType, ref RFIDMASKPARAMS_EX
          Mask, string EpcData);

RFID_RESULT ReadEpcEx(BOOL bSyncMode, READ_TYPE ReadType, RFIDSELMASKPARAMS_EX[]
          Masks, Uint NumberOfMasks, string EpcData);
```

This function is just like the ReadEpc() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters :

bSyncMode

[IN] same as ReadEpc.

ReadType

[IN] same as ReadEpc.

Mask

[IN] a reference to the bit-level tag mask used while reading a tag.

Masks

[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.

NumberOfMasks

[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.

EpcData

[OUT] same as ReadEpc.

Returns:

RFID_RESULT_SUCCESS if successfully read EPC data while in SyncMode, or start to read EPC data in asynchronous mode

2.5.13 ReadMemBank()

```
RFID_RESULT  ReadMemBank(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
          UINT nWordCount, BOOL bContinuous String AccessPassword,
          String MemBankData);

RFID_RESULT  ReadMemBank(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
          UINT nWordCount, BOOL bContinuous String AccessPassword,
          ref RFIDMASKPARAMS Mask, String MemBankData);
```

Reads data from an arbitrary memory location of a tag. You can read data in one-word units (two bytes, 16 bits) from any bank and location that is supported by the tag. Locked password fields may require the correct Access password to be passed in as well. You may also optionally pass in a reference to a RFIDMASKPARAMS structure, to read data only from selected tags.

Parameters:

bSyncMode

[IN] if TRUE, synchronously reads data from one tag. The read data is placed in the MemBankData variable. If FALSE, data in read asynchronously, and the registered callback delegate is called instead.

MemBank

[IN] the memory bank to read from.

nWordPtr

[IN] starting word-offset into the bank (starting with 0).

nWordCount

[IN] the number of words (2 bytes) to read (1-255).

bContinuous

[IN] if TRUE, continually repeats the read operation until issued the Stop() command. Not available in synchronous mode (bSyncMode=TRUE);

AccessPassword

[IN] the access password used when accessing protected tag memory.

Mask

[IN] if not null, the reader will apply the specified mask pattern to filter out only certain tags.

MemBankData

[OUT] string parameter where data read from the tag is placed, when operating in synchronous mode (bSyncMode=TURE).

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.14 ReadMemBankEx()

```
RFID_RESULT ReadMemBankEx(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
           UINT nWordCount, BOOL bContinuous, String AccessPassword,
           ref RFIDMASKPARAMS_EX Mask, String MemBankData);

RFID_RESULT ReadMemBankEx(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
           UINT nWordCount, BOOL bContinuous, String AccessPassword,
           RFIDSELMASKPARAMS_EX[] Masks, Uint NumberOfMasks, String MemBankData);
```

This function is just like the ReadMemBank() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:

bSyncMode

[IN] same as ReadMemBank.

MemBank

[IN] same as ReadMemBank.

nWordPtr

[IN] same as ReadMemBank.

nWordCount

[IN] same as ReadMemBank.

bContinuous

[IN] same as ReadMemBank.

AccessPassword

[IN] same as ReadMemBank.

Mask

[IN] a reference to the bit-level tag mask used while reading a tag.

Masks
[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.
NumberOfMasks
[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.
MemBankData
[OUT] same as ReadMemBank.

Returns:
RFID_RESULT of the operation when executed in SyncMode.
RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.15 WriteMemBank()

```
RFID_RESULT  WriteMemBank(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
           String Data, bool bContinuous, String  AccessPassword,
           ref RFIDMASKPARAMS Mask);
```

Writes the given data into a specific bank and offset within a tag's memory. Data must be written in one-word units (2 bytes, 16 bits).

Parameters:
bSyncMode
[IN] if TRUE, data is written synchronously. If FALSE, data is written asynchronously, and the Callback delegate is called with the results.
MemBank
[IN] the memory bank where data is written.
nWordPtr
[IN] word-pointer offset into the memory bank (starting with 0).
Data
[IN] the string data to be written (e.g. "DEADBEEF0001").
bContinuous
[IN] if TRUE, continually repeats the write operation until issued the Stop()command. Not available in synchronous mode (bSyncMode=TRUE);
AccessPassword
[IN] the access password used when accessing protected tag memory.
Mask
[IN] if not null, the reader will apply the specified mask pattern to filter out only certain tags.

Returns:
RFID_RESULT of the operation when executed in SyncMode.
RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.16 WriteMemBankEx()

```
RFID_RESULT WriteMemBankEx(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
           String Data, bool bContinuous, String  AccessPassword,
           ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT WriteMemBankEx(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
           String Data, bool bContinuous, String  AccessPassword,
           RFIDSELMASKPARAMS_EX[] Masks, Uint NumberOfMasks);
```

This function is just like the WriteMemBank() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:
bSyncMode
[IN] same as WriteMemBank.

MemBank
[IN] same as WriteMemBank.
nWordPtr
[IN] same as WriteMemBank.
Data
[IN] same as WriteMemBank.
bContinuous
[IN] same as WriteMemBank.
AccessPassword
[IN] same as WriteMemBank.
Mask
[IN] a reference to the bit-level tag mask used while reading a tag.
Masks
[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.
NumberOfMasks
[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.

Returns:
RFID_RESULT of the operation when executed in SyncMode.
RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.17 LockField()

```
RFID_RESULT  LockField(bool bSyncMode, LOCKUNLOCKFIELD  LockField,
          bool bContinuous, String AccessPassword)

RFID_RESULT  LockField(bool bSyncMode, LOCKUNLOCKFIELD  LockField,
          bool bContinuous, String AccessPassword, ref RFIDMASKPARAMS Mask)
```

Locks the fields specified by the LockField parameter. Locked EPC, TID, and USER fields are read-only without access password. Locked Access and Kill password fields cannot be read from or written to without the access password.

Parameters:
bSyncMode
[IN] if TRUE, the results of the lock operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.
LockField
[IN] structure indicating which field should be locked.
bContinuous
[IN] if TRUE, continually repeats the lock operation until issued the Stop()command. Not available in synchronous mode (bSyncMode=TRUE);
AccessPassword
[IN] the access password used when accessing protected tag memory.
Mask
[IN] if not null, the reader will apply the specified mask pattern to filter out only certain tags.

Returns:
RFID_RESULT of the operation when executed in SyncMode.
RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.18 LockFieldEx()

```
RFID_RESULT LockFieldEx(bool bSyncMode, LOCKUNLOCKFIELD LockField, bool
          Continuous, String AccessPassword, ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT LockFieldEx(bool bSyncMode, LOCKUNLOCKFIELD LockField, bool
          Continuous, String AccessPassword, RFIDSELMASKPARAMS_EX[] Masks);
```

This function is just like the LockField() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:

bSyncMode
`[IN] same as LockField.`

LockField
`[IN] same as LockField.`

bContinuous
`[IN] same as LockField.`

AccessPassword
`[IN] same as LockField.`

Mask
`[IN] a reference to the bit-level tag mask used while reading a tag.`

Masks
`[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.`

NumberOfMasks
`[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.`

Return Values :

`RFID_RESULT` of the operation when executed in SyncMode.
`RFID_RESULT_SUCCESS` if command started successfully in asynchronous mode.

2.5.19 UnLockField()

```
RFID_RESULT  UnLockField(bool bSyncMode, LOCKUNLOCKFIELD UnLockField,
           bool bContinuous, String AccessPassword)

RFID_RESULT  UnLockField(bool bSyncMode, LOCKUNLOCKFIELD UnLockField,
           bool bContinuous, String AccessPassword, ref RFIDMASKPARAMS Mask)
```

Unlocks the fields specified by the UnlockField parameter. If a field is already locked, the correct Access password must be provided in order to unlock it.

Parameters:

bSyncMode
`[IN]  if TRUE, the results of the unlock operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.`

UnLockField
`[IN]  structure indicating which field should be unlocked.`

bContinuous
`[IN]  if TRUE, continually repeats the lock operation until issued the Stop()command. Not available in synchronous mode (bSyncMode=TRUE);`

AccessPassword
`[IN]  the access password used when accessing protected tag memory.`

Mask
`[IN]  if not null, the reader will apply the specified mask pattern to filter out only certain tags.`

Returns:

`RFID_RESULT` of the operation when executed in SyncMode.
`RFID_RESULT_SUCCESS` if command started successfully in asynchronous mode.

2.5.20 UnLockFieldEx

```
RFID_RESULT UnLockFieldEx(bool bSyncMode, LOCKUNLOCKFIELD UnLockField,
          bool bContinuous, String AccessPassword, ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT UnLockFieldEx(bool bSyncMode, LOCKUNLOCKFIELD UnLockField,
          bool bContinuous, String AccessPassword, RFIDSELMASKPARAMS_EX[] Masks,
          Uint NumberOfMasks);
```

This function is just like the UnLockField() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:

bSyncMode
[IN] same as UnLockField.

UnLockField
[IN] same as UnLockField.

bContinuous
[IN] same as UnLockField.

AccessPassword
[IN] same as UnLockField.

Mask
[IN] a reference to the bit-level tag mask used while reading a tag.

Masks
[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.

NumberOfMasks
[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.

Returns:

RFID_RESULT of the operation when executed in SyncMode.
RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.21 PermalockField()

```
RFID_RESULT  PermalockField(bool bSyncMode, PERMALOCK_FIELD PermalockField,
          bool bSecured, bool bContinuous, String AccessPassword)

RFID_RESULT  PermalockField(bool bSyncMode, PERMALOCK_FIELD PermalockField,
          bool bSecured, bool bContinuous, String AccessPassword,
          ref RFIDMASKPARAMS Mask)
```

Permanently locks or unlocks the fields specified by the PermalockField parameter. Use caution when permalocking or permaunlocking – the results are… permanent!

Parameters:

bSyncMode
[IN] if TRUE, the results of the permalock operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.

PermalockField
[IN] structure indicating which field should be locked/unlocked.

bSecured
[IN] if TRUE, fields are permanently *locked*. If FALSE, fields are permanently *unlocked*.

bContinuous
[IN] if TRUE, continually repeats the permalock operation until issued the Stop()command. Not available in synchronous mode (bSyncMode=TRUE);

AccessPassword
[IN] the access password used when accessing protected tag memory.

Mask

[IN] if not null, the reader will apply the specified mask pattern to filter out only certain tags.

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.22 PermalockFieldEx

```
RFID_RESULT PermalockFieldEx(bool bSyncMode, PERMALOCK_FIELD PermalockField,
          bool bSecured, bool bContinuous, String AccessPassword,
          ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT PermalockFieldEx(bool bSyncMode, PERMALOCK_FIELD PermalockField,
          bool bSecured, bool bContinuous, String AccessPassword,
          RFIDSELMASKPARAMS_EX[] Masks, Uint NumberOfMasks);
```

This function is just like the PermaLockField() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:

bSyncMode

[IN] same as PermalockField.

PermalockField

[IN] same as PermalockField.

bSecured

[IN] same as PermalockField.

bContinuous

[IN] same as PermalockField.

AccessPassword

[IN] same as PermalockField.

Mask

[IN] a reference to the bit-level tag mask used while reading a tag.

Masks

[IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.

NumberOfMasks

[IN] count of RFIDSELMASKPARAMS_EX in the supplied array.

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.23 KillTag()

```
RFID_RESULT  KillTag(bool bSyncMode, String KillPassword, bool bContinuous)
          RFID_RESULT KillTag(bool bSyncMode, String KillPassword,
          bool bContinuous, ref RFIDMASKPARAMS Mask)
```

Kills an RFID tag, preventing it from further communicating with a reader. If the tag's Kill Password is "00000000", then you cannot kill the tag. In order to perform kill command, the tag must be programmed with a non-zero password. Once a tag is killed, you can never recover it again.

Parameters:

bSyncMode

[IN] if TRUE, the results of the kill operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.

```
KillPassword
  [IN]  the Kill password that was stored in the tag in advance of the Kill
  operation.
bContinuous
  [IN]  if TRUE, continually repeats the kill operation until issued the
  Stop()command. Not available in synchronous mode (bSyncMode=TRUE);
Mask
  [IN]  if not null, the reader will apply the specified mask pattern to
  filter out only certain tags.
```

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.24 KillTagEx

```
RFID_RESULT KillTagEx(bool bSyncMode, String KillPassword, bool bContinuous,
          ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT KillTagEx(bool bSyncMode, String KillPassword, bool bContinuous,
          RFIDSELMASKPARAMS_EX[] Masks, Uint NumberOfMasks);
```

This function is just like the PermaLockField() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters :

```
bSyncMode
  [IN] same as KillTag.
KillPassword
  [IN] same as KillTag.
bContinuous
  [IN] same as KillTag.
Mask
  [IN] a reference to the bit-level tag mask used while reading a tag.
Masks
  [IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the
  Select command.
NumberOfMasks
  [IN] count of RFIDSELMASKPARAMS_EX in the supplied array.
```

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.25 EnableExtendedInformation()

```
RFID_RESULT  EnableExtendedInformation(bool bEnable)
```

Enables or disables the reading of the extended RSSI (Return Signal Strength Indication) value, while reading tags.

Parameters:

```
bEnable
  [IN] if TRUE, tag reports include RSSI information.
```

Returns:

RFID_RESULT_SUCCESS if performed successfully.

2.5.26 Rfid900SetDefault()

```
RFID_RESULT  Rfid900SetDefault()
```

Initializes the default values of the RFID module.

Parameters:
None

Returns:
`RFID_RESULT_SUCCESS` if performed successfully.

2.5.27 BlockPermalock()

```
RFID_RESULT BlockPermalock(bool bSyncMode, uint ReadLock, MEM_BANK MemBank, uint
            BlockPtr, ushort BlockMask, bool bContinuous, string AccessPassword,
            ref RFIDMASKPARAMS Mask);

RFID_RESULT BlockPermalock(bool bSyncMode, uint ReadLock, MEM_BANK MemBank, uint
            BlockPtr, byte BlockRange, ushort[] BlockMask, bool bContinuous,
            string AccessPassword, ref RFIDMASKPARAMS Mask);
```

BlockPermalock is an optional Class1/Gen2 tag feature which gives you the ability to permanently lock individual blocks of USER memory. Permalocking a block prevents the contents of that block from ever changing again (whether you know the Access password or not), and cannot be undone. You can permalock individual blocks of USER memory with this command. Tags with non-zero Access password already stored in them may require that correct Access password to be passed in to this function call. You may also optionally pass in a reference to a RFIDMASKPARAMS structure, to read data only from selected tags.

BlockPermalock() requires you to specify which bank to place the readlock on (currently, only the USER bank supports this feature), and a 16-bit BlockMask which indicates the desired permalock state of each of 16 USER memory blocks. The most-significant bit in the BlockMask represents the first User block, and so on, which means that to place a permalock on block #1, the BlockMask would be 0x8000 (10000000 00000000_{binary}), not 0x01 (00000000 00000001_{binary}).

There is also a BlockPtr argument which allows you to address and permalock 16-block chunks of USER memory beyond the first 16 blocks. The second version of the BlockPermalock() function takes an array of BlockMasks (so you can permalock more than 16 blocks at a time) instead of a single 16-bit BlockMask, and a BlockRange argument, which is the count of the number of BlockMasks inside the array.

The ReadLock argument is not currently in use, and should always be set to “1”.

Alien Higgs3 and Higgs4 tags have blocks four words (8 bytes, or 64 bits) in size, so the maximum 512-bits of available User memory is subdivided into eight blocks. Other tag manufacturers may subdivide their USER memory into blocks of different size.

> **Note:** The BlockPermalock and AlienBlockReadLock functionality requires the latest version of RFID module firmware to be loaded on your handheld. Units built after September 2013 may come preloaded with the updated RFID module firmware, but older units will need to have their firmware upgraded. Check with Alien support or scan the Alien FTP site for details about updating module firmware.
>
> You can query the RFID module for its firmware version with the FirmwareVersion() function. Updated RFID module firmware reports 1.3.37 (1.3.87 for China/Japan/Europe). Older firmware probably doesn’t support these new RFID commands.

Parameters:
bSyncMode
`[IN]  if TRUE, the results of the BlockPermalock operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.`

ReadLock
`[IN] should always be set to 1. ReadLock=0 means “interrogate the tag for the state of its BlockPermalocks” and isn’t currently supported.`
MemBank
`[IN] the memory bank to operate on (only MEM_BANK.USER is supported).`
BlockPtr
`[IN] a pointer to the specific 16-block group of blocks to operate on.`
BlockMask
`[IN] Bitmask indicating which blocks to permalock (1), and which to leave alone (0).`
bContinuous
`[IN] if TRUE, continually repeats the read operation until issued the Stop() command. Not available in synchronous mode (bSyncMode=TRUE);`
AccessPassword
`[IN] the access password used when accessing protected tag memory.`
Mask
`[IN] if not null, the reader will apply the specified mask pattern to filter out only certain tags.`

Returns:
`RFID_RESULT` of the operation when executed in SyncMode.
`RFID_RESULT_SUCCESS` if command started successfully in asynchronous mode.

2.5.28 BlockPermalockEx()

```
RFID_RESULT BlockPermalockEx(bool bSyncMode, uint ReadLock, MEM_BANK MemBank, uint
            BlockPtr, ushort BlockMask, bool bContinuous, string AccessPassword,
            ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT BlockPermalockEx(bool bSyncMode, uint ReadLock, MEM_BANK MemBank,
            uint BlockPtr, byte BlockRange, ushort[] BlockMask, bool bContinuous,
            string AccessPassword, ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT BlockPermalockEx(bool bSyncMode, uint ReadLock, MEM_BANK MemBank,
            uint BlockPtr, ushort BlockMask, bool bContinuous, string
            AccessPassword, RFIDSELMASKPARAMS_EX[] Masks, uint NumberOfMasks);

RFID_RESULT BlockPermalockEx(bool bSyncMode, uint ReadLock, MEM_BANK MemBank, uint
            BlockPtr, byte BlockRange, ushort[] BlockMask, bool bContinuous,
            string AccessPassword, RFIDSELMASKPARAMS_EX[] Masks, uint
            NumberOfMasks);
```

These functions are just like the BlockPermalock() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode. Just like with BlockPermalock(), there are two versions of the function for each of the mask types – one with a single BlockMask, and another with an array of BlockMasks.

Parameters:
bSyncMode
`[IN] same as BlockPermalock.`
ReadLock
`[IN] same as BlockPermalock.`
MemBank
`[IN] same as BlockPermalock.`
BlockPtr
`[IN] same as BlockPermalock.`
BlockMask
`[IN] same as BlockPermalock.`
bContinuous
`[IN] same as BlockPermalock.`

AccessPassword
  [IN] same as BlockPermalock.
Mask
  [IN] a reference to the bit-level tag mask used while reading a tag.
Masks
  [IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the Select command.
NumberOfMasks
  [IN] count of RFIDSELMASKPARAMS_EX in the supplied array.

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.29 AlienBlockReadLock()

```
RFID_RESULT  AlienBlockReadLock(bool bSyncMode, MEM_BANK MemBank, byte LockMask,
          bool bContinuous, string AccessPassword, ref RFIDMASKPARAMS Mask);
```

An Alien-only feature of Higgs3 and Higgs4 tags, AlienBlockReadLock allows you to prevent unauthorized people (those who don't know the tag's Access password) from being able to read blocks of USER data. Regular locks on tag-memory only prevent others from changing that data, but do nothing to hide sensitive information stored in tag memory, but AlienBlockReadLock allows you to hide information as well. You can hide as well as reveal individual blocks of memory with this command. Tags with non-zero Access password already stored in them may require that correct Access password to be passed in to this function call. You may also optionally pass in a reference to a RFIDMASKPARAMS structure, to read data only from selected tags.

AlienBlockReadLock() requires you to specify which bank to place the readlock on (currently, only the USER bank), and also a single byte mask argument, which is a bitmask representing the desired readlocks on all (up to) eight User blocks. The most-significant bit in the bitmask represents the first User block, and so on, which means that to place a read lock on block #1, the bitmask would be 0x80 (10000000binary), not 0x01 (00000001binary).

Alien Higgs3 tags subdivide User memory into 4-word blocks. The maximum amount of User memory a Higgs3 tag can have is 512 bits, which corresponds to eight blocks. Alien Higgs4 tags subdivide User memory into 2-word blocks. The maximum amount of User memory a Higgs4 tag can have is 128 bits which corresponds to four blocks.

> **Note:** The AlienBlockReadLock and BlockPermalock functionality requires the latest version of RFID module firmware to be loaded on your handheld. Units built after September 2013 may come preloaded with the updated RFID module firmware, but older units will need to have their firmware upgraded. Check with Alien support or scan the Alien FTP site for details about updating module firmware.
>
> You can query the RFID module for its firmware version with the FirmwareVersion() function. Updated RFID module firmware reports 1.3.37 (1.3.87 for China/Japan/Europe). Older firmware probably doesn't support these new RFID commands.

Parameters:

bSyncMode
  [IN] if TRUE, the results of the AlienBlockReadLock operation is immediately reported. If FALSE, the registered Callback delegate is called with the result of the operation.
MemBank
  [IN] the memory bank to operate on (only MEM_BANK.USER is supported).
LockMask
  [IN] Bitmask indicating which blocks to hide (1), and which to reveal (0).
bContinuous
  [IN] if TRUE, continually repeats the read operation until issued the Stop() command. Not available in synchronous mode (bSyncMode=TRUE);

```
AccessPassword
  [IN]  the access password used when accessing protected tag memory.
Mask
  [IN]  if not null, the reader will apply the specified mask pattern to
  filter out only certain tags.
```

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.5.30 AlienBlockReadLockEx()

```
RFID_RESULT  AlienBlockReadLockEx(bool bSyncMode, MEM_BANK MemBank, byte LockMask,
          bool bContinuous, string AccessPassword, ref RFIDMASKPARAMS_EX Mask);

RFID_RESULT  AlienBlockReadLockEx(bool bSyncMode, MEM_BANK MemBank, byte LockMask,
          bool bContinuous, string AccessPassword, RFIDSELMASKPARAMS_EX[] Masks,
          uint NumberOfMasks);
```

This function is just like the AlienBlockReadLock() function, but you can use the extended (bit-level, and action/session-aware) masking parameters. The state-aware version also allows you to provide an array of masks to use during the Select command to the tag. The version with the RFIDSELMASKPARAMS_EX parameter cannot be used while under Sync Mode.

Parameters:

```
bSyncMode
  [IN] same as AlienBlockReadLock.
MemBank
  [IN] same as AlienBlockReadLock.
LockMask
  [IN] same as AlienBlockReadLock.
bContinuous
  [IN] same as AlienBlockReadLock.
AccessPassword
  [IN] same as AlienBlockReadLock.
Mask
  [IN] a reference to the bit-level tag mask used while reading a tag.
Masks
  [IN] array of RFIDSELMASKPARAMS_EX parameters to be issued during the
  Select command.
NumberOfMasks
  [IN] count of RFIDSELMASKPARAMS_EX in the supplied array.
```

Returns:

RFID_RESULT of the operation when executed in SyncMode.

RFID_RESULT_SUCCESS if command started successfully in asynchronous mode.

2.6 RFID Properties

2.6.1 FirmwareVersion

```
string FirmwareVersion { get; }
```

Returns the firmware version of the RFID Module.

2.6.2 HoppingMode

```
HOPPING_MODE HoppingMode { get; }
```

Returns the Hopping Mode which has been set in the module.

2.6.3 InventoryTarget

```
INVENTORIED_STATE InventoryTarget { set; get; }
```

Gets or Sets the target while performing the ReadEpc action.

2.6.4 ChannelState

```
uint ChannelState { set; get; }
```

Gets or Sets the RF channel(s) in use by the reader. Only certain regions (e.g. EU models) allow end-users to select particular channels. This property's value is a binary numeral - the least significant bit corresponds to RF channel #1, the next most significant bit corresponds to RF channel #2, etc. Any bit that is set will cause the corresponding RF channel to be utilized. To stay on one fixed channel, make sure only one bit is set (the ChannelState will then be a power of 2).

2.6.5 OperationTime

```
uint OperationTime { set; get; }
```

Gets or Sets or RFID module's maximum command execution time, in units of seconds. Specifying an OperationTime of 0 tells the reader to only stop when you issue the Stop() command.

OperationTime limits are 0-60. The default value is 0 (never stop).

2.6.6 PowerLevel

```
uint PowerLevel { set; get; }
```

Gets or Sets the RFID module's power attenuation level. The default value is 0 (no attenuation, max power), and the maximum value is 30 (full attenuation, minimum power). Transmitted power is full power reduced by this setting. For example, Powerlevel = 3 means the reader's output power is reduced by 3dBm (about half power).

2.6.7 ProtocolVersion

```
string ProtocolVersion { get; }
```

Returns the protocol version of the RFID Module.

2.6.8 QValue

```
uint QValue { set; get; }
```

Gets or Sets the starting Q parameter value that is used in the Gen2 protocol to read tags. It indicates approximately how many tags to expect, where numTags = 2^Q.

QValue limits are 0-15, with the default value being 5.

2.6.9 Session

```
SESSION_TYPE Session { set; get; }
```

Gets or Sets the reader's Session value that is used in the Gen2 protocol to keep track of inventoried tags. Please refer to the Gen2 protocol specification for information on the Session value.

2.6.10 Selects

```
int Selects { set; get; }
```

When using the xxxEx functions with the Select Mask option, then the reader will issue the number of Selects given by this parameter when sending out mask commands. “Selecting” a tag has the effect of marking it as “not inventoried”, so that it will participate in the next inventory round.

If you set the Selects property to 0, then the reader will not issue and Select commands to the tags.

If no mask is given and you set Selects to >0, then the reader will select all tags in the field.

3 Barcode Scanner

3.1 Introduction

All Alien handheld readers include a laser barcode scanner. The ALR-9000/9010 handhelds have a 1D scanner (single red line), while the ALH-9001/9011 units have a 2D scanner (scans a rectangular area).

3.2 Barcode Scanner Data Structures

3.2.1 BARCODE_RESULT

Type	Item	Description
BARCODE_RESULT	BARCODE_RESULT_SUCCESS	operation successfully performed
	BARCODE_RESULT_INVALID_ARGS	invalid parameter
	BARCODE_RESULT_OUTOFMEMORY	failed to assign a memory resource
	BARCODE_RESULT_UNSUPPORTED	command not currently supported
	BARCODE_RESULT_ALREADY_OPENED	device has already been opened
	BARCODE_RESULT_FAILURE	failed to perform operation
	BARCODE_RESULT_INIT_FAILURE	scanner initialization failed
	BARCODE_RESULT_NOT_OPENED	function called w/o calling Open() first

3.2.2 BARCODECALLBACK

A callback delegate function that you provide to receive barcode data from the scanner. In order to process the barcode data in your application, you first need to use the *SetCallback(BARCODECALLBACK pFunc)* function to register your delegate function.

```
public delegate void BARCODECALLBACK();
```

3.2.3 IMAGE_SIZE

Holds the size of an image captured by the 2D Scanner.

```
public enum IMAGE_SIZE
{
  CAP_IMG_VGA = 1, // 640*480
  CAP_IMG_QVGA,    // 320*240
  CAP_IMG_PREVIEW  // 240*180
}
```

3.2.4 IMAGE_FORMAT

Represents format of an image captured by the 2D Scanner.

```
public enum IMAGE_FORMAT
{
  IMG_FORMAT_JPEG = 1,
  IMG_FORMAT_BMP
}
```

3.2.5 BARCODECAPTUREPARAMS

This structure defines images captured by the 2D Scanner.

```
public struct BARCODECAPTUREPARAMS
{
  byte   ImgFormat;
  byte   ImgSize;
  string sFilePath; // path to storage location of image (including filename)
}
```

3.2.6 GENERAL_CONFIG

The structure defines the security level, scanning angle, and bi-directional redundancy of 1D Scanner.

```
public struct GENERAL_CONFIG
{
  bool bBiDirectionalRedundancy;
  int  nAngle;
  int  nLinearCodeSecurityLevel;
}
```

- **bBiDirectionalRedundancy:** Whether a bar code must be successfully scanned in both directions (forward and reverse) before being decoded.
- **nAngle:** Scanning angle: 5=Narrow, 6=Wide
- **nLinearCodeSecurityLevel:** Four levels(1-4) of decode security for linear code types are supported. Select a higher security level for barcodes of low quality. As security levels increase, the decoder's aggressiveness decreases.

3.2.7 OCRMode

This determines which OCR fonts (if any) are selected for decoding.

```
public enum OCRMode
{
  OCRMODE_A
  OCRMODE_B
  OCRMODE_DISABLED
  OCRMODE_MICR_UNSUPPORTED
  OCRMODE_MONEY
}
```

3.2.8 SYMBOL_CONFIG

This structure holds each symbology's configuration options.

```
public struct SYMBOL_CONFIG
{
  IntPtr pConfigData; // pointer to a symbology configuration structure
  int Symbol          // the symbology type enumeration
}
```

3.2.9 SYMBOLOGIES_1D

This enumeration lists all of the 1D scanner symbologies.

```
public enum SYMBOLOGIES_1D
{
  NUM_OF_1D_SYMBOLOGIES
  SYMBOL_1D_BOOKLAND
  SYMBOL_1D_CHINESE25
  SYMBOL_1D_CODABAR
  SYMBOL_1D_CODE11
  SYMBOL_1D_CODE128
  SYMBOL_1D_CODE39
  SYMBOL_1D_CODE93
  SYMBOL_1D_DISCRETE25
  SYMBOL_1D_EAN13
  SYMBOL_1D_EAN8
  SYMBOL_1D_INTERLEAVED25
  SYMBOL_1D_ISBT128
  SYMBOL_1D_MSI
  SYMBOL_1D_RSS14
  SYMBOL_1D_RSSEXPANDED
  SYMBOL_1D_RSSLIMITED
  SYMBOL_1D_TRIOPTIC39
  SYMBOL_1D_UCCEAN128
  SYMBOL_1D_UPCA
  SYMBOL_1D_UPCE
  SYMBOL_1D_UPCE1
}
```

3.2.9.1 CONFIG_1D_CHINESE25

```
public struct CONFIG_1D_CHINESE25
{
  bool  bEnabled;
}
```

3.2.9.2 CONFIG_1D_CODABAR

```
public struct CONFIG_1D_CODABAR
{
  bool  bEnabled;
  bool  bCLSIEditing;
  bool  bNOTISEditing;
  short nMaxLength;
  short nMinLength;
}
```

3.2.9.3 CONFIG_1D_CODE128

```
public struct CONFIG_1D_CODE128
{
  bool  bEnabled;
}
```

3.2.9.4 CONFIG_1D_CODE39

```
public struct CONFIG_1D_CODE39
{
  bool  bEnabled;
  bool  bCheckDigitVerify;
  bool  bCode32Prefix;
  bool  bConvertCode39ToCode32;
  bool  bEnableTriopticCode39;
  bool  bFullAscii;
  bool  bXmitCheckDigit;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether Code 39 is enabled or disabled
- **bCheckDigitVerify**: Checks the integrity of a Code 39 symbol to ensure it complies with specified algorithms. Only Code 39 symbols which include a modulo 43 check digit are decoded.
- **bCode32Prefix**: Appends the character 'A' to the start of the decode data if enabled.
- **bConvertCode39ToCode32**: Converts Code 39 to Code 32.
- **bEnableTriopticCode39**: whether Trioptic Code 39 is enabled or disabled.
- **bFullAscii**: Enables or disables Code 39 Full ASCII
- **bXmitCheckDigit**: whether Code 39 Check Digit is enabled or disabled.
- **nMaxLength**: max length for Code 39
- **nMinLength**: min length for Code 39

3.2.9.5 CONFIG_1D_CODE93

```
public struct CONFIG_1D_CODE93
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.9.6 CONFIG_1D_DISCRET25

```
public struct CONFIG_1D_DISCRET25
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.9.7 CONFIG_1D_INTERLEAVED25

```
public struct CONFIG_1D_INTERLEAVED25
{
  bool  bEnabled;
  bool  bConvertI2of5ToEAN13;
  bool  bXmitCheckDigit;
  short nCheckDigitVerify;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether Interleaved 2 of 5 is enabled or disabled
- **bConvertI2of5ToEAN13**: Returns whether a 14-character I 2 of 5 code is converted to EAN-13 and transmitted to the host as EAN-13.
- **bXmitCheckDigit**: whether Interleaved 2 of 5 Check Digit is enabled or disabled.
- **nCheckDigitVerify**: Checks the integrity of an I 2 of 5 symbol to ensure it complies with specified algorithms, either USS (Uniform Symbology Specification), or OPCC (Optical Product Code Council).
- **nMaxLength**: max length for Interleaved 2 of 5
- **nMinLength**: min length for Interleaved 2 of 5

3.2.9.8 CONFIG_1D_ISBT128

```
public struct CONFIG_1D_ISBT128
{
  bool  bEnabled;
}
```

3.2.9.9 CONFIG_1D_MSI

```
public struct CONFIG_1D_MSI
{
  bool  bEnabled;
  bool  bCheckDigitAlgorithm;
  bool  bCheckDigitVerify;
  bool  bXmitCheckDigit;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether MSI is enabled or disabled
- **bCheckDigitAlgorithm**: When Two MSI check digits is selected, sets an additional verification is required to ensure integrity. Two following algorithms may be selected: MOD10/MOD11 or MOD10/MOD10.
- **bCheckDigitVerify**: Sets the number of check digits at the end of the bar code that verify the integrity of the data. At least one check digit is always required. Check digits are not automatically transmitted with the data.
- **bXmitCheckDigit**: whether MSI Check Digit is transmitted with the data.
- **nMaxLength**: max length for MSI
- **nMinLength**: min length for MSI

3.2.9.10 CONFIG_1D_RSS

```
public struct CONFIG_1D_RSS
{
  bool  bRSS14Enabled;
  bool  bRSSExpandedEnabled;
  bool  bRSSLimitedEnabled;
  bool  bConvertRSSToUPCEAN;
}
```

3.2.9.11 CONFIG_1D_UPCEAN

```
public struct CONFIG_1D_UPCEAN
{
  bool  bBooklandEnabled;
  bool  bConvertEAN8ToEAN13;
  bool  bConvertUPCE1ToUPCA;
  bool  bConvertUPCEToUPCA;
  bool  bEAN13Eanbled;
  bool  bEAN8Enabled;
  bool  bEANZeroExtend;
  bool  bUCCCouponExtendedCode;
  bool  bUCCEAN128Enabled;
  bool  bUPCAEnabled;
  bool  bUPCE1Enabled;
  bool  bUPCEEnabled;
  bool  bXmitUPCACheckChar;
  bool  bXmitUPCE1CheckChar;
  bool  bXmitUPCECheckChar;
  short nDecodeUPCEANSupplemental;
  short nDecodeUPCEANSupplementalRedundancy;
  short nUPCAPreamble;
  short nUPCE1Preamble;
  short nUPCEANSecurityLevel;
  short nUPCEPreamble;
}
```

- **bBooklandEnabled**: Whether Bookland is enabled or disabled
- **bConvertEAN8ToEAN13**: When EAN Zero Extend is enabled, labels the extended symbol as either an EAN-13 bar code, or an EAN-8 bar code. When EAN Zero Extend is disabled, this parameter has no effect on bar code data.
- **bConvertUPCE1ToUPCA**: Converts UPC-E1 (zero suppressed) decoded data to UPC-A format before transmission. After conversion, data follows UPC-A format and is affected by UPC-A programming selections (e.g., Preamble, Check Digit).
- **bConvertUPCEToUPCA**: Converts UPC-E (zero suppressed) decoded data to UPC-A format before transmission. After conversion, data follows UPC-A format and is affected by UPC-A programming selections (e.g., Preamble, Check Digit).
- **bEAN13Eanbled**: Whether EAN-13 is enabled or disabled
- **bEAN8Enabled**: Whether EAN-8 is enabled or disabled
- **bEANZeroExtend**: When enabled, five leading zeros are added to decoded EAN-8 symbols to make them compatible in format to EAN-13 symbols.
- **bUCCCouponExtendedCode**: Whether UCC Coupon is enabled or disabled
- **bUCCEAN128Enabled**: Whether UCCEAN 128 is enabled or disabled
- **bUPCAEnabled**: Whether UPC-A is enabled or disabled
- **bUPCE1Enabled**: Whether UPC-E1 is enabled or disabled
- **bUPCEEnabled**: Whether UPC-E is enabled or disabled
- **bXmitUPCACheckChar**: Transmits the symbol with or without the UPC-A check digit.
- **bXmitUPCE1CheckChar**: Transmits the symbol with or without the UPC-E1 check digit.
- **bXmitUPCECheckChar**: Transmits the symbol with or without the UPC-E check digit.
- **nDecodeUPCEANSupplemental**: Sets Decode UPC/EAN supplementals option. Supplementals are additionally appended characters (2 or 5) according to specific code format conventions (e.g., UPC A+2, UPC E+2, EAN 8+2). Three options are available.
    - 0 : Decode supplementals
    - 1 : Ignore supplementals
    - 2 : Auto-discriminate supplementals
- **nDecodeUPCEANSupplementalRedundancy**: When auto-discriminate UPC/EAN supplementals is selected, this adjusts the number of times a symbol without supplementals is decoded before transmission. The range is from 2 to 20 times. Five or above is recommended when decoding a mix of UPC/EAN symbols with and without supplementals, and the auto-discriminate option is selected. This is an integer value in the range [2..20]
- **nUPCAPreamble**: Returns the selected UPC-A Preamble option: transmit system character only, transmit system character and country code (“0” for USA), or no preamble transmitted. The lead-in characters are considered part of the symbol.
    - 0 : No preamble
    - 1 : System character
    - 2 : System character, country code

- **nUPCE1Preamble**: Returns the selected UPC-E1 Preamble option: transmit system character only, transmit system character and country code (“0” for USA), or no preamble transmitted. The lead-in characters are considered part of the symbol.
  - 0 : No preamble
  - 1 : System character
  - 2 : System character, country code
- **nUPCEANSecurityLevel**: Sets the UPC/EAN Security Level. There are four levels of decode security for UPC/EAN bar codes. Select a higher level of security are provided for decreasing levels of bar code quality. There is an inverse relationship between security and decoder aggressiveness, so be sure to choose only that level of security necessary for any given application.
  - - UPC/EAN Security Level 0: This default setting allows the decoder to operate in its most aggressive state, while providing sufficient security in decoding “in-spec” UPC/EAN bar codes.
  - - UPC/EAN Security Level 1: As bar code quality levels diminish, certain characters become prone to mis-decodes before others (i.e. 1, 2, 7, and 8). If you are experiencing mis-decodes of poorly printed bar codes and the mis-decodes are limited to these characters, select this security level.
  - - UPC/EAN Security Level 2: If you are experiencing mis-decodes of poorly printed bar codes and the mis-decodes are not limited to characters 1, 2, 7, and 8, select this security level.
  - - UPCIEAN Security Level 3: If you have tried Security Level 2, and are still experiencing mis-decodes, select this security level. Be advised, selecting this option is an extreme measure against mis-decoding severely out of spec bar codes. Selection of this level of security significantly impairs the decoding ability of the decoder. If this level of security is necessary you should try to improve the quality of your bar codes.
- **nUPCEPreamble**: Sets the UPC-E Preamble option. Three options are given for lead-in characters for UPC-E symbols transmitted to the host device: transmit system character only, transmit system character and country code (“0” for USA), and no preamble transmitted. The lead-in characters are considered part of the symbol.
  - 0 : No preamble
  - 1 : System character
  - 2 : System character, country code

3.2.9.12 CONFIG_1D_CODE11

```
public struct CONFIG_1D_CODE11
{
  bool  bEnabled;
  bool  bCheckDigitVerify;
  bool  bXmitCheckDigit;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: Whether Code 11 is enabled or disabled
- **bCheckDigitVerify**: Sets the number of check digits at the end of the bar code that verify the integrity of the data. At least one check digit is always required. Check digits are not automatically transmitted with the data.
- **bXmitCheckDigit**: Whether Code 11 Check Digit is transmitted with the data.
- **nMaxLength**: Max length for Code 11
- **nMinLength**: Min length for Code 11

3.2.10 SYMBOLOGIES_2DSWD

This enumeration lists all of the 2D scanner symbologies.

```
public enum SYMBOLOGIES_2DSWD
{
  NUM_OF_2DSWD_SYMBOLOGIES
  SYMBOL_2DSWD_AUSPOST
  SYMBOL_2DSWD_AZTEC
  SYMBOL_2DSWD_BPO
  SYMBOL_2DSWD_CANPOST
  SYMBOL_2DSWD_CHINAPOST
  SYMBOL_2DSWD_CODABAR
  SYMBOL_2DSWD_CODABLOCK
  SYMBOL_2DSWD_CODE11
  SYMBOL_2DSWD_CODE128
  SYMBOL_2DSWD_CODE16K
  SYMBOL_2DSWD_CODE32
  SYMBOL_2DSWD_CODE39
  SYMBOL_2DSWD_CODE49
  SYMBOL_2DSWD_CODE93
  SYMBOL_2DSWD_COMPOSITE
  SYMBOL_2DSWD_COUPONCODE
  SYMBOL_2DSWD_DATAMATRIX
  SYMBOL_2DSWD_DUTCHPOST
  SYMBOL_2DSWD_EAN13
  SYMBOL_2DSWD_EAN8
  SYMBOL_2DSWD_GEN_CODE128
  SYMBOL_2DSWD_GS1_128
  SYMBOL_2DSWD_IATA25
  SYMBOL_2DSWD_IDTAG
  SYMBOL_2DSWD_INT25
  SYMBOL_2DSWD_ISBT
  SYMBOL_2DSWD_JAPOST
  SYMBOL_2DSWD_KOREAPOST
  SYMBOL_2DSWD_MATRIX25
  SYMBOL_2DSWD_MAXICODE
  SYMBOL_2DSWD_MESA
  SYMBOL_2DSWD_MICROPDF
  SYMBOL_2DSWD_MSI
  SYMBOL_2DSWD_OCR
  SYMBOL_2DSWD_PDF417
  SYMBOL_2DSWD_PLANET
  SYMBOL_2DSWD_PLESSEY
  SYMBOL_2DSWD_POSICODE
  SYMBOL_2DSWD_POSTNET
  SYMBOL_2DSWD_QR
  SYMBOL_2DSWD_RSS
  SYMBOL_2DSWD_STRT25
  SYMBOL_2DSWD_TELEPEN
  SYMBOL_2DSWD_TLCODE39
  SYMBOL_2DSWD_TRIOPTIC
  SYMBOL_2DSWD_UPCA
  SYMBOL_2DSWD_UPCE0
  SYMBOL_2DSWD_UPCE1
  SYMBOL_2DSWD_USPS4CB
}
```

3.2.10.1 CONFIG_2DSWD_AUSPOST

```
public struct CONFIG_2DSWD_AUSPOST
{
  bool  bEnabled;
}
```

3.2.10.2 CONFIG_2DSWD_AZTEC

```
public struct CONFIG_2DSWD_AZTEC
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.3 CONFIG_2DSWD_BPO

```
public struct CONFIG_2DSWD_BPO
{
  bool  bEnabled;
}
```

3.2.10.4 CONFIG_2DSWD_CANPOST

```
public struct CONFIG_2DSWD_CANPOST
{
  bool  bEnabled;
}
```

3.2.10.5 CONFIG_2DSWD_CHINAPOST

```
public struct CONFIG_2DSWD_CHINAPOST
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.6 CONFIG_2DSWD_CODABAR

```
public struct CONFIG_2DSWD_CODABAR
{
  bool  bEnabled;
  bool  bCheckCharOn;
  bool  bSSXmit;
  bool  bXmitCheckChar;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether Codabar is enabled or disabled
- **bCheckCharOn**: Whether the engine will read Codabar bar codes with or without check characters. If TRUE, the engine only decodes Codabar codes with a check character. If FALSE, the decoder decodes codes with or without a check character.
- **bSSXmit**: Whether the start and stop characters are returned in the data string after a successful Codabar decode.
- **bXmitCheckChar**: Whether the engine will return the check character as part of the data string after a successful decode.
- **nMaxLength**: max length for Codabar
- **nMinLength**: min length for Codabar

3.2.10.7 CONFIG_2DSWD_CODABLOCK

```
public struct CONFIG_2DSWD_CODABLOCK
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.8 CONFIG_2DSWD_CODE11

```
public struct CONFIG_2DSWD_CODE11
{
  bool  bEnabled;
  bool  bTwoCheckDigits;
  short nMaxLength;
  short nMinLength;
}
```

- bEnabled: whether Code 11 is enabled or disabled
- bTwoCheckDigits: Whether the engine is decoding Code 11 bar codes that have two check digits. If TRUE, the engine is decoding Code 11 bar codes that have two check digits. If FALSE the engine decodes Code 11 bar codes as if they were printed with only one check digit.
- nMaxLength: max length for Code 11
- nMinLength : min length for Code 11

3.2.10.9 CONFIG_2DSWD_CODE128

```
public struct CONFIG_2DSWD_CODE128
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.10 CONFIG_2DSWD_CODE16K

```
public struct CONFIG_2DSWD_CODE16K
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.11 CONFIG_2DSWD_CODE32

```
public struct CONFIG_2DSWD_CODE32
{
  bool  bEnabled;
}
```

3.2.10.12 CONFIG_2DSWD_CODE39

```
public struct CONFIG_2DSWD_CODE39
{
  bool  bEnabled;
  bool  bAppend;
  bool  bCheckCharOn;
  bool  bFullAscii;
  bool  bSSXmit;
  bool  bXmitCheckChar;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: Whether Code 39 is enabled or disabled
- **bAppend**: Whether the engine should append together and buffer up Code 39 symbols that start with a space (excluding the start and stop characters). The engine stores the symbols in the order in which they are read. It returns the data after a Code 39 symbol with no leading space is read. The return data has the leading spaces removed.
- **bCheckCharOn**: Whether the engine will read Code 39 bar codes with or without check characters. If TRUE, the engine only decodes Code 39 codes with a check character. If FALSE, the decoder decodes codes with or without a check character.
- **bFullAscii**: Whether certain character pairs within the bar code symbol are interpreted and returned as a single character.
- **bSSXmit**: Whether the start and stop characters are returned in the data string after a successful Code 39 decode.
- **bXmitCheckChar**: Whether the engine will return the check character as part of the data string after a successful decode.
- **nMaxLength**: max length for Code 39
- **nMinLength**: min length for Code 39

3.2.10.13 CONFIG_2DSWD_CODE49

```
public struct CONFIG_2DSWD_CODE49
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.14 CONFIG_2DSWD_CODE93

```
public struct CONFIG_2DSWD_CODE93
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.15 CONFIG_2DSWD_COMPOSITE

```
public struct CONFIG_2DSWD_COMPOSITE
{
  bool  bEnabled;
  bool  bCompositeOnUpcEan;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: Whether Composite code is enabled or disabled
- **bCompositeOnUpcEan**: BOOL variable that contains the enabled state of EAN•UCC Composite code associated with EAN and UPC codes.
- **nMaxLength**: max length for Composite code
- **nMinLength**: min length for Composite code

3.2.10.16 CONFIG_2DSWD_COUPONCODE

```
public struct CONFIG_2DSWD_COUPONCODE
{
  bool  bEnabled;
}
```

3.2.10.17 CONFIG_2DSWD_DATAMATRIX

```
public struct CONFIG_2DSWD_DATAMATRIX
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.18 CONFIG_2DSWD_DUTCHPOST

```
public struct CONFIG_2DSWD_DUTCHPOST
{
  bool  bEnabled;
}
```

3.2.10.19 CONFIG_2DSWD_EAN13

```
public struct CONFIG_2DSWD_EAN13
{
  bool  bEnabled;
  bool  bAddenda2Digit;
  bool  bAddenda5Digit;
  bool  bAddendaReq;
  bool  bAddendaSeparator;
  bool  bXmitCheckChar;
}
```

- **bEnabled**: whether EAN 13 is enabled or disabled
- **bAddenda2Digit**: Whether the engine will look for a 2 digit addendum at the end of the EAN/JAN-13 barcode. If TRUE, and an addendum is present, the engine adds the two digit addendum data to the end of the message. If FALSE, the engine ignores addendum data.
- **bAddenda5Digit**: Whether the engine will look for a 5 digit addenda at the end of the EAN/JAN-13 barcode. If TRUE, and an addendum is present, the engine adds the five digit addendum data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddendaReq**: Whether the engine will decode only EAN/JAN-13 bar codes that have a 2 or 5 digit addenda. If TRUE, the engine decodes only EAN symbols with an addenda. If FALSE, the engine decodes all enabled EAN/JAN-13 symbols.
- **bAddendaSeparator**: Whether there is a space character between the data from the bar code and the data from the addenda.
- **bXmitCheckChar**: Whether the engine will return the check character as part of the data string after a successful decode.

3.2.10.20 CONFIG_2DSWD_EAN8

```
public struct CONFIG_2DSWD_EAN8
{
  bool  bEnabled;
  bool  bAddenda2Digit;
  bool  bAddenda5Digit;
  bool  bAddendaReq;
  bool  bAddendaSeparator;
  bool  bXmitCheckChar;
}
```

- **bEnabled**: whether EAN 13 is enabled or disabled
- **bAddenda2Digit**: Whether the engine will look for a 2 digit addenda at the end of the EAN/JAN-8 barcode. If TRUE, and an addendum is present, the engine adds the two digit addenda data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddenda5Digit**: Whether the engine will look for a 5 digit addenda at the end of the EAN/JAN-8 barcode. If TRUE, and an addendum is present, the engine adds the five digit addenda data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddendaReq**: Whether the engine will decode only EAN/JAN-8 bar codes that have a 2 or 5 digit addenda. If TRUE, the engine decodes only EAN/JAN-8 symbols with an addenda. If FALSE, the engine decodes all enabled EAN/JAN-8 symbols.
- **bAddendaSeparator**: Whether there is a space character between the data from the bar code and the data from the addenda.
- **bXmitCheckChar**: Whether the engine will return the check character as part of the data string after a successful decode.

3.2.10.21 CONFIG_2DSWD_GENERICCODE128

```
public struct CONFIG_2DSWD_GENERICCODE128
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.22 CONFIG_2DSWD_GS1128

```
public struct CONFIG_2DSWD_GS1128
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.23 CONFIG_2DSWD_IATA25

```
public struct CONFIG_2DSWD_IATA25
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.24 CONFIG_2DSWD_INT25

```
public struct CONFIG_2DSWD_INT25
{
  bool  bEnabled;
  bool  bCheckDigitOn;
  bool  bXmitCheckDigit;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether Interleaved 2 of 5 is enabled or disabled
- **bCheckDigitOn**: Whether the engine will read Interleaved 2 of 5 bar codes with or without check characters.
- **bXmitCheckDigit**: Whether the engine will return the check digit as part of the data string after a successful decode.
- **nMaxLength**: max length for Interleaved 2 of 5
- **nMinLength**: min length for Interleaved 2 of 5

3.2.10.25 CONFIG_2DSWD_ISBT

```
public struct CONFIG_2DSWD_ISBT
{
  bool  bEnabled;
}
```

3.2.10.26 CONFIG_2DSWD_JAPOST

```
public struct CONFIG_2DSWD_JAPOST
{
  bool  bEnabled;
}
```

3.2.10.27 CONFIG_2DSWD_KOREAPOST

```
public struct CONFIG_2DSWD_KOREAPOST
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.28 CONFIG_2DSWD_MAXICODE

```
public struct CONFIG_2DSWD_MAXICODE
{
  bool  bEnabled;
  bool  bCarrierMsgOnly;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: Whether Maxi Code is enabled or disabled
- **bCarrierMsgOnly**: This parameter is ignored and no longer supported.
- **nMaxLength**: max length for Maxi Code
- **nMinLength**: min length for Maxi Code

3.2.10.29 CONFIG_2DSWD_MESA

```
public struct CONFIG_2DSWD_MESA
{
  bool  b1MSEnabled;
  bool  b3MSEnabled;
  bool  b9MSEnabled;
  bool  bEMSEnabled;
  bool  bIMSEnabled;
  bool  bUMSEnabled;
}
```

- **b1MSEnabled**: Whether Code 128 Mesa is enabled
- **b3MSEnabled**: Whether Code 39 Mesa is enabled
- **b9MSEnabled**: Whether Code 93 Mesa is enabled
- **bEMSEnabled**: Whether EAN13 Mesa is enabled
- **bIMSEnabled**: Whether Interleaved 2 of 5 Mesa is enabled
- **bUMSEnabled**: Whether UPCA Mesa is enabled

3.2.10.30 CONFIG_2DSWD_MICROPDF

```
public struct CONFIG_2DSWD_MICROPDF
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.31 CONFIG_2DSWD_MSI

```
public struct CONFIG_2DSWD_MSI
{
  bool  bEnabled;
  bool  bXmitCheckChar;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether MSI is enabled or disabled
- **bXmitCheckChar**: Whether the engine will return the check character as part of the data string after a successful decode.
- **nMaxLength**: max length for MSI
- **nMinLength**: min length for MSI

3.2.10.32 CONFIG_2DSWD_MX25

```
public struct CONFIG_2DSWD_MX25
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.33 CONFIG_2DSWD_OCR

```
public struct CONFIG_2DSWD_OCR
{
  String  CheckChar;
  String  GroupG;
  String  GroupH;
  String  Template;
  OCRMode nFont;
}
```

- **CheckChar**: A null-terminated string that represents a check character position in the template strings.
- **GroupG**: A null-terminated string that represents a list of characters that can be substituted for the lower-case 'g' in the template strings.
- **GroupH**: A null-terminated string that represents a list of characters that can be substituted for the lower-case 'h' in the template strings.
- **Template**: A null-terminated string that indicates one or more template patterns for the OCR decode. The following characters are allowed:
  - A-Z : capital letters are matched as is
  - d : a digit from 0 - 9
  - a : alphanumeric character
  - l : alphabetic letter
  - g : any character specified in group G
  - h : any character specified in group H

3.2.10.34 CONFIG_2DSWD_PDF417

```
public struct CONFIG_2DSWD_PDF417
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.35 CONFIG_2DSWD_PLANET

```
public struct CONFIG_2DSWD_PLANET
{
  bool  bEnabled;
  bool  bXmitCheckDigit;
}
```

- **bEnabled**: whether Planet code is enabled or disabled
- **bXmitCheckDigit**: Whether the engine will return the check digit as part of the data string after a successful decode.

3.2.10.36 CONFIG_2DSWD_PLESSEY

```
public struct CONFIG_2DSWD_PLESSEY
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.37 CONFIG_2DSWD_POSICODE

```
public struct CONFIG_2DSWD_POSICODE
{
  bool  bEnabled;
  short wLimited;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether PosiCode is enabled or disabled
- **wLimited**: short variable that reflects if Posicode Limited A or Posicode Limited B decoding is enabled. A value of 1 indicates Posicode Limited A is enabled, and a value of 2 indicates Posicode Limited B decoding is enabled. A value of 0 indicates that decoding of both Limited A and Limited B is disabled.
- **nMaxLength**: max length for PosiCode
- **nMinLength**: min length for PosiCode

3.2.10.38 CONFIG_2DSWD_POSTNET

```
public struct CONFIG_2DSWD_POSTNET
{
  bool  bEnabled;
  bool  bXmitCheckDigit;
}
```

- **bEnabled**: whether Postnet is enabled or disabled
- **bXmitCheckDigit**: Whether the engine will return the check digit as part of the data string after a successful decode.

3.2.10.39 CONFIG_2DSWD_QR

```
public struct CONFIG_2DSWD_QR
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.40 CONFIG_2DSWD_RSS

```
public struct CONFIG_2DSWD_RSS
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.41 CONFIG_2DSWD_STRT25

```
public struct CONFIG_2DSWD_STRT25
{
  bool  bEnabled;
  short nMaxLength;
  short nMinLength;
}
```

3.2.10.42 CONFIG_2DSWD_TELEPEN

```
public struct CONFIG_2DSWD_TELEPEN
{
  bool  bEnabled;
  bool  bOldStyle;
  short nMaxLength;
  short nMinLength;
}
```

- **bEnabled**: whether Telepen is enabled or disabled
- **bOldStyle**: Whether the engine is configured to reads Telepen labels that were encoded with either the original or the AIM specification.
- **nMaxLength**: max length for Telepen
- **nMinLength**: min length for Telepen

3.2.10.43 CONFIG_2DSWD_TLC39

```
public struct CONFIG_2DSWD_TLC39
{
  bool  bEnabled;
}
```

3.2.10.44 CONFIG_2DSWD_TRIOPTIC

```
public struct CONFIG_2DSWD_TRIOPTIC
{
  bool  bEnabled;
}
```

3.2.10.45 CONFIG_2DSWD_UPCA

```
public struct CONFIG_2DSWD_UPCA
{
  bool  bEnabled;
  bool  bAddenda2Digit;
  bool  bAddenda5Digit;
  bool  bAddendaReq;
  bool  bAddendaSeparator;
  bool  bXmitCheckDigit;
  bool  bXmitNumSys;
}
```

- **bEnabled**: whether UPC-A is enabled or disabled
- **bAddenda2Digit**: Whether the engine will look for a 2 digit addenda at the end of the UPC bar code. If TRUE, and an addenda is present, the engine adds the two digit addenda data to the end of the message. If FALSE, the engine ignores addenda data. the original or the AIM specification.
- **bAddenda5Digit**: Whether the engine will look for a 5 digit addenda at the end of the UPC bar code. If TRUE, and an addenda is present, the engine adds the five digit addenda data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddendaReq**: Whether the engine will decode only UPC bar codes that have a 2 or 5 digit addenda. If TRUE, the engine decodes only UPC symbols with an addenda. If FALSE, the engine decodes all enabled UPC symbols.
- **bAddendaSeparator**: Whether there is a space character between the data from the bar code and the data from the addenda.
- **bXmitCheckDigit**: Whether the engine will return the check digit as part of the data string after a successful decode.
- **bXmitNumSys**: Whether the engine will return the numeric system digit of the UPC label.

3.2.10.46 CONFIG_2DSWD_UPCE

```
public struct CONFIG_2DSWD_UPCE
{
  bool  bAddenda2Digit;
  bool  bAddenda5Digit;
  bool  bAddendaReq;
  bool  bAddendaSeparator;
  bool  bE0Enabled;
  bool  bE1Enabled;
  bool  bExpandVersionE;
  bool  bXmitCheckDigit;
  bool  bXmitNumSys;
}
```

- **bAddenda2Digit**: Whether the engine will look for a 2 digit addendum at the end of the UPC bar code. If TRUE, and an addendum is present, the engine adds the two digit addenda data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddenda5Digit**: Whether the engine will look for a 5 digit addendum at the end of the UPC bar code. If TRUE, and an addendum is present, the engine adds the five digit addenda data to the end of the message. If FALSE, the engine ignores addenda data.
- **bAddendaReq**: Whether engine will decode only UPC bar codes that have a 2 or 5 digit addendum. If TRUE, the engine decodes only UPC symbols with an addendum. If FALSE, the engine decodes all enabled UPC symbols.
- **bAddendaSeparator**: Whether there is a space character between the data from the bar code and the data from the addendum.
- **bXmitCheckDigit**: Whether the engine will return the check digit as part of the data string after a successful decode.
- **bXmitNumSys**: Whether the engine will return the numeric system digit of the UPC label.

3.3 Barcode Scanner Methods

Many of the barcode scanner API functions are the same, regardless of whether you are scanning 1D or 2D barcodes. See the next section for functions specific to 2D barcode scanning, such as the preview window, and camera-like image capture functions.

3.3.1 BarcodeApi()

```
BarcodeApi barcode = new BarcodeApi()
```

This is the constructor method for the main barcode scanner class. All subsequent API operations are performed by calling methods of your BarcodeApi object.

Parameters:
None

Returns:
None

3.3.2 Open()

```
BARCODE_RESULT Open()
```

Supplies power to the scanner device, and initializes the scanner device. Allocates all necessary system resources and opens a communication channel.

Parameters:
None

Returns:
BARCODE_RESULT_SUCCESS if successfully executed.

3.3.3 Close()

```
BARCODE_RESULT Close()
```

Removes power from the scanner module and frees all the assigned system resources associated with it.

Parameters:
None

Returns:
BARCODE_RESULT_SUCCESS if successfully executed.

3.3.4 IsOpened()

```
bool IsOpened()
```

Checks whether the scanner device is already opened or not.

Parameters:
None

Returns:
TRUE if the Scanner is already open
FALSE if the Scanned is not opened

3.3.5 Start()

```
BARCODE_RESULT Start()
```

Starts the barcode scanner.

Parameters:
None

Returns:
BARCODE_RESULT_SUCCESS if barcode scanning was successfully started.

3.3.6 Stop()

```
BARCODE_RESULT Stop()
```

Stops the barcode scanner.

Parameters:
None

Returns:
BARCODE_RESULT_SUCCESS if barcode scanning was successfully stopped.

3.3.7 GetBarcodeData()

```
BARCODE_RESULT GetBarcodeData(ref string BarcodeValue, ref string
             BarcodeTypeName, ref string BarcodeTypeId)
```

Retrieves the barcode information which was successfully decoded by the barcode scanner. This function must be called from within registered delegate function by user.

Parameters:

```
BarcodeValue
  [OUT] string parameter which is stored barcode value.
  Pass by reference by using the ref keyword.
BarcodeTypeName
  [OUT] string parameter which is stored barcode Type Name.
  Pass by reference by using the ref keyword.
BarcodeTypeId
  [OUT] string parameter that will store barcode Type Id.
  Pass by reference by using the ref keyword.
```

Returns:

BARCODE_RESULT_SUCCESS if the scanned barcode was retrieved properly.

3.3.8 GetVersionInfo()

```
BARCODE_RESULT GetVersionInfo(ref string ScannerVersion)
```

Provides the barcode scanner's version information.

Parameters:

```
ScannerVersion
  [OUT] string parameter that will store the scanner's version information.
  Pass by reference using the ref keyword.
```

Returns:

BARCODE_RESULT_SUCCESS if the scanner version information was retrieved successfully.

3.3.9 SetCallback()

```
BARCODE_RESULT SetCallback(BARCODECALLBACK  pFunc)
```

When the scanner has data or a message for your application, it calls the delegate function that you provide. You register your callback delegate with the SetCallback() method.

Parameters:

```
pFunc
  [IN]  the BARCODECALLBACK function pointer for your callback delegate.
```

Returns:

BARCODE_RESULT_SUCCESS if the delegate function was registered properly.

3.3.10 GetEnableStateAll()

```
BARCODE_RESULT GetEnableStateAll(ref bool[] bSymbolTable)
```

This method is used to get the enabled/disabled state of all symbologies.

Parameters:

```
bSymbolTable
  [OUT] bool array that contains the enabled state of all symbologies
```

Returns:

BARCODE_RESULT_SUCCESS is returned if the function executed successfully.

3.3.11 EnableSymbologiesAll()

```
BARCODE_RESULT EnableSymbologiesAll(bool bEnable)
```

This method is used to set enable/disable all symbologies.

Parameters:

`bSymbolTable`

```
[IN]  TRUE: all symbologies are set to the enabled state.
      FALSE: all symbologies are set to the disabled state.
```

Returns:

`BARCODE_RESULT_SUCCESS` is returned if the function executed successfully.

3.3.12 EnableSymbologies()

```
BARCODE_RESULT EnableSymbologies(bool[] bSymbolTable)
```

This method is used to enable specific symbologies.

Parameters:

`bSymbolTable`

```
[IN] bool array that contains the symbologies to be enabled
```

Returns:

`BARCODE_RESULT_SUCCESS` is returned if the function executed successfully.

3.3.13 DisableSymbologies()

```
BARCODE_RESULT DisableSymbologies(bool[] bSymbolTable)
```

This method is used to disable specific symbologies.

Parameters:

`bSymbolTable`

```
[IN] bool array that contains the symbologies to be disabled
```

Returns:

`BARCODE_RESULT_SUCCESS` is returned if the function executed successfully.

3.3.14 GetSymbologyConfig()

```
BARCODE_RESULT GetSymbologyConfig(ref SYMBOL_CONFIG SymbolConfig)
```

This method is used to get configuration information of a specified symbology.

Parameters:

`SymbolConfig`

```
[OUT] SYMBOL_CONFIG variable contains IntPtr of specified symbol
configuration structure.
```

Returns:

`BARCODE_RESULT_SUCCESS` is returned if the function executed successfully.

3.3.15 SetSymbologyConfig()

```
BARCODE_RESULT SetSymbologyConfig(SYMBOL_CONFIG SymbolConfig)
```

This method is used to set configuration information of a specified symbology.

Parameters:

`SymbolConfig`

```
[IN] SYMBOL_CONFIG variable contains IntPtr of specified symbol
configuration structure.
```

Returns:
BARCODE_RESULT_SUCCESS is returned if the function executed successfully.

3.3.16 GetGeneralConfig()

```
BARCODE_RESULT GetGeneralConfig(ref GENERAL_CONFIG GeneralConfig)
```

This method is used to get general configuration of the 1D barcode scanner. This is only supported on units with a 1D barcode scanner.

Parameters:
`GeneralConfig`
`[OUT] GENERAL_CONFIG variable`

Returns:
BARCODE_RESULT_SUCCESS is returned if the function executed successfully.

3.3.17 SetGeneralConfig()

```
BARCODE_RESULT SetGeneralConfig(GENERAL_CONFIG GeneralConfig)
```

This method is used to set general configuration of the 1D barcode scanner. This is only supported on units with a 1D barcode scanner.

Parameters:
`GeneralConfig`
`[IN] GENERAL_CONFIG variable`

Returns:
BARCODE_RESULT_SUCCESS is returned if the function executed successfully.

3.3.18 SetDefaultSymbol()

```
BARCODE_RESULT SetDefaultSymbol()
```

This method is used to set default configuration values of the 1D barcode scanner. This is only supported on units with a 1D barcode scanner.

Parameters:
None

Returns:
BARCODE_RESULT_SUCCESS is returned if the function executed successfully.

3.4 2D Barcode Scanner Methods

NOTE: The following methods are only supported on the 2D barcode scanner installed on the **ALH-9001/9011** handhelds. The **ALH-9000/9010** handhelds only support the common methods described above.

3.4.1 StartScanRawData()

```
BARCODE_RESULT StartScanRawData()
```

Starts the 2D barcode scanner. Scanned results are delivered as a byte array, instead of a string.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if barcode scanning was successfully started.

3.4.2 StopScanRawData()

```
BARCODE_RESULT StopScanRawData()
```

Stops the barcode scanning of raw data.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if barcode scanning was successfully stopped.

3.4.3 GetBarcodeRawData()

```
BARCODE_RESULT GetBarcodeRawData(ref byte[] BarcodeValue, ref int Length, ref
            string BarcodeTypeName, ref string BarcodeTypeId)
```

Retrieves the raw barcode information (as a byte array) which was successfully decoded by the barcode scanner. This function must be called from within registered delegate function by user.

Parameters:

```
BarcodeValue
  [OUT] byte array parameter which stores the barcode value.
  Pass by reference by using the ref keyword.
Length
  [OUT] length of stored raw data.
  Pass by reference by using the ref keyword.
BarcodeTypeName
  [OUT] string parameter which stores the barcode Type Name.
  Pass by reference by using the ref keyword.
BarcodeTypeId
  [OUT] string parameter which stores the barcode Type Id.
  Pass by reference by using the ref keyword.
```

Returns:
`BARCODE_RESULT_SUCCESS` if the scanned barcode was retrieved properly.

3.4.4 SetPreviewHwnd()

```
BARCODE_RESULT SetPreviewHwnd(IntPtr pHandle)
```

If you would like to see a preview window of images through the 2D scanner, call SetPreviewHwnd with the handle of your preview window.

Parameters:

```
pHandle
  [IN] preview window screen’s windows handle
```

Returns:
`BARCODE_RESULT_SUCCESS` if the Windows handle is registered properly.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

3.4.5 InitCapture()

```
BARCODE_RESULT InitCapture()
```

In order to capture images through the 2D scanner, you must first assigned the necessary system resources and initialize related parameters, using the InitCapture() method.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if the capture initializes properly.
`BARCODE_RESULT_FAILURE` if the operation failed.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

3.4.6 DeinitCapture()

```
BARCODE_RESULT DeinitCapture()
```

After scanning images, use this method to clear the assigned system resources.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if the system resources cleared properly.
`BARCODE_RESULT_FAILURE` if failed.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

3.4.7 StartPreview()

```
BARCODE_RESULT StartPreview()
```

Draws the image data that is transmitted from the scanner to the preview window screen.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if the preview window started properly.
`BARCODE_RESULT_FAILURE` if failed.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

3.4.8 StopPreview()

```
BARCODE_RESULT StopPreview()
```

Stops to the preview operation.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if stop to preview window execute properly.
`BARCODE_RESULT_UNSUPPORTED` if call is made to a 1D scanner.

3.4.9 PausePreview()

```
BARCODE_RESULT PausePreview()
```

Pauses the preview operation.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if pause preview window execute properly.
`BARCODE_RESULT_UNSUPPORTED` if call is made to a 1D scanner.

3.4.10 ResumePreview()

```
BARCODE_RESULT ResumePreview()
```

Resumes the preview operation.

Parameters:
None

Returns:
`BARCODE_RESULT_SUCCESS` if the resume preview executes properly.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

3.4.11 DoCapture()

```
BARCODE_RESULT DoCapture(ref BARCODECAPTUREPARAMS CaptureParams)
```

Captures image data from the scanner and stores it into files.

Parameters:
`CaptureParams`
[IN] transmit image’s resolution, format, and storage path.

Returns:
`BARCODE_RESULT_SUCCESS` if the image is captured successfully.
`BARCODE_RESULT_UNSUPPORTED` if the call is made to a 1D scanner.

4 GPS

NOTE: Data structures and methods described in this section only apply to the **ALH-9001** handhelds. Control of the GPS module on the **ALH-9011** handhelds is accomplished by using the standard Windows Mobile .NET Compact Framework Class Libraries.

4.1 Introduction

The Alien ALH-9001/9011 handheld readers include a GPS receiver which you can use to determine the approximate position of the handheld just about anywhere on the surface of the earth (or above it!). The receiver must be powered on, initialized, and then given time to acquire the extremely weak signals from at least three orbiting satellites, before it will provide you with useful data. Once it has acquired a fix it will stream data to your application at regular intervals.

GPS stands for Global Positioning System, and requires unimpeded line-of-sight with three or four satellites, for proper operation. Buildings, mountains, trees, and even cloud-cover can degrade the GPS signal and even prevent the receiver from determining a valid location.

4.2 GPS Data Structures

4.2.1 GPS_RESULT

Type	Item	Description
GPS_RESULT	GPS_RESULT_SUCCESS	operation successfully performed
	GPS_RESULT_INVALID_ARGS	invalid parameter
	GPS_RESULT_OUTOFMEMORY	failed to assign a memory resource
	GPS_RESULT_UNSUPPORTED	command not currently supported
	GPS_RESULT_ALREADY_OPENED	device has already been opened
	GPS_RESULT_NOT_OPENED	function called w/o calling Open() first
	GPS_RESULT_FAILURE	failed to perform operation
	GPS_RESULT_INVALID_DEVICE	device not installed

4.2.2 GPSCALLBACK

This is a delegate function that is called when GPS data is received from the receiver. In order to process data in your application, use the **`SetCallbackFunc(GPSCALLBACK GpsCallback)`** function to assign your delegate function. Raw NMEA data is returned to your callback through the sData string parameter that you pass in.

```
public delegate void GPSCALLBACK(string sData);
```

Each time the system calls your delegate, it provides you with one line of data from the GPS receiver. The receiver outputs data in the form of NMEA (National Marine Electronics Association) “sentences”, which you must parse to extract the useful information. A good reference for the structure of these sentences can be found here:

http://www.gpsinformation.org/dale/nmea.htm

4.2.3 PORT

This enumeration specifies which port to use when connecting to the GPS device.

```
public enum PORT
{
  COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, COM10
}
```

4.2.4 BAUDRATE

The baud rate is the serial communication rate used when communicating with the GPS device. The default value is 9,600 baud.

```
public enum BAUDRATE
{
  BR_2400,
  BR_4800,
  BR_9600,
  BR_14400,
  BR_19200,
  BR_38400,
  BR_56000,
  BR_115200,
  BR_256000
}
```

4.3 GPS API Reference

4.3.1 GpsApi()

```
GpsApi gps = new GpsApi()
```

This is the constructor method for the main GPS class. All subsequent API operations are performed by calling methods on the GpsApi object.

Parameters:
None

Returns:
None

4.3.2 Open()

```
GPS_RESULT Open(PORT gpsPort, BAUDRATE baudrate)
```

Applies power to the GPS device, initializes it, and opens a serial connection. The gpsPort should always be COM6, and the baud rate should always be 9,600. After this function is called, the system will perform a delegate callback to the registered callback function every time it receives GPS data (about once per second).

Parameters:
gspPort
  [IN] the serial port name where the GPS receiver is located
gpsBaud
  [IN] baud rate that is to communicate serially

Returns:
GPS_RESULT_SUCCESS if executed successfully

4.3.3 Close()

```
GPS_RESULT Close()
```

Powers down the GPS receiver and frees up the serial interface and other resources.

Parameters:
None

Returns:
GPS_RESULT_SUCCESS if executed successfully

4.3.4 IsOpen()

```
bool       IsOpen()
```

The IsOpen() function reports whether the GPS device has already been opened.

Parameters:
None

Returns:
TRUE if GPS device is open
FALSE if GPS device is not open

4.3.5 SetCallbackFunc()

```
GPS_RESULT SetCallbackFunc(GPSCALLBACK gpsCallback)
```

The SetCallbackFunc() function registers a specific callback function to receive data from the GPS receiver.

Parameters:
gpsCallback
[IN] the callback delegate to receive NMEA data

Returns:
GPS_RESULT_SUCCESS if executed successful

5 Camera

NOTE: Data structures and methods described in this section only apply to the **ALH-9001** handhelds.
Control of the Camera module on the **ALH-9011** handhelds is accomplished by using the standard Windows Mobile .NET Compact Framework Class Libraries.

The camera installed in the ALH-9001/9011 handhelds allows you to take snapshots at various resolutions (from 160x120 to 2048x1536) and quality settings, with full control over contrast, white balance, saturation, exposure, and various digital effects. You can preview images, and captured images are stored directly in the filesystem.

5.1 Camera Data Structures

5.1.1 CAM_RESULT

Type	Item	Description
CAM_RESULT	CAM_RESULT_SUCCESS	operation successfully performed
	CAM_RESULT_INVALID_ARGS	invalid parameter
	CAM_RESULT_OUTOFMEMORY	failed to assign a memory resource
	CAM_RESULT_UNSUPPORTED	command not currently supported
	CAM_RESULT_ALREADY_OPENED	device has already been opened
	CAM_RESULT_NOT_OPENED	function called w/o calling Open() first
	CAM_RESULT_FAILURE	failed to perform operation
	CAM_RESULT_INVALID_DEVICE	device not installed

5.1.2 CAMERACALLBACK

This is a delegate function that is called when image data is received from the camera module. In order to process data in your application, use the **`SetCallbackFunc(CAMERACALLBACK CamCallback)`** function to assign your delegate function.

```
public delegate void CAMERACALLBACK(CAMERAMSG camMsg)
```

5.1.3 CAM_COL_SP

```
public enum CAM_COL_SP {
  Raw,
  RGB,
  YCbCr
}
```

5.1.4 CAM_BPP

```
public enum CAM_BPP {
  RawRGB8Bits,
  RGB565,
  YCbCr422
}
```

5.1.5 CAM_RESOL

```
public enum CAM_RESOL {
  QQVGA=0, // {160, 120}
  QVGA,    // {320, 240}
  CIF,     // {352, 288}
  VGA,     // {640, 480}
  SVGA,    // {800, 600}
  XGA,     // {1024, 768}
  SXGA,    // {1280, 960}
  UXGA,    // {1600, 1200}
  QXGA,    // {2048, 1536}
}
```

5.1.6 FLIP_MIRROR

```
public enum FLIP_MIRROR {
  NO_MIRROR_,
  MIRR,
  FLIP,
  FLIP_AND_MIRROR,
  MIRR_FLIP_UNSUPPORTED
}
```

5.1.7 WHITE_BAL

Color Temperature value.

```
public enum WHITE_BAL {
  WB_AUTO,
  WB_CLOUDY,
  WB_SUNNY,
  WB_FLUORESCENT,
  WB_INCANDESCENT,
  WB_UNSUPPORTED
}
```

5.1.8 SAT_OPTIONS

```
public enum SAT_OPTIONS {
  SAT_0X,
  SAT_025X,
  SAT_05X,
  SAT_075X,
  SAT_1X,
  SAT_125X,
  SAT_15X,
  SAT_175X,
  SAT_2X,
}
```

5.1.9 PM_OPTIONS

```
public enum PM_OPTIONS {
  PM_NORMAL,
  PM_SPOT,
}
```

5.1.10 EV_OPTIONS

Exposure value.

```
public enum EV_OPTIONS {
  EV_N1D7,
  EV_N1D3,
  EV_N1D0,
  EV_N0D7,
  EV_N0D3,
  EV_0,
  EV_P0D3,
  EV_P0D7,
  EV_P1D0,
  EV_P1D3,
  EV_P1D7
}
```

5.1.11 CONT_OPTIONS

Contrast value.

```
public enum CONT_OPTIONS {
  CONT_N5,
  CONT_N4,
  CONT_N3,
  CONT_N2,
  CONT_N1,
  CONT_0,
  CONT_P1,
  CONT_P2,
  CONT_P3,
  CONT_P4,
  CONT_P5
}
```

5.1.12 EFFECT

Digital filter.

```
public enum EFFECT {
  NO_EFFECT,
  EF_SEPIA,
  EF_MONO,
  EF_NEGATIVE,
  EF_RED,
  EF_BLUE,
  EF_GREEN,
  EF_VIOLET
}
```

5.1.13 SENSOR_SETUP_INFO

This struct contains all of the setup information for the sensor.

```
public struct SENSOR_SETUP_INFO {
  CAM_COL_SP   ColorSpace;
  CAM_BPP      bpp;
  CAM_RESOL    Resol;
  int          Zoom;
  WHITE_BAL    WhiteBalance;
  EV_OPTIONS   Exposure;
  EFFECT       Effect;
  PM_OPTIONS   PhotoMetry;
  FLIP_MIRROR  FlipMirror;
  int          Brightness;
  CONT_OPTIONS Contrast;
  SAT_OPTIONS  Saturation;
  bool         bAutoFocus;
  bool         bEnableAF;
  bool         bSupportFlash;
}
```

5.1.14 SEN_CTRL_CODE

```
public enum SEN_CTRL_CODE {
  SEN_CTRL_WHITE_BALANCE,
  SEN_CTRL_EXPOSURE,
  SEN_CTRL_BRIGHTNESS,
  SEN_CTRL_EFFECT,
  SEN_CTRL_PHOTOMETRY,
  SEN_CTRL_FLIP_MIRROR,
  SEN_CTRL_CONTRAST,
  SEN_CTRL_SATURATION,
  SEN_CTRL_AUTOFOCUS
}
```

5.1.15 IMG_ENCODE_FORMAT

Image encoding format.

```
public enum IMG_ENCODE_FORMAT {
  IMG_ENCODE_BMP,
  IMG_ENCODE_JPG,
  IMG_ENCODE_GIF,
  IMG_ENCODE_TIFF,
  IMG_ENCODE_EXIF,
  IMG_ENCODE_ICON
}
```

5.1.16 CAPTURE_SETUP_INFO

The structure of the necessary information while capture image.

```
public struct CAPTURE_SETUP_INFO {
  SENSOR_SETUP_INFO  SensorSetup;
  IMG_ENCODE_FORMAT  ImgFormat;
  TCHAR              pSavePath[MAX_PATH];
  int                nJpegQuality;
}
```

5.1.17 stZOOM_CAPA

The structure of information of zoom function.

```
public struct stZOOM_CAPA {
  bool         bZoomSupport;
  unsigned int MaxZoom;
  int          ZoomStep;
}
```

5.1.18 stEXPOSURE

Exposure information structure.

```
public struct stEXPOSURE {
  int        ExposureDefault;
  signed int ExposureMin;
  signed int ExposureMax;
  char       ExposureSteps;
}
```

5.1.19 stCONTRAST

Contrast information structure.

```
public struct stCONTRAST {
  int        ContrastDefault;
  signed int ContrastMin;
  signed int ContrastMax;
  char       ContrastSteps;
}
```

5.1.20 stSATURATION

```
public struct stSATURATION {
  int        SaturationDefault;
  signed int SaturationMin;
  signed int SaturationMax;
  char       SaturationSteps;
}
```

5.1.21 stBRIGHTNESS

Brightness Information structure.

```
public struct stBRIGHTNESS {
  int        BrightDefault;
  signed int BrightMin;
  signed int BrightMax;
  char       BrightSteps;
}
```

5.1.22 SENSOR_CAPA

The structure contains the entire capability information of the camera.

```
public struct SENSOR_CAPA {
  DWORD        ColorSpace;
  DWORD        bpp;
  DWORD        MaxResol;
  stZOOM_CAPA  ZoomCapa;
  DWORD        WhiteBalance;
  stEXPOSURE   Exposure;
  DWORD        Effect;
  DWORD        PhotoMetry;
  DWORD        FlipMirror;
  stBRIGHTNESS Brightness;
  stCONTRAST   Contrast;
  stSATURATION Saturation;
  bool         bSupportAF;
  bool         bSupportFlash;
}
```

5.1.23 CAMERAMSG

The camera API can generate messages to your application programs according to the status of the image sensor. Each type of message is identified by one of the CAMERAMSG enumerations.

```
public enum CAMERAMSG{
  CAMERA_INITIALIZE,
  CAPTURE,
  COMPLETE,
  PRIVIEW_DONE
  START_TO_SAVE
}
```

- **CAMERA_INITIALIZE**: indicates the image sensor's initialization operation has completed.
- **CAPTURE**: indicates image processing has completed.
- **COMPLETE**: indicates complete operation.
- **PREVIEW_DONE**: indicates a preview is ready.
- **START_TO_SAVE**: indicates starting to save captured images.

5.2 Camera Methods

The camera API gives you the ability to control power to the camera module, preview image data, capture image data, and even adjust the resolution, white-balance, exposure, and apply digital effects. You can save the image in various formats. Before opening the connection to the camera, you should call SetHandle() and SetCallBack() first, to register a windows handle to receive image previews and a function to receive callbacks.

5.2.1 CamApi()

```
CamApi camera = new CamApi()
```

This is the constructor method for the main Camera class. All subsequent API operations are performed by calling methods on the CamApi object.

Parameters:
None

Returns:
None

5.2.2 Open()

```
CAM_RESULT Open()
```

Opens the camera device, assigns necessary system resources, and opens a communications channel.

Parameters:
None

Returns:
`CAM_RESULT_SUCCESS` if the camera device opened properly.

5.2.3 Close()

```
CAM_RESULT Close()
```

Removes the assigned system resources and closes the camera device.

Parameters:
None

Returns:
`CAM_RESULT_SUCCESS` if the camera device closed properly.

5.2.4 IsOpen()

```
bool IsOpen()
```

Reports whether the camera has been initialized and the communication channel is open.

Parameters:
None

Returns:
TRUE if the camera is open.
FALSE if the camera is closed.

5.2.5 StartPreview()

```
CAM_RESULT StartPreview()
```

Draws the image data received from the camera module to the preview window given during the call to SetHandle().

Parameters:
None

Returns:
CAM_RESULT_SUCCESS if the preview started successfully.

5.2.6 StopPreview()

```
CAM_RESULT StopPreview()
```

Stops the preview.

Parameters:
None

Returns:
CAM_RESULT_SUCCESS if the preview stopped successfully

5.2.7 Capture()

```
CAM_RESULT Capture(CAPTURE_SETUP_INFO *pCamCaptureParams)
```

Takes a picture with the built-in camera, according to the settings given in the CAPTURE_SETUP_INFO parameter. The CAPTURE_SETUP_INFO structure contains information about the camera settings, and desired image file type, filename, and path.

Parameters:
pCamCaptureParams
[IN] the camera settings and resulting file format information.

Returns:
CAM_RESULT_SUCCESS if the image capture operation successfully performed.

5.2.8 GetInfo()

```
CAM_RESULT GetInfo(SENSOR_SETUP_INFO *pCamInfo)
```

Gets the current settings of the camera module.

Parameters:
`pCamInfo`
`[OUT] pointer to your structure where the current settings will be copied.`

Returns:
`CAM_RESULT_SUCCESS` if the setting were properly retrieved.

5.2.9 GetSensorCapa()

```
CAM_RESULT GetSensorCapa(SENSOR_CAPA * pCamCapa)
```

Returns the supported capabilities of the camera module. In order to change the settings of the camera, you should first know the supported values of the camera module, by using this function.

Parameters:
`pCamCapa`
`[OUT] pointer to your struct where the camera capabilities are copied.`

Returns:
`CAM_RESULT_SUCCESS` if the camera capabilities successfully retrieved.

5.2.10 SetHandle()

```
void SetHandle(IntPtr pPicboxhWnd)
```

Registers the windows handle (pPicboxhWnd) that the application program will use for drawing camera previews.

Parameters:
`hWnd`
`[IN] windows handle to receive windows message to your application.`
`hPrevWnd`
`[IN] windows handle for drawing preview images in the application.`

Returns:
None

5.2.11 ZoomPreview()

```
CAM_RESULT ZoomPreview(DWORD nZoomStep)
```

Sets the zoom level of the preview screen. Zooming in on the preview screen (using higher values for nZoomStep) will lower the image quality of the images.

Parameters:
`nZoomStep`
`[IN] the level of zoom when taking pictures.`

Returns:
`CAM_RESULT_SUCCESS` if performed zoom function peroperly.

5.2.12 SetWhiteBalance()

```
CAM_RESULT SetWhiteBalance(SENSOR_SETUP_INFO *CamParams)
```

Changes the white balance value of the camera.

Parameters:
`CamParams`
`[IN]  the SENSOR_SETUP_INFO struct with the new white balance value.`

Returns:
CAM_RESULT_SUCCESS if the White balance changed successfully.

5.2.13 SetContrast()

```
CAM_RESULT SetContrast(SENSOR_SETUP_INFO *CamParams)
```

Changes the contrast value of the camera.

Parameters:
CamParams
[IN] the SENSOR_SETUP_INFO struct with the new contrast value.

Returns:
CAM_RESULT_SUCCESS if changed the contrast value perperly.

5.2.14 SetSaturation()

```
CAM_RESULT SetSaturation(SENSOR_SETUP_INFO *CamParams)
```

Changes the saturation value of the camera.

Parameters:
CamParams
[IN] the SENSOR_SETUP_INFO struct with the new saturation value.

Returns:
CAM_RESULT_SUCCESS if changed saturation properly.

5.2.15 SetBrightness()

```
CAM_RESULT SetBrightness(SENSOR_SETUP_INFO *CamParams)
```

Changes brightness value of the camera.

Parameters:
CamParams
[IN] the SENSOR_SETUP_INFO struct with the new brightness value.

Returns:
CAM_RESULT_SUCCESS if changed the brightness value properly.

5.2.16 SetEffect()

```
CAM_RESULT SetEffect(SENSOR_SETUP_INFO *CamParams)
```

Applies a digital filter effect to the image.

Parameters:
CamParams
[IN] the SENSOR_SETUP_INFO struct with the new digital effect setting.

Returns:
CAM_RESULT_SUCCESS if applied the digital filter effect properly.

5.2.17 SetFlip()

```
CAM_RESULT SetFlip(SENSOR_SETUP_INFO *CamParams)
```

Applies a Flip/Mirror effect to the image.

Parameters:

`CamParams`

```
[IN]  the SENSOR_SETUP_INFO struct with the new mirror/flip value.
```

Returns:

`CAM_RESULT_SUCCESS` if applied Flip/Mirror effects properly.

5.2.18 SetAutoFocus()

```
CAM_RESULT SetAutoFocus(SENSOR_SETUP_INFO *CamParams)
```

When Autofocus is enabled, the camera automatically performs the focus adjustment.

Parameters:

`CamParams`

```
[IN]  the SENSOR_SETUP_INFO struct with the new autofocus value.
```

Returns:

`CAM_RESULT_SUCCESS` if performed successfully.

6 Bluetooth

NOTE: Data structures and methods described in this section only apply to the **ALH-9001** handhelds. Control of the Bluetooth module on the **ALH-9011** handhelds is accomplished by using the standard Windows Mobile .NET Compact Framework Class Libraries.

This API only provides functions for turning the Bluetooth module on and off. The configuration of the Bluetooth module, including discovering, pairing with, and connecting to devices is handled using the WinCE Bluetooth Device Properties control panel, and standard Windows APIs. A connected Bluetooth device is accessed through the COM3 serial port. If you have connected to a device in the MASTER mode, even resetting the handheld or Bluetooth module will not break the MASTER/SLAVE pairing, and they will reconnect automatically the next time.

6.1 Bluetooth Data Structures

6.1.1 BT_RESULT

Type	Item	Description
BT_RESULT	BT_RESULT_SUCCESS	operation successfully performed
	BT_RESULT_FAILURE	failed to perform operation
	BT_RESULT_INVALID_DEVICE	device not installed
	BT_RESULT_INVALID_ARGS	invalid parameter
	BT_RESULT_OUTOFMEMORY	failed to assign memory resources
	BT_RESULT_ALREADY_OPENED	the Bluetooth port is already open
	BT_RESULT_NOT_OPENED	called a function without opening first

6.2 Bluetooth Methods

6.2.1 BluetoothApi ()

```
BluetoothApi bt = new BluetoothApi()
```

This is the constructor method for the main Bluetooth class. All subsequent API operations are performed by calling methods on the BluetoothApi object.

Parameters:
None

Returns:
None

6.2.2 PowerEnable()

```
BT_RESULT  PowerEnable(bool isEnable)
```

Turns on or off the Bluetooth module's power.

Parameters:

```
isEnable
  [IN]  TRUE : power on
        FALSE: power off
```

Returns:
`BT_RESULT_SUCCESS` if called successfully.

6.2.3 GetPowerEnable()

```
BT_RESULT  GetPowerEnable(ref bool isEnabled)
```

Gets the on/off state of the Bluetooth module’s power.

Parameters:
`isEnabled`
`[IN]  parameter which will hold the power state value (as a ref type).`

Returns:
`BT_RESULT_SUCCESS` if called successfully.

7 System Services

The System Services API provides control over many aspects or the ALH-90xx handheld's hardware, sleep timeouts, WLAN radio, CPU clock speed, OS firmware version, etc.

7.1 System Data Structures

7.1.1 SYSSVC_RESULT

Type	Item	Description
SYSSVC_RESULT	SYSSVC_RESULT_SUCCESS	operation successfully performed
	SYSSVC_RESULT_INVALID_ARGS	invalid parameter
	SYSSVC_RESULT_OUTOFMEMORY	failed to assign a memory resource
	SYSSVC_RESULT_UNSUPPORTED	command not currently supported
	SYSSVC_RESULT_FAILURE	failed to perform operation

7.1.2 VERSION_INFOS

(under development)

7.1.3 AUDIOCODECVOLCTL

(under development)

7.1.4 IPM_PRODUCT_OP

(under development)

7.1.5 KBD_STATUS

(under development)

7.2 System Service Methods

7.2.1 SysSvcApi ()

```
SysSvcApi sys = new SysSvcApi()
```

This is the constructor method for the main SysSvc class. All subsequent API operations are performed by calling methods on the SysSvcApi object.

Parameters:
None

Returns:
None

7.3 Backlight

These methods allow you to control the brightness of the screen's backlight, as well as the idle timeout before the backlight is dimmed. The timeout and brightness level can be set separately for the two power modes, AC power and battery power.

7.3.1 BacklightWriteTimeoutValue()

```
SYSSVC_RESULT BacklightWriteTimeoutValue(byte cMode, UInt16 nSeconds)
```

Sets the timeout period (in seconds) before backlight is turned off, for each of the power modes (AC or battery). If the user doesn't perform any operations within the timeout period, such as tapping the touchscreen or pressing a button, then the backlight will turn off. The timer will be reset, if the user performs an operation within the timeout period.

Parameters:

cMode

[IN] Specifies the power mode (MODE_AC or MODE_BAT) for the new timeout.

nSeconds

[IN] Sets the value of the backlight timeout. The value must be 0-60 seconds in MODE_BAT, and 0-300 seconds in MODE_AC. A value of 0 prevents the backlight from turning off.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

SYSSVC_RESULT_INVALIDARG if the value of *cMode* is not MODE_BAT or MODE_AC, or if the value of *nSeconds* is out of range.

7.3.2 BacklightReadTimeoutValue()

```
SYSSVC_RESULT BacklightReadTimeoutValue(bye cMode, ref UInt16 nSeconds)
```

Gets the value of currently backlight timeout for a given power mode (AC or battery).

Parameters:

cMode

[IN] Specifies which power mode to get the timeout value for.

nSeconds

[OUT] The reference UInt16 parameter to receive the timeout value.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

SYSSVC_RESULT_INVALIDARG if the value of *cMode* is not MODE_BAT or MODE_AC.

7.3.3 BacklightSetBrightness()

```
SYSSVC_RESULT BacklightSetBrightness(UInt16 nBrightness)
```

Sets the value of the backlight brightness in the current power mode.

Parameters:

nBrightness

[IN] The desired backlight brightness value (0-100).

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

SYSSVC_RESULT_INVALIDARG if the value of *nBrightness* is out of range.

7.3.4 BacklightGetBrightness()

```
SYSSVC_RESULT BacklightGetBrightness(ref UInt16 nBrightness)
```

Gets the value of the backlight brightness in the current power mode.

Parameters:

nBrightness

[OUT] The reference UInt16 parameter to receive the current brightness value.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.3.5 BacklightReadBrightness()

```
SYSSVC_RESULT BacklightReadBrightness(byte cMode, UInt16 nBrightness)
```

Gets the value of the backlight brightness for the specified power mode (AC or Battery).

Parameters:
cMode
[IN] Specifies the power mode (MODE_AC or MODE_BAT) for the desired backlight brightness value.
nBrightness
[OUT] The reference UInt16 parameter to receive the brightness value*.*

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.
SYSSVC_RESULT_INVALIDARG if the value of *cMode* is not MODE_BAT or MODE_AC.

7.3.6 BacklightWriteBrightness()

```
SYSSVC_RESULT BacklightWriteBrightness(byte cMode, UInt16 nBrightness)
```

Sets the value of the backlight brightness for the specified power mode (AC or Battery).

Parameters:
cMode
[IN] Specifies the power mode (MODE_AC or MODE_BAT) for the new backlight brightness value.
nBrightness
[IN] The desired backlight brightness value (0-100) for the specified mode.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.
RESULT_INVALIDARG if out of the range (0-100).

7.4 Key Lamp

Alien handheld has a Key Lamp, which lights up the buttons on the front of the unit. You can configure it to automatically turn on the key lamp when the user presses a button, and turn off again after a configurable amount of time. There is an additional feature which allows you to disable the key lamp during certain times of the day, to save battery life when the key lamp is not needed during the day. You also have the ability to programmatically turn the key lamp on and off whenever you want.

7.4.1 KeyLampTurnOn ()

```
SYSSVC_RESULT KeyLampTurnOn()
```

Turns the Key Lamp on.

Parameters:
None

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.2 KeyLampTurnOff ()

```
SYSSVC_RESULT KeyLampTurnOff()
```

Turns the Key Lamp off.

Parameters:
None

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.3 KeyLampWriteTimeoutValue ()

```
SYSSVC_RESULT KeyLampWriteTimeoutValue(UInt16 nSeconds)
```

Sets the value of the Key Lamp timeout. After a button is pressed, the Key Lap stays on for this amount of time. Pressing a button during the timeout period resets the timer.

Parameters:
nSeconds
[IN] The time (in seconds) to leave the Key Lamp on, after a button press.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.4 KeyLampReadTimeoutValue ()

```
SYSSVC_RESULT KeyLampReadTimeoutValue(ref UInt16 nSeconds)
```

Gets the value of the Key Lamp timeout.

Parameters:
nSeconds
[OUT] The reference UInt16 parameter to receive the timeout setting.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.5 KeyLampWriteOffTimeValue ()

```
SYSSVC_RESULT KeyLampWriteOffTimeValue(Unt16 nHoursFrom, Unt16 nMinutesFrom,
           Unt16 nHoursTo, Unt16 nMinutesTo)
```

Sets the time period during which the Key Lamp will not turn on. You specify a starting hour:minute and an ending hour:minute. For this feature to work, you must also enable it with KeyLampWriteOffTimeEnable().

Parameters:
nHoursFrom
[IN] Specifies the starting time (hour) when the Key Lamp is disabled.
nMinutesFrom
[IN] Specifies the starting time (minutes) when the Key Lamp is disabled.
nHoursTo
[IN] Specifies the ending time (hour) when the Key Lamp is disabled.
nMinutesTo
[IN] Specifies the ending time (minutes) when the Key Lamp is disabled.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.6 KeyLampReadOffTimeValue ()

```
SYSSVC_RESULT KeyLampReadOffTimeValue(ref Unt16 nHoursFrom, ref Unt16
          nMinutesFrom, ref Unt16 nHoursTo, ref Unt16 nMinutesTo)
```

Gets the time period during which the Key Lamp will not turn on.

Parameters:

```
nHoursFrom
  [OUT] The Reference UInt16 type parameter to receive the starting hour.
nMinutesFrom
  [OUT] The Reference UInt16 type parameter to receive the starting minute.
nHoursTo
  [OUT] The Reference UInt16 type parameter to receive the ending hour.
nMinutesTo
  [OUT] The Reference UInt16 type parameter to receive the enging minute.
```

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.7 KeyLampWriteOffTimeEnable ()

```
SYSSVC_RESULT KeyLampWriteOffTimeEnable(bool bEnable)
```

Enables or disables the feature to keep the Key Lamp on during a period of time each day.

Parameters:

```
bEnable
  [IN]  TRUE: enables the Key Lamp off function.
        FALSE: disables the Key Lamp off function.
```

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.4.8 KeyLampReadOffTimeEnable ()

```
SYSSVC_RESULT KeyLampReadOffTimeEnable(ref bool bEnable)
```

Returns the enabled/disabled state of the Key Lamp off feature.

Parameters:

```
pbEnable
  [OUT] The reference bool to receive the enabled/disabled state.
```

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.5 Suspend Time-Out Setting

Alien handheld can be configured to automatically enter a suspended mode, where the system power will be off and only enough power is used to maintain data in memory. If there is no user input and no running program during the timeout period, then the handheld will enter the suspend mode. A different timeout period can be specified for each of the power modes (AC or battery). Press the power button to wake up the system, and the system will be restore to its previous running state.

7.5.1 SuspendWriteTimeoutValue ()

```
SYSSVC_RESULT SuspendWriteTimeoutValue(byte cMode, UInt16 nSeconds)
```

Sets the timeout value (in seconds) for suspend mode, for the specified power mode.

Parameters:

`cMode`

`[IN] Specifies the power mode (MODE_AC or MODE_BAT) for the new timeout.`

`nSeconds`

`[OUT] The value of the new suspend timeout for the given power mode.`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

`SYSSVC_RESULT_INVALIDARG` if the value of *cMode* is not MODE_BAT or MODE_AC.

7.5.2 SuspendReadTimeoutValue ()

```
SYSSVC_RESULT SuspendReadTimeoutValue(byte cMode, ref UInt16 nSeconds)
```

Gets the timeout value for suspend mode, for the specified power mode.

Parameters:

`cMode`

`[IN] Specifies the power mode (MODE_AC or MODE_BAT) of the desired timeout.`

`nSeconds`

`[IN] The reference UInt16 to receive the suspend timeout value.`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

`SYSSVC_RESULT_INVALIDARG` if the value of *cMode* is not MODE_BAT or MODE_AC.

7.6 Software Version Information

This function provides a way to get the version information of the Bootloader, BSP, CE OS component and software APIs currently installed in the handheld.

7.6.1 GetSoftwareVersion()

```
SYSSVC_RESULT GetSoftwareVersion(ref VERSION_INFOS VersionInfos)
```

Gets all of the software version information, populating the VERSION_INFOS parameter that you pass in.

Parameters:

`VersionInfos`

`[OUT] The reference VERSION_INFOS parameter to receive the software version information.`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

7.7 WLAN

7.7.1 SetWlanPowerEnable()

```
SYSSVC_RESULT SetWlanPowerEnable(bool bEnable)
```

Turns the power supply to the WLAN module on or off. After applying voltage, wait until it binds with the WLAN driver.

Parameters:

`bEnable`

`[IN] TRUE to turn on power, FALSE to turn it off`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

`SYSSVC_RESULT_FAILURE` if executed unsuccessfully.

7.7.2 GetWlanPowerStatus()

```
SYSSVC_RESULT GetWlanPowerStatus(ref bool bEnable)
```

Gets the current on/off state of the WLAN module’s power.

Parameters:

`bEnable`

`[OUT] parameter to receive the on/off state of the module power (passed as a ref)`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

7.8 Audio Codec Control

These functions control the hardware gain for the handheld’s microphone and the speaker level. The hardware gain sets the upper limit on the volume of the system speaker.

Note: This section is still under development.

7.8.1 AudioCodecControl()

```
SYSSVC_RESULT GetAudioCodecControl(ref AUDIOCODECVOLCTL AudioCodecVolCtrl)
```

Sets the default gain value of the microphone and speaker.

Parameters:

`AudioCodecVolCtrl`

`[IN] pass the AUDIOCODECVOLCTL type structure parameter, this stores the control information of the gain of the mic, speaker, as ref type.`

Returns:

`SYSSVC_RESULT_SUCCESS` if executed successfully.

7.9 Sleep/Wakeup Notification

These functions provide your application with notifications from the operating system when the handheld is about to enter sleep more, and after it wakes up again. To receive these notifications, you need to first register your callback function with the system, using the SleepWakeupNotificationSet() function.

The function signature for your callback procedure should look like this:

```
public void SleepWakeupNotifyCALLBACK(bool bEnterSleep);
```

7.9.1 SleepWakeupNotificationSet()

```
SYSSVC_RESULT SleepWakeupNotificationSet(SleepWakeupNotifyCALLBACK CallbackProc)
```

Registers your application’s callback function, which will then receive the Sleep/Wakeup notifications.

Parameters:

`CallbackProc`

`[IN] The function that will receive the sleep/wakeup notifications.`

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.9.2 SleepWakeupNotificationReset()

```
SYSSVC_RESULT SleepWakeupNotificationReset()
```

Removes your application's registered sleep/wakeup notifications. Call this to stop receiving these notifications.

Parameters:
None.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.10 Vibrator Control

This function allows you to manually turn the vibrator motor on or off.

7.10.1 SetVibratorEnable()

```
SYSSVC_RESULT SetVibratorEnable(bool bEnable)
```

Enables or disables the Vibrator motor.

Parameters:
`bEnable`
`[IN] TRUE: enable the vibrator motor, FALSE: disable the vibrator motor.`

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.11 CPU clock Setting

These functions allow you to get and set the CPU clock mode. There are four available clock frequencies: 208, 416, 624, and 806 MHz. You can set a specific clock speed, or set the mode to Auto and let the handheld automatically change the clock frequency, according to the CPU load. If you set a fixed clock frequency, the CPU will continue to use that frequency, regardless of the CPU load.

7.11.1 SetCpuClock()

```
SYSSVC_RESULT SetCpuClock(IPM_PRODUCT_OP Clock)
```

Sets the CPU clock to Auto, 208, 416, 624, or 806Mhz.

Parameters:
`Clock`
`[IN] The CPU clock mode which you would like to set.`

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.11.2 GetCpuClock()

```
SYSSVC_RESULT GetCpuClock(ref IPM_PRODUCT_OP Clock)
```

Gets the current setting for the CPU clock mode.

Parameters:

Clock

[IN] The reference IPM_PRODUCT_OP parameter to receive the current CPU clock mode.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.12 Modem Screen Off Function

The ALH-9001’s 3G modem can conceivable be used to make voice calls. When using the handheld as a phone, you normally would like the touchscreen to be off during the call. These functions get and set the timeout value for turning off the screen once a call has been initiated.

7.12.1 SetModemScreenOffTime()

```
SYSSVC_RESULT SetModemScreenOffTime(UInt16 nSeconds)
```

Sets the timeout (in seconds) to turn off the screen after starting a call.

Parameters:

nSeconds

[IN] The timeout value to wait before turning off the screen.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.12.2 GetModemScreenOffTime()

```
SYSSVC_RESULT GetModemScreenOffTime(ref UInt16 nSeconds)
```

Gets the current timeout value for turning off the screen during a call.

Parameters:

nSeconds

[IN] The reference UInt16 parameter to receive the screen-off timeout.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.13 Enter Suspend State Function

This function allows you to programmatically put the handheld to sleep.

7.13.1 EnterSusendState()

```
SYSSVC_RESULT EnterSusendState()
```

Enters the suspend/sleep mode.

Parameters:

None.

Returns:

SYSSVC_RESULT_SUCCESS if executed successfully.

7.14 Soft Reset Function

This function allows you to programmatically perform a soft reset on the handheld. This is the same as the user depressing the recessed reset button on the unit, and reboots the unit.

7.14.1 SoftReset()

```
SYSSVC_RESULT SoftReset()
```

Performs a soft reset of the handheld (i.e. reboot).

Parameters:
None.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.15 KBD Status

These functions allow you to get and set the current mode of the onscreen keyboard - either uppercase, lowercase, or numeric.

7.15.1 SetKBDStatus()

```
SYSSVC_RESULT SetKBDStatus(KBD_STATUS KbdStatus)
```

Sets the mode of the onscreen keyboard to uppercase, lowercase, or numeric.

Parameters:
KbdStatus
[IN] The keyboard’s new text-entry mode.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.15.2 GetKBDStatus()

```
SYSSVC_RESULT GetKBDStatus(ref KBD_STATUS pKbdStatus)
```

Gets the currently mode of the onscreen keyboard.

Parameters:
pKbdStatus
[OUT] The reference KDB_STATUS enumeration parameter to receive the keyboard status value.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.16 Flashlight Control Function

This function allows you to manually turn on and off the white LED flashlight that is part of the built-in camera. This function is only available on the ALH-9001 and ALH-9011 handhelds.

7.16.1 EnableFlashLight()

```
SYSSVC_RESULT EnableFlashLight(bool Enable)
```

Turns on/off the camera flashlight.

Parameters:
Enable
[IN] true = on; false = off.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.17 Keyboard Event Service Function

These functions allow you to subscribe to system-level key events, without requiring the use of Windows Forms KeyEvent handlers.

7.17.1 KeyEventNotificationSet()

```
SYSSVC_RESULT KeyEventNotificationSet(KeyEventNotifyCALLBACK CallbackProc)
```

Registers an application’s callback function which will receive key event notifications from the OS.

Parameters:
CallbackProc
[IN] CallbackProc function that will receive key event notification service.

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.17.2 SYSSVC_RESULT KeyEventNotificationReset( )

```
SYSSVC_RESULT KeyEventNotificationReset()
```

Removes the key event notification service’s Callback which has been registered with the OS.

Parameters:
None

Returns:
SYSSVC_RESULT_SUCCESS if executed successfully.

7.18 System API Properties

These properties can be access directly, providing you with the handheld’s DeviceID and UUID (universally unique identifier).

7.18.1 DeviceID (get)

```
string DeviceID { get; }
```

7.18.2 UUID (get)

```
string UUID { get; }
```

8 3G Modem

NOTE: Data structures and methods described in this section only apply to the **ALH-9001** handhelds. Control of the 3G Modem module on the **ALH-9011** handhelds is accomplished by using the standard Windows Mobile .NET Compact Framework Class Libraries.

The Modem API provides control over many aspects or the ALH-9001 handheld's 3G Modem module. The 3G modem allows you to connect to the device remotely, or offload data from the device onto your network, using a 3G cellular modem connection. To use the 3G modem, you must have a valid SIM (Subscriber Identity Module) card installed in the device, with an activated data plan.

A separate class, RAS_DEMO.cs, is provided to show you how you might handle 3G-specific connection functions, such as dialing and authenticating with a Remote Access Service (RAS), since there are different options for different localities.

8.1 Modem Data Structures

8.1.1 MODEM_RESULT

Type	Item	Description
MODEM_RESULT	MODEM_RESULT_SUCCESS	operation successfully performed
	MODEM_RESULT_INVALID_ARGS	invalid parameter
	MODEM_RESULT_OUTOFMEMORY	failed to assign a memory resource
	MODEM_RESULT_UNSUPPORTED	command not currently supported
	MODEM_RESULT_FAILURE	failed to perform operation
	MODEM_RESULT_ALREADY_OPENED	modem port has already been opened
	MODEM_RESULT_NO_BATTERY	the main battery level is insufficient
	MODEM_RESULT_TIMEOUT	no response from module
	MODEM_RESULT_ALREADY_ALLOCATED	modem memory already assigned
	MODEM_RESULT_NOT_OPENED	executed a modem function withint Opening first
	MODEM_RESULT_NOT_POWER_ON	executed a modem function without turhing Power on first
	MODEM_RESULT_ALREADY_POWER_ON	modem power already turned on

8.1.2 MODEM_SYS_INDEX_USER_WND

(Under Construction)

8.1.3 MODEM_SYS_NOTI

Information from the 3G modem is transmitted to your application via a callback function and a MODEM_CALLBACK_DATA parameter. Within that MODEM_CALLBACK_DATA is a field describing the source of the message, defined by the MODEM_SYS_NOTI enumeration.

```
public enum MODEM_SYS_NOTI {
  NETWORK_NOT_REGISTERED,
  NETWORK_REGISTERED,
  NOCARRIER,
  PORT_CLOSE_FAIL,
  PORT_CLOSE_SUCCESS,
  PORT_OPEN_FAIL,
  PORT_OPEN_SUCCESS,
  PORT_DOWN_FAIL,
  PORT_DOWN_SUCCESS,
  PWR_UP_INITIALIZE,
  PWR_UP_SUCCESS
}
```

- **NETWORK_NOT_REGISTERED**: Cannot confirm access to the 3G network.
- **NETWORK_REGISTERED**: Confirmed access to the 3G network.
- **NOCARRIER**: Disconnected from the 3G network.
- **PORT_CLOSE_FAIL**: Failed to close the 3G modem port.
- **PORT_CLOSE_SUCCESS**: Successfully closed the 3G modem port.
- **PORT_OPEN_FAIL**: Failed to open the 3G modem port.
- **PORT_OPEN_SUCCESS**: Successfully opened the 3G modem port.
- **PWR_DOWN_FAIL**: Failed to turn off the 3G modem port.
- **PWR_DOWN_SUCCESS**: Successfully turned off the 3G modem port.
- **PWR_UP_FAIL**: Failed to turn on the 3G modem port.
- **PWR_UP_SUCCESS**: Successfully turned on the 3G modem port.

8.1.4 MODEM_POWER_STATUS

When asking for the power status of the 3G modem, the API responds with a MODEM_POWER_STATUS, enumerated as follows:

```
public enum MODEM_POWER_STATUS {
  MODEM_POWER_STATUS_OFF,
  MODEM_POWER_STATUS_ON,
  MODEM_POWER_STATUS_INITIALIZING,
  MODEM_POWER_STATUS_DEINITIALIZING
}
```

- **MODEM_POWER_STATUS_OFF**: Modem power is off.
- **MODEM_POWER_STATUS_ON**: Modem power is on.
- **MODEM_POWER_STATUS_INITIALIZING**: Modem power is in the process of turning on.
- **MODEM_POWER_STATUS_DEINITIALIZING**: Modem power is in the process of turning off.

8.1.5 MODEM_SIM_PIN_AUTH_STATUS

When asking for the status of the SIM card installed in the 3G modem with SimQueryCardHolderStatus (), the API responds with a MODEM_SIM_PIN_AUTH_STATUS, enumerated as follows:

```
public enum MODEM_SIM_PIN_AUTH_STATUS {
  CARD_HOLDER_TRAY_REMOVED,
  INCORRECT_PASSWORD,
  INVALID_INPUT_VALUE,
  SIM_BLOCKED,
  SIM_BUSY,
  SIM_FAILURE,
  SIM_INSERTED,
  SIM_PIN,
  SIM_PUK,
  SIM_READY,
  SIM_SUCCESS,
  SIM_WRONG
}
```

- **CARD_HOLDER_TRAY_REMOVED**: SIM card is not installed.
- **INCORRECT_PASSWORD**: Password is incorrect.
- **INVALID_INPUT_VALUE**: Inputted value is invalid.
- **SIM_BLOCKED**: Aborted because of a previous operation.
- **SIM_BUSY**: Previous operation is still going.
- **SIM_FAILURE**: Failed to execute SIM function.

- **SIM_INSERTED**: SIM card inserted into the slot.
- **SIM_PIN**: Waiting for you to send PIN #.
- **SIM_PUK**: Waiting for you to send a PUK #. PUK codes are required after entering the wrong PIN # three times in a row.
- **SIM_READY**: PIN is inputted and SIM card is ready.
- **SIM_SUCCESS**: Succeeded in performing the 3G function.
- **SIM_WRONG**: Invalid SIM card.

8.1.6 MODEM_NETWORK_REGISTRATION_STATUS

Using GetNetworkRegistrationStatus function to checking the network connection status.

```
public enum MODEM_NETWORK_REGISTRATION_STATUS {
  COMMAND_FAILED,
  NOT_REGISTERED,
  NOT_REGISTERED_BUT_SEARCHING_OPERATOR,
  REGISTERED_TO_GSM,
  REGISTERED_TO_UTRAN,
  REGISTRATION_DENIED,
  UNKNOWN
}
```

- **COMMAND_FAILED**: Failed to perform command.
- **NOT_REGISTERED**: Not connected, and unable to register. The reason might be:
  - no SIM card available
  - no PIN entered
  - no valid Home PLMN entry found on the SIM
- **NOT_REGISTERED_BUT_SEARCHING_OPERATOR**: Searching for an available network.
- **REGISTERED_TO_GSM**: Connected to GSM network (2G).
- **REGISTERED_TO_UTRAN**: Connected to UTRAN network (3G).
- **REGISTRATION_DENIED**: Failed registration or authentication, The reason might be:
  - IMSI unknown at HLR
  - illegal Mobile Station
  - illegal Mobile Equipment
- **UNKNOWN**: Unexpected error.

8.1.7 MODEM_CALLBACK_DATA

You use the SetCallback() function to register your own ModemCallbackProc obect as a listener for events sent by the 3G modem. When an event happens, your ModemCallbackProg is called and passed a MODEM_CALLBACK_DATA object. This struct has everything you need to know about the event, including its type and handles required to fetch nested data/parameters that are attached to it.

```
public struct MODEM_CALLBACK_DATA {
  nWndIndex;
  m_UsrWndMsg;
  bBatteryDetect;
  wParam;
  lParam;
}
```

- **nWndIndex**: (Not used).
- **m_UsrWndMsg:** The reason for the callback (a MODEM_SYS_NOTI).
- **bBatteryDetect:** Whether of not the main battery is installed.
- **wParam:** More information on the message (not used).
- **lParam**: More information on the message (not used).

8.1.8 ModemCallbackProc

This is a delegate function that is called when 3G modem data is received from the modem module. In order to process data in your application, use the Modem**`SetCallback(ModemCallbackProc ModemCallback)`** function to assign your delegate function. When the API notifies your application, it will use the callback with the following signature:

```
public delegate void ModemCallbackProc(MODEM_CALLBACK_DATA CallbackData);
```

- **CallbackData**: The structure containing the reason for the callback.

8.2 Modem API Methods

These methods allow you to power the 3G modem on and off, open and close a connection with the modem, and interact with the SIM card in various ways, such as: whether it is inserted or not, setting the PIN and PUK, and SIM locking & unlocking.

8.2.1 ModemApi()

```
ModemApi modem = new ModemApi()
```

This is the constructor method for the main Modem class. All subsequent API operations are performed by calling methods on the ModemApi object.

8.2.2 AllocContext()

```
MODEM_RESULT AllocContext()
```

You are required to allocate resources for the modem device. This should be the firstl thing you do before using the modem API.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if resources allocated successfully.

8.2.3 DeallocContext()

```
MODEM_RESULT DeallocContext()
```

Deallocate the modem device resources. Call this when you are all done using the modem.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if resources deallocated successfully.

8.2.4 PowerUp()

```
MODEM_RESULT PowerUp()
```

Turns the modem power on. After this function is called, the modem will call your registered callback delegate function with status updates. When the value of CallbackData.m_UserWndMsg passed to the delegate function reports PWR_UP_SUCCESS, then the modem power has successfully turned on.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.5 PowerDown()

```
MODEM_RESULT PowerDown()
```

Turns the modem power off. After this function is called, the modem will call your registered callback delegate function with status updates. When the value of CallbackData.m_UserWndMsg passed to the delegate function reports PWR_DOWN_SUCCESS, then the modem power has successfully turned off.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.6 SetCallback()

```
MODEM_RESULT SetCallback(ModemCallbackProc CallbackProc)
```

Registers your callback delegate with the modem API. Your application is informed of changes to the modem status or the results of the function execution through this callback mechanism.

Parameters:
pFunc
`[IN] Callback delegate which will be called by the modem API.`

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.7 GetPowerStatus()

```
MODEM_POWER_STATUS GetPowerStatus()
```

Returns the current status of the modem power, as a `MODEM_POWER_STATUS` enumeration.

Parameters:
None.

Returns:
The current power status of the modem (see `MODEM_POWER_STATUS`).

8.2.8 Open()

```
MODEM_RESULT Open()
```

Opens the communication port to the modem, and then checks the status of the Network. Under normal operation, this function can take up to 1 minute. When complete, the API will report `NETWORK_REGISTERED` to your callback delegate. If there is an error or it takes too long, it will report `NETWORK_NOT_REGISTERED` to your callback delegate.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.9 Close()

```
MODEM_RESULT Close()
```

Closes the communication port with the modem.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.10 IsOpened()

```
BOOL        IsOpened()
```

Reports whether the modem communication port is already open or not.

Parameters:
None.

Returns:
`TRUE` if port is open; `FALSE` if port is closed.

8.2.11 IsRegistered()

```
BOOL        IsRegistered()
```

Reports whether the modem is aleady registered with the network or not.

Parameters:
None.

Returns:
`TRUE` if modem is connected to the network; `FALSE` if it is not.

8.2.12 Reset()

```
MODEM_RESULT Reset()
```

This function is not currently supported, `MODEM_RESULT_UNSUPPORTED` will be returned this function is called.

Parameters:
None.

Returns:
`MODEM_RESULT_SUCCESS` if successfully executed.

8.2.13 SimChangePin()

```
MODEM_SIM_PIN_AUTH_STATUS SimChangePin(string OldPIN, string NewPIN)
```

Sets/changes the SIM card’s PIN. SIM cards that are locked require the user to manually enter the PIN before they can be used.

Parameters:
`OldPIN`
`[IN] Current PIN number.`
`NewPIN`
`[IN] New PIN number`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.14 SimQueryAuthStatus()

```
MODEM_SIM_PIN_AUTH_STATUS SimQueryAuthStatus()
```

Reports the current status of the Network authentication of the SIM/modem. Before you can communicate over the modem, you need to first check whether or not the SIM card’s authentication status is READY. If it reports a status of SIM_PIN, should you need to enter the PIN number.

Parameters:
None.

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.15 SimEnterPin()

```
MODEM_SIM_PIN_AUTH_STATUS SimEnterPin(string PIN)
```

Enters the SIM card PIN. If the SIM is locked and you may need to tell it the PIN number at the right time during communication (when it reports SIM_PIN as a status).

Parameters:
`PIN`
`[IN] SIM Card PIN number.`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.16 SimEnterPuk()

```
MODEM_SIM_PIN_AUTH_STATUS SimEnterPuk(string PUK, string NewPIN)
```

Enters the SIM card's PUK. If you continually enter an incorrect PIN number three times, then you will receive a SIM status of SIM_PUK, and will need to enter the PUK number with this function call. A PUK is available from your mobile carrier, when needed.

Parameters:
`PUK`
`[IN] SIM Card PUK number.`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.17 SimGetPinCounter()

```
MODEM_SIM_PIN_AUTH_STATUS SimGetPinCounter(ref intPinCounter)
```

Reads the PIN counter. The PIN counter will decrement by one every time an incorrect PIN number is entered. Not currently supported in the 3G Modem (HC25).

Parameters :
`Active`
`[out] the remaining frequency of enter PIN number.`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.18 SimGetLockStatus()

```
MODEM_SIM_PIN_AUTH_STATUS SimGetLockStatus(ref bool Active)
```

Returns the current lock status of SIM card. If the SIM is locked, every time you apply power to the modem, it wait in the status of SIM_PIN until you enter the PIN number.

Parameters :
`Active`
`[out] reference parameter which will contain the current lock setting.`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.19 SimLockUnlock()

```
MODEM_SIM_PIN_AUTH_STATUS SimLockUnlock(string PIN, bool bLock)
```

Sets the SIM card's Lock/Unlock status. If the SIM is locked, every time you apply power to the modem, it wait in the status of SIM_PIN until you enter the PIN number.

Parameters :
`PIN`
`[IN] PIN number of SIM Card.`
`bLock`
`[IN] New lock setting: TRUE=lock, FALSE=unlock`

Returns:
The current status of the SIM (see MODEM_SIM_PIN_AUTH_STATUS).

8.2.20 GetNetworkRegistrationStatus()

```
MODEM_NETWORK_REGISTRATION_STATUS GetNetworkRegistrationStatus()
```

Read the connection status to the network. During the initial connection to the network, the modem goes through various steps – registering, authenticating, etc. This call reports back to you the current status of the network registration.

Parameters:
None.

Returns:
Current status of Modem connection to the network.
(Refer to MODEM_NETWORK_REGISTRATION_STATUS)

8.2.21 RasDial()

```
Bool  RasDial(string EntryName, string UserName, string Password, IntPtr hWnd)
```

Try to connect to the Remote Access Server (RAS). The result of the connection attempt is returned by the function. The system will send you RAS dial events (same as the Microsoft RAS dial events), if you provide a hWnd handle.

If you set the register value of \HKEY_CURRENT_USER\Software\ATID\EnableCheckPPPAdapter to 1 (the default value is 0), then when the modem fails to connect to the RAS, it will re-try again up to ten times.

Parameters:
`EntryName`
`[IN] name of phone-book entry, which will be used at phone. It must to be created before calling the RasDial function.`
`UserName`
`[IN] User name that will be used for RAS dial-up authentication.`
`Password`
`[IN] password that will be used for RAS access authentication.`
`hWnd`
`[IN] user window handle that will received RAS dial event.`

Returns:
TRUE if RAS connection succeeded; FALSE if it did not.

8.2.22 RasHangUp()

```
void  RasHangUp()
```

Hang up the RAS Connection.

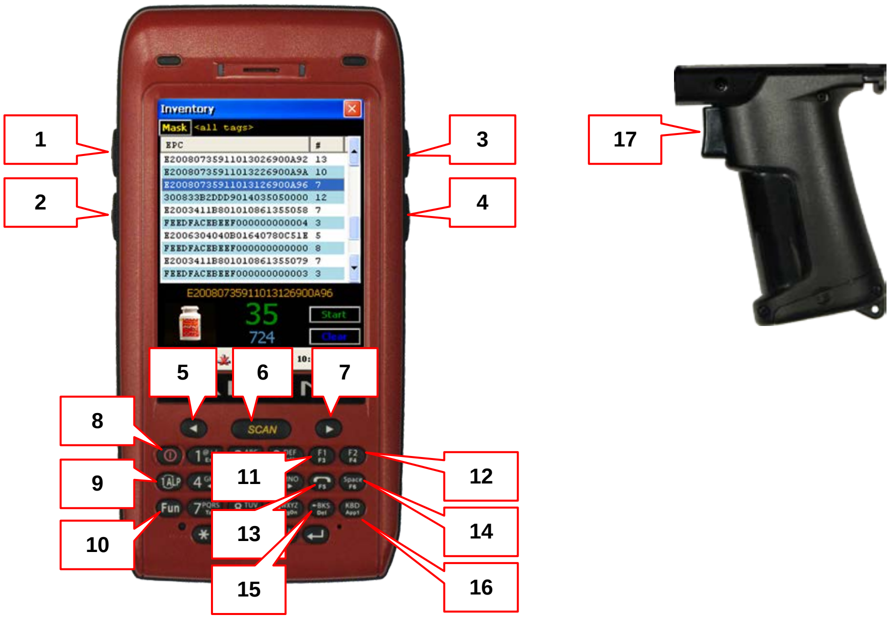
Parameters & Returns:

None.

9 Key Codes

The physical keys on the handheld each generate a unique Key Code, and when you handle key events in your application you look at the reported key code to determine which button was pressed. In the Windows CE versions of the handheld (ALH-9000 and ALH-9001), those key codes correspond to enumerated function keys, arrow keys, and other common keys such as Space, Back, and Delete. In the Windows Mobile versions of the handheld (ALH-9010 and ALH-9011), those convenient enumerated values are not available for use, so you have to instead look for specific hexadecimal values.

The image below labels each of the hardware keys, and the following tables detail the specific codes generated by each key.



9.1 ALH-9000/9001 Key Codes (Windows CE)

Key Number	Key State		Key State with 'Fun' key pressed	
	Up	Down	Up	Down
1	Up	Up		
2	Down	Down		
3	F8	F8	F8	F8
4	F7	F7	F7	F7
5	Left	Left	Left	Left
6	F9	F9	F9	F9
7	Right	Right	Right	Right
8	F17	F17	F17	F17
9	F20	F11	F20	F11
10	F12	F12		
11	F1	F1	F3	F3
12	F2	F2	F4	F4
13		F15	F5	F5
14	Space	Space	F6	F6
15	Back	Back	Delete	Delete
16		F10		
17	F19	F19	F19	F19

9.2 ALH-9010/9011 Key Codes (Windows Mobile)

Number	Key State		Key State with 'Fun' key pressed	
	Up	Down	Up	Down
1	0x26	0x26		
2	0x28	0x28		
3	0xF0	0xF0	0xF0	0xF0
4	0xEF	0xEF	0xEF	0xEF
5	0x25	0x25	0x25	0x25
6	0xD3	0xD3	0xD3	0xD3
7	0x27	0x27	0x27	0x27
8	0x7C	0x7C	0x7C	0x7C
9	0xD0	0xD0	0xD0	0xD0
10	0xD1	0xD1		
11	0xE9	0xE9	0xEB	0xEB
12	0xEA	0xEA	0xEC	0xEC
13			0xED	0xED
14	0x20	0x20	0xEE	0xEE
15	0x08	0x08	0x2E	0x2E
16				
17	0xD2	0xD2	0xD2	0xD2