CS108 C# API and Programmer's Guide for iOS, Android and Windows 10

Version 2.0.4

1 Content

2 Release Notes

Dates	Release	Description
2017 05 02	1.0.0	Initial Release
2017 12 05	2.0.1	Add regulatory region setting, product serial number, and other APIs.
2017 12 12	2.0.2	Add compact inventory (fast inventory) API
2018 03 15	2.0.3	Change title to add Windows 10
2019 04 01	2.0.4	New Operations ---------- Operation.TAG_KILL Operation.TAG_READ Operation.TAG_WRITE Operation.TAG_AUTHENTICATE Operation.TAG_UNTRACEABLE NeSetw APIs ---------- SetSelectCriteria CancelAllSelectCriteria

3 Introduction

This Programmer's Guide provides a comprehensive guideline for the CS108 C# Callback-based API programming and software development. This API is applicable to both iPhone iOS and Android environment. This document provides detailed information about the programming interface made available by the CS108 Integrated Reader API interface (interface) for configuring, controlling, and accessing the RFID reader.

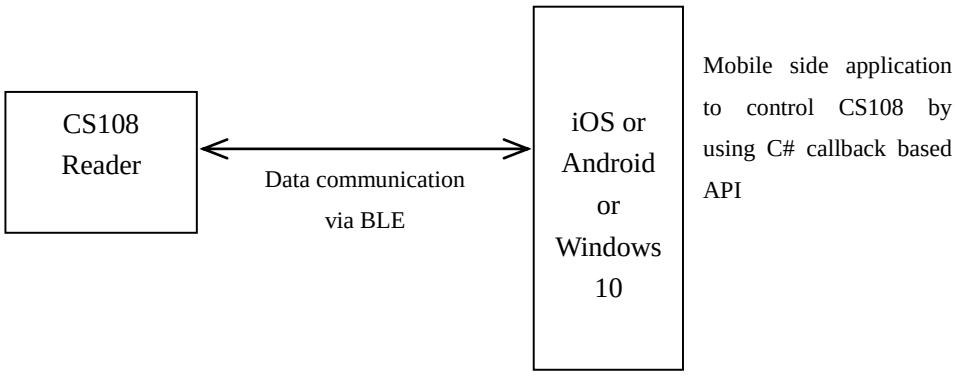
This document begins with a system level overview of the interface and a discussion of topics related to interface development. Each API call and its arguments are then described in detail. The API flow and sample codes of some basic operations of the reader (e.g. setting up, doing inventory, read/write/lock tags etc.) are provided. The target audiences of this document are assumed to have basic knowledge in UHF RFID, EPC Class 1 Gen 2 Protocol and Microsoft Dot Net C# programming language.

Chapter 4:	System requirements for developers
Chapter 5, 6, and 7:	API and Library
Chapter 8 and 9:	Programmer's Guide
Appendices:	Key RFID operation parameters explanation

4 Pre-requisite

The build environment consists of tools and the corresponding configurations of the Visual Studio 2017(Windows) or Xamarin (Mac). It is expected that the system integrator or the software system programming house will be developing the applications on Visual Studio 2017(Windows) or Xamarin (Mac). With this tool, typically he has to write programs on the PC/Mac. The following are needed to set up the build environment.

4.1 System Diagram



4.2 Basic Developer Configuration on PC side:

Operating System requirement:

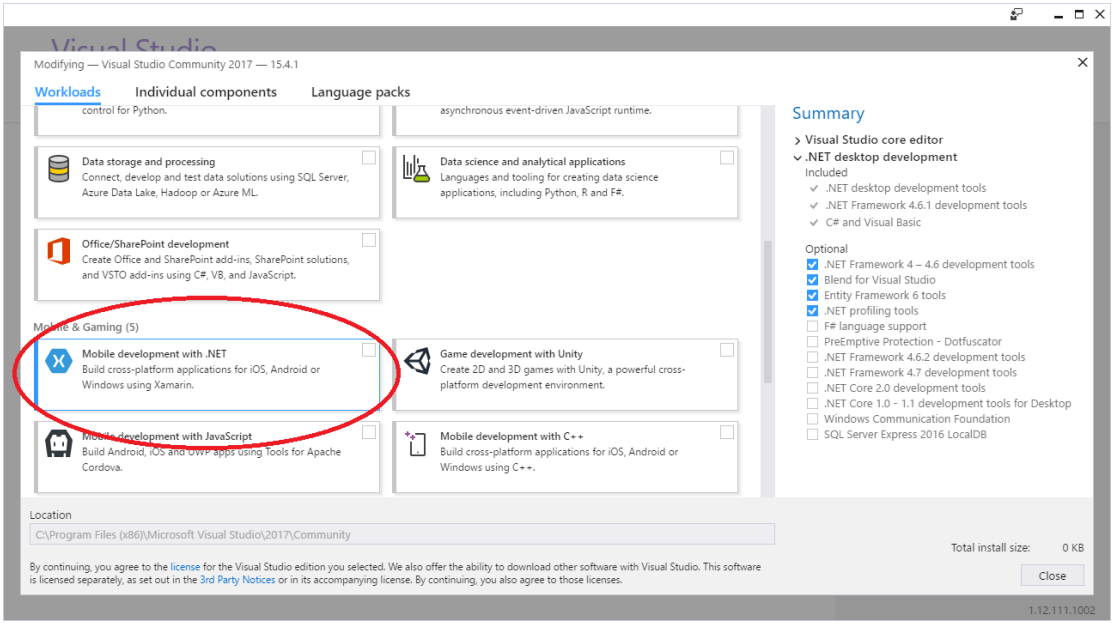
- Microsoft Windows 10 (English)

Software package required:

- Microsoft Visual Studio 2017

To build demo application successfully, you need to install Microsoft Visual Studio 2017 or above. For more detailed information, please go to Microsoft webpage (https:/docs.microsoft.com/en-us/visualstudio/welcome-to-visual-studio).
Visual Studio 2017 -
https://www.visualstudio.com/zh-hant/vs/whatsnew/

In Visual Studio, you need to use the Visual Studio Installer to add the Xamarin package:



4.3 Basic Developer Configuration on Mac side:

Operating System requirement:

- Mac OS 10 (English)

Software package required:

- Microsoft Visual Studio 2017 for MAC
- Xcode 9

To build demo application successfully, you need to install Visual Studio 2017 or above. For more detailed information, please go to Xamarin webpage (https://www.xamarin.com/).
Xamarin -
https://www.xamarin.com/platform

5 Callback-based API Library

5.1 API Library Files

The CS108 Callback-based API Library consists of the following files.

File	Location of source code	Remarks
CSLibrary.dll	Inside folder "*CS108\CSLibrary\bin\Debug \"* of the CS108 Demo Code	CSL C# Callback-based API Class Library. Different library files.

5.2 Version History

- The information provided in this document is for Callback-based API (CSL Unified API).

6 CS108 C# API: Theory of Operation

The CS108 C# Application Programming Interface (API) provides a programming interface for controlling CS108 integrated reader. The interface is loaded by a mobile application; the application in turn explicitly initializes the interface. The interface supports enumeration of attached RFID radio modules, returning unique identification information for each currently-attached RFID radio modules. An mobile application uses the CS108 C# API to establish a connection and grant the application exclusive control of the corresponding RFID radio module. After an application is granted exclusive control of an RFID radio module, the application can configure the RFID radio module for operation and tag protocol operations can be issued. The CS108 C# API allows an application control of low level functions of the Firmware, including but not limited to:

- regulatory configuration of frequencies
- antenna output power
- air protocol parameters, such as Q value

Some of these configuration parameters are abstracted by the CS108 C# API. The application initiates transactions with ISO 18000-6C tags or tag populations by executing ISO 18000-6C tag-protocol operations. The interface exposes direct access to the following ISO 18000-6C tag-protocol operations:

- Inventory
- Read
- Write
- Kill
- Erase
- Lock

When executing tag-protocol operations, the interface provides the application with the ability to configure tag-selection (i.e., ISO 18000-6C Select) criteria and query (i.e. ISO 18000-6C Query) parameters. The interface extends the ISO 18000-6C tag- protocol operations by additionally providing the application the ability to specify a post-singulation Electronic Product Code (EPC) match mask as well as the number of tags to which the operation is to be applied. Additionally, the interface supports configuration of dense-reader mode during ISO 18000-6C tag-protocol operations. The interface supports the configuration and control of the antenna. The application is given fine-grained control to configure:

- A time limit for performing tag-protocol operations (dwell time)
- The number of times a tag-protocol operation is executed (number of inventory rounds)
- RF characteristics (for example, RF power).

The interface supports a callback model for presenting tag-protocol operation response data to the application. When an application issues a tag-protocol request (i.e. inventory, read, etc.), it also provides a pointer to a callback function API invokes. To help simplify the packet-processing code the API provides complete tag-protocol operation response packets. Tag-protocol operation results

include EPC values returned by the Inventory operation, read data returned by the Read operation, and operation status returned by the Write, Kill, Erase, and Lock operations. The application can request the returned data be presented in one of three formats: compact, normal, or extended. Compact mode contains the minimum amount of data necessary to return the results of tag-protocol operations to the application. Normal mode augments compact mode by interleaving additional status/contextual information in the operation results, so that the application can detect, for example, the start of inventory rounds, when a new antenna is being used, etc. Extended mode augments normal mode by interleaving additional diagnostic and statistical data with the operation results. The interface supports diagnostic and statistical reporting for radio, inventory, singulation, tag access, and tag performance as well as status packets to alert the application to unexpected errors during tag- protocol operations.

The interface provides the application with access to (i.e., read and write) the configuration data area on the RFID radio module. The application can use the configuration data area to store and retrieve the specific hardware configuration and capabilities of the radio. The application can read a RFID radio module's configuration data area immediately after gaining exclusive control of the RFID radio module and use that data to configure and control low-level radio parameters. The interface also supports low-level control of the RFID Firmware.

7 Callback-based API Classes, Methods and Events

7.1 Overview

There are 4 events for RFID related process. To receive data from event, you must attach those events first.

1. StateChangedEvent – report all operation state.
   e.g., Inventory started, the state will change from STATE.IDLE to STATE.BUSY.
   Note: You can use this event to determine whether the reader is busy or not, only one operation can execute at anytime.

2. TagInventoryEvent – report Inventory data to user, contain detailed info.
   This operation is configurable through SetTagGroup ,SetFixedQParms , SetDynamicQParms, SetDynamicAdjustParms, SetDynamicThresholdParms and SetOperationMode.

3. TagRangingEvent – report Ranging data to user

4. TagAccessCompletedEvent – reoprt the tag access result to user.
   This operation is only use if Operation is
   Operation.TAG_READ_ACC_PWD,
   Operation.TAG_READ_KILL_PWD,
   Operation.TAG_READ,
   Operation.TAG_READ_PC,
   Operation.TAG_READ_EPC,
   Operation.TAG_READ_TID,
   Operation.TAG_READ_USER,
   Operation.TAG_WRITE_ACC_PWD,
   Operation.TAG_WRITE_KILL_PWD,
   Operation.TAG_WRITE,
   Operation.TAG_WRITE_PC,
   Operation.TAG_WRITE_EPC,
   Operation.TAG_WRITE_USER,
   Operation.TAG_LOCK_ACC_PWD,
   Operation.TAG_LOCK_KILL_PWD,
   Operation.TAG_LOCK_EPC,
   Operation.TAG_LOCK_TID,


```
Operation.TAG_LOCK_USER,
Operation.TAG_KILL,
Operation.TAG_AUTHENTICATE,
Operation.TAG_UNTRACEABLE
```

Events for barcode. To receive data from event, you must attach those events first.

5. OnCapturedNotify – report barcode data to user

6. OnStateChanged – report barcode reader status to user

Events for general notification. To receive data from event, you must attach those events first.

7. OnKeyEvent – report Key button status to user

8. OnVoltageEvent – report battery level to user

Event for SiliconLab IC process. To receive data from event, you must attach those events first.

9. OnAccessCompletedEvent – report all data return.
   e.g., device serial number

Classes, Methods and Events in Callback-based API

For detail description of all Classes, Methods, Events, Fields and Parameters, please refer to the CS108 Callback-based API CSLibrary.

7.1.1 CSLibrary Namespace

7.1.1.1 Classes Overview

Classes

	Class	Description
	HighLevelInterface	Reader HighLevelInterface
	rfid	For RFID
	barcode	For Barcode
	notification	For Common notification (Key button)

7.1.1.2 HighLevelInterface Class

Methods for Class HighLevelInterface

	Name	Description
	ConnectAsync	Connect to CS108 reader.
	DisconnectAsync	Diusconnect the reader

7.1.1.3 rfid Class

Methods for Class rfid

	Name	Description
	Dispose	Destructor
	GetActiveLinkProfile	Overloaded.
	GetActiveMaxPowerLevel	Overloaded.
	GetActiveRegionCode	Available region you can use
	GetAvailableFrequencyTable	Overloaded.
	GetCountryCode	GetCountryCode
	GetCSLibraryVersion	Get RFID CSharp Library Version
	GetCurrentLinkProfile	Allows the application to retrieve the current link profile for the radio module. The current link profile cannot be retrieved while a radio module is executing a tag-protocol operation.
	GetCurrentSingulationAlgorithm	Get Current Singulation Algorithm
	GetDynamicQParms	Get DynamicQ Singulation Algorithm
	GetFixedQParms	Get FixedQ Singulation Algorithm
	GetInventoryCycle	Get the maximum number of inventory cycles to attempt on the antenna port during a tag-protocol- operation cycle before switching to the next enabled antenna port. An inventory cycle consists of one or more executions of the singulation algorithm

		for a particular inventory-session target (i.e., A or B). If the singulation algorithm [SING-ALG] is configured to toggle the inventory-session, executing the singulation algorithm for inventory session A and inventory session B counts as two inventory cycles. A value of zero indicates that there is no maximum number of inventory cycles for this antenna port. If this parameter is zero, then dwellTime may not be zero. See for the effect of antenna-port dwell time and number of inventory cycles on the amount of time spent on an antenna port during a single tag-protocol-operation cycle. NOTE: when performing any non-inventory ISO 18000- 6C tag access operation (i.e., read, write, kill, or lock), the radio module ignores the number of inventory cycles for the antenna port which is used for the tag-protocol operation.
	GetInventoryDuration	This is used to get inventory duration
	GetMaxForwardPowerLevel	Get the maximum PA output power level measured in 0.1 dBm units.
	GetOperationMode	Retrieves the operation mode for the RFID radio module. The operation mode cannot be retrieved while a radio module is executing a tag-protocol operation.
	GetPowerLevel	GetPowerLevel
	GetSelectCriteria	Get Current Criteria Settings
	GetSingulationAlgorithmParms	GetSingulationAlgorithmParms
	GetTagGroup	Get Tag Group
	SetCurrentLinkProfile	Overloaded.
	SetCurrentSingulationAlgorithm	Allows the application to set the currently-active singulation algorithm (i.e., the one that is used when performing a tag-protocol operation (e.g., inventory, tag read, etc.)). The currently-active singulation algorithm may not be changed while a radio module is executing a tag-protocol operation.
	SetDynamicQParms	Overloaded.
	SetFixedChannel	Overloaded.
	SetFixedQParms	Overloaded.
	SetInventoryCycle	Specifies the maximum number of inventory cycles to attempt on the antenna port during a tag-protocol- operation cycle before switching to the next enabled antenna port. An inventory cycle consists of one or more executions of the singulation algorithm for a particular inventory-session target (i.e., A or B). If the singulation algorithm [SING-ALG] is configured to toggle the inventory-session, executing the singulation algorithm for inventory session A and inventory session B counts as two inventory cycles. A value of zero indicates that there is no maximum number of inventory cycles for this antenna port. If this parameter is zero, then dwellTime may not

		be zero. See for the effect of antenna-port dwell time and number of inventory cycles on the amount of time spent on an antenna port during a single tag-protocol-operation cycle. NOTE: when performing any non-inventory ISO 18000- 6C tag access operation (i.e., read, write, kill, or lock), the radio module ignores the number of inventory cycles for the antenna port which is used for the tag-protocol operation.
	SetInventoryDuration	This is used to set inventory duration
	SetOperationMode	Sets the operation mode of RFID radio module. By default, when an application opens a radio, the RFID Reader Library sets the reporting mode to non-continuous. An RFID radio module's operation mode will remain in effect until it is explicitly changed via RFID_RadioSetOperationMode, or the radio is closed and re- opened (at which point it will be set to non-continuous mode). The operation mode may not be changed while a radio module is executing a tag-protocol operation.
	SetPostMatchCriteria	Configures the post-singulation match criteria to be used by the RFID radio module. The supplied post-singulation match criteria will be used for any tag-protocol operations (i.e., Inventory, etc.) in which the application specifies that a post-singulation match should be performed on the tags that are singulated by the tag-protocol operation (i.e., the SelectFlags.POST_MATCH flag is provided to the appropriate RFID_18K6CTag* function). The post-singulation match criteria will stay in effect until the next call to SetPostMatchCriteria. Post-singulation match criteria may not be changed while a radio module is executing a tag-protocol operation.
	SetPowerLevel	Set Power Level(Max 300)...
	SetSelectCriteria	Configures the tag-selection criteria for the ISO 18000-6C select command. The supplied tag-selection criteria will be used for any tag-protocol operations (i.e., Inventory, etc.) in which the application specifies that an ISO 18000-6C select command should be issued prior to executing the tag-protocol operation (i.e., the SelectFlags.SELECT flag is provided to the appropriate RFID_18K6CTag* function). The tag-selection criteria will stay in effect until the next call to SetSelectCriteria. Tag-selection criteria may not be changed while a radio module is executing a tag-protocol operation.
	SetSingulationAlgorithmParms	SetSingulationAlgorithmParms
	SetTagGroup	Overloaded.
	StartOperation	Start operation
	StopOperation	Stop current operation by abort or cancel
	SetSelectCriteria	Configures the tag selection criteria

	CancelAllSelectCriteria	Cancel all tag selection criteria

Events for Class rfid

	Name	Description
	OnAccessCompleted	Tag Access (including Tag read/write/kill/lock) completed event
	OnAsyncCallback	Tag Inventory(including Inventory and search) callback event
	OnStateChanged	Reader Operation State Event

Events for Class SiliconLab IC

	Name	Description
	OnAccessCompleted	Data Access (including serial number) completed event

7.1.2 CSLibrary.Constants Namespace

7.1.2.1 Enumerations Overview

Enumerations

	Name	Description
	Action	Specifies the action that will be applied to the tag populations (i.e, the matching and non-matching tags).
	BandState	Frequency Band State
	Bank	Memory bank
	CallbackType	Callback Type
	EpcMDID	EPCglobal Tag Mask Designer Identifier
	ErrorCode	Operation Error Code
	ErrorType	Error Type
	ExtCmd	Extended Kill command for UHF class 1 gen-2 version 1.2
	MemoryBank	Tag's memory bank
	MillerNumber	The miller number (i.e., cycles per symbol) that is sent as part of the Query command.
	ModulationType	The modulation type that is used by the profile.
	Operation	RFID Operation Mode
	Permission	A tag-permission command (aka, tag lock) allows the application to set the access permissions of a tag. These include the following: • Set whether or not an access password is required to write to the EPC, TID, or user memory banks. • Set whether or not the above memory-write permission is permanently set. Once the memory-write permission has been permanently set, attempts to change the permission or turn off the permanent setting will fail. • Set a memory bank to be read-only. • Set whether or not the individual passwords (i.e., access and kill) may be accessed (i.e., read and written) and, if they are accessible, whether or not an access password is required to read the individual passwords (i.e., access and kill). • Set whether or not the above password-access permission is permanently set. Once the password-access permission has been permanently set, attempts to change the permission or turn off the permanent

		setting will fail. • Set the individual passwords to be inaccessible (i.e., unable to be read or written).
	RegionCode	Region Profile
	ResponseMode	The requested data-reporting mode for the data type specified by responseType.
	ResponseType	The type of data that will have its mode set to the mode specified by responseMode. For version 1.1 of the RFID Reader Library, the only valid value is RFID_RESPONSE_TYPE_DATA.
	Result	function result value definitions
	RFID_ACCESS	Tag access enum result
	RFState	Reader Operation State
	Selected	Specifies the state of the selected (SL) flag for tags that will have the operation applied to them.
	SelectFlags	RFID Flags
	SelectMaskFlags	Tag Parameters Selected flags
	Session	Specifies which inventory session flag (i.e., S0, S1, S2, or S3) will be matched against the inventory state specified by target.
	SessionTarget	Specifies the state of the inventory session flag (i.e., A or B), specified by session, for tags that will have the operation applied to them.
	SingulationAlgorithm	Based upon usage scenarios, different singulation algorithms (i.e., Q-adjustment, etc.) may be desired. This document simply documents the mechanisms by which an application can choose and configure singulation algorithms. This document does not provide specific information about the singulation algorithms.
	TagAccess	The values that can be found in the command field for tag access packets
	Target	Specifies what flag, selected (i.e., SL) or one of the four inventory flags (i.e., S0, S1, S2, or S3), will be modified by the action.
	WriteType	The type of write that will be performed – i.e., sequential or random. The value of this field determines which of the structures within parameters contains the write parameters.

7.1.3 CSLibrary.Events Namespace

7.1.3.1 Classes Overview

Classes

	Class	Description
	OnAccessCompletedEventArgs	Tag Access Completed Argument
	OnAsyncCallbackEventArgs	Inventory or tag search callback event argument
	OnStateChangedEventArgs	Reader Operation changed EventArgs

7.1.3.2 OnAccessCompletedEventArgs Class

Fields

	Name	Description
	Access	Access Type
	Bank	Access bank
	Data	Access Data only use for Tag Read operation
	Success	Access result

7.1.3.3 OnAsyncCallbackEventArgs Class

Fields

	Name	Description
	Cancel	Cancel async callback.
	Info	Callback Tag Information
	Type	Async callback data type

7.1.3.4 OnStateChangedEventArgs Class

Fields

	Name	Description
	State	Current operation state

7.1.3.5 ResultArgs Class

Properties

	Name	Description
	Result	Result

7.1.4 CSLibrary.Structures Namespace

7.1.4.1 Classes Overview

Classes

	Class	Description
	CSLibraryOperationParms	Operation Parameter
	DynamicQParms	The parameters for the dynamic-Q algorithm, MAC singulation algorithm 1, (i.e., RFID_18K6C_SINGULATION_ALGORITHM_DYNAMICQ)
	EASParms	EAS

	FixedQParms	The parameters for the fixed-Q algorithm, MAC singulation algorithm 0, (i.e., RFID_18K6C_SINGULATION_ALGORITHM_FIXEDQ)
	LibraryVersion	Library Version Structure
	ProfileConfig	Link Profile Configuration Structure
	QueryParms	Once the tag population has been partitioned into disjoint groups, a subsequent tag-protocol operation (i.e., an inventory operation or access command) is then applied to one of the tag groups. A tag group is specified using the following:
	S_DATA	Custom Data Structure
	S_EPC	Electronic Product Code
	S_PC	Protocol Control(must be 2 Bytes)
	S_PWD	Custom Password Structure
	S_TID	Protocol Control(must be 2 Bytes)
	SelectAction	The partitioning of tags into disjoint groups is accomplished by applying actions to the tags that match and/or do not match the specified mask.
	SelectCriteria	Tag-selection criteria
	SelectCriterion	Together, the selection mask and selection action form a single selection criterion.
	SelectMask	The tag mask is used to specify a bit pattern that is used to match against one of a tag's memory banks to determine if it is matching or non-matching. A mask is a combination of a memory bank and a sequence of bits that will be matched at the specified offset within the chosen memory bank.
	SingulationAlgorithmParms	SingulationAlgorithmParms
	SingulationCriteria	Post-singulation match criteria can be grouped together using the following:
	SingulationCriterion	Together, the selection mask and an indication if the application is interested in matching or non-matching tags form a single post-singulation match criterion.
	SingulationMask	The post-singulation match mask is used to specify a bit pattern of up to 496 bits that is used to match against the EPC backscattered by a tag during singulation to determine if a tag is matching or non-matching.
	TagBlockPermalockParms	block lock structure, configure this before do block lock
	TagCallbackInfo	Tag Callback Information
	TagGroup	A class that specifies the tag group that will have a subsequent tag-protocol operation applied to it. This parameter must not be NULL.
	TagInventoryParms	The ISO 18000-6C tag-inventory operation parameters
	TagKillParms	Tag Kill structure, configure this before do tag kill
	TagLockParms	Tag lock structure, configure this before do tag lock
	TagPerm	A structure that contains the access permissions to be set for the tag.
	TagPostMachParms	Post match tag parameters, configure this before any specific tag operation
	TagRangingParms	this parms is same as inventory parms
	TagReadParms	Read any bank structures, configure this before read data at any bank

	TagReadEpcParms	Read EPC structures, configure this before read current EPC
	TagReadPcParms	Read PC structures, configure this before read current PC
	TagReadProtectParms	TagReadProtectParms
	TagReadPwdParms	Read password structures, configure this before read current password
	TagReadTidParms	Read TID structures, configure this before read current TID
	TagReadUserParms	Read User memory structures, configure this before read current User memory
	TagSearchingParms	search one tag parms
	TagSelectedParms	Selected tag parameters, configure this before any specific tag operation
	TagWriteParms	Write any bank structures, configure this before write new value
	TagWriteEpcParms	Write EPC structures, configure this before write new EPC value
	TagWritePcParms	Write PC structures, configure this before write new PC value
	TagWritePwdParms	Write password structures, configure this before write new password value
	TagWriteUserParms	Write User structures, configure this before write new user data
	Version	Version Structure
	WriteParmsBase	Used internally only to represent the write parameter(s) anonymous union in the native RFID_18K6C_WRITE_PARMS struct
	WriteRandomParms	Write one or more 16-bit words to potentially non-contiguous 16-bit offsets to one of the tag's memory banks. This can be thought of as a random-access write to a single tag memory bank.
	WriteSequentialParms	Write, beginning at the specified 16-bit offset, a contiguous set of one or more 16-bit words to one of the tag's memory banks.
	TagAuthenticateParms	UCODE DNA Authenticate parameters
	TagUntraceableParms	UCODE DNA Untraceable parameters

Structures

	Name	Description
	DEVICE_STATUS	RFID Device Status Structures
	FrequencyBandParms	Frequency Band Parms
	INVENTORY_PKT	Tag Access Packet
	S_MASK	Custom Data Structure
	TAG_ACCESS_PKT	Tag Access Packet

Interfaces

	Name	Description
	IBANK	

Delegates

	Name	Description
	GPI_INTERRUPT_CALLBACK	GPI Interrupt callback
	InventoryCallbackDelegate	Tag access callback
	TagAccessCallbackDelegate	Tag access callback

7.1.4.2 CSLibraryOperationParms Class

Fields

	Name	Description
	EASConfig	Custom command EAS
	TagBlockLock	User Bank Perm-Lock
	TagInventory	Config this before search
	TagKill	Config this before kill
	TagLock	Config this before lock
	TagPostMatch	Selected a tag for read, write, lock, kill, and search one operation
	TagRanging	Config this before ranging all tags
	TagReadAccPwd	Config this before read Access password
	TagReadEPC	Config this before read EPC
	TagReadKillPwd	Config this before read Kill password
	TagReadPC	Config this before read PC
	TagReadProtect	Custom command TagReadProtect
	TagReadTid	Config this before read TID
	TagReadUser	Config this before read USER
	TagResetReadProtect	Custom command TagResetReadProtect
	TagSearchOne	Config this before search one tag
	TagSelected	Selected a tag for read, write, lock, kill, and search one operation
	TagWriteAccPwd	Config this before write access password
	TagWriteEPC	Config this before write EPC
	TagWriteKillPwd	Config this before write kill password
	TagWritePC	Config this before write PC
	TagWriteUser	Config this before write USER

7.1.4.3 DynamicQParms Class

Fields

	Name	Description
	length_	Structure size (Inherited from SingulationAlgorithmParms.)
	maxQueryRepCount	The maximum number of ISO 18000-6C QueryRep commands that will follow the ISO 18000-6C Query command during a single inventory round. Valid values are 0-255, inclusive.
	maxQValue	The maximum Q value to use. Valid values are 0-15, inclusive. maxQValue must be greater than or equal to startQValue and minQValue.
	minQValue	The minimum Q value to use. Valid values are 0-15, inclusive. minQValue

		must be less than or equal to startQValue and maxQValue.
	retryCount	Specifies the number of times to try another execution of the singulation algorithm for the specified session/target before either toggling the target (if toggleTarget is non- zero) or terminating the inventory/tag access operation. Valid values are 0-255, inclusive.
	startQValue	The starting Q value to use. Valid values are 0-15, inclusive. startQValue must be greater than or equal to minQValue and less than or equal to maxQValue.
	toggleTarget	A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e., A to B or B to A) and another inventory cycle run. A non-zero value indicates that the target should be toggled. A zero value indicates that the target should not be toggled. Note that if the target is toggled, retryCount and maxQueryRepCount will also apply to the new target.

Properties

	Name	Description
	length	Structure size (Inherited from SingulationAlgorithmParms.)

7.1.4.4 EASParms Class

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	enable	Enable EAS
	flags	Flag - Zero or combination of Select or Post-Match
	retryCount	Max retry count can't excess 15

7.1.4.5 FixedQParms Class

Fields

	Name	Description
	length_	Structure size (Inherited from SingulationAlgorithmParms.)
	qValue	The Q value to use. Valid values are 0-15, inclusive.
	repeatUntilNoTags	A flag that indicates whether or not the singulation algorithm should continue performing inventory rounds until no tags are singulated. A non-zero value indicates that, for each execution of the singulation algorithm, inventory rounds should be performed until no tags are

		singulated. A zero value indicates that a single inventory round should be performed for each execution of the singulation algorithm.
	retryCount	Specifies the number of times to try another execution of the singulation algorithm for the specified session/target before either toggling the target (if toggleTarget is non-zero) or terminating the inventory/tag access operation. Valid values are 0- 255, inclusive.
	toggleTarget	A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e., A to B or B to A) and another inventory cycle run. A non- zero value indicates that the target should be toggled. A zero value indicates that the target should not be toggled. Note that if the target is toggled, retryCount and repeatUntilNoTags will also apply to the new target.

Properties

	Name	Description
	length	Structure size (Inherited from SingulationAlgorithmParms.)

7.1.4.6 FrequencyBandParms Class

Fields

	Name	Description
	AffinityBand	AffinityBand
	Band	Frequency Band
	DivideRatio	DivideRation
	Frequency	Frequency
	GuardBand	GuardBand
	MaximumDACBand	MaximumDACBand
	MinimumDACBand	MinimumDACBand
	MultiplyRatio	MultiplyRatio
	State	State

7.1.4.7 LibraryVersion Class

Fields

	Name	Description
	major	Major (Inherited from Version.)
	minor	Minor (Inherited from Version.)
	patch	Patch (Inherited from Version.)

7.1.4.8 QueryParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Convert Version to string (Inherited from Version.)

Fields

	Name	Description
	singulationParms	Based upon usage scenarios, different singulation algorithms (i.e., Q-adjustment, etc.) may be desired. This document simply documents the mechanisms by which an application can choose and configure singulation algorithms.
	tagGroup	Once the tag population has been partitioned into disjoint groups, a subsequent tag-protocol operation (i.e., an inventory operation or access command) is then applied to one of the tag groups. A tag group is specified using the following:

7.1.4.9 S_DATA Class

Methods

	Name	Description
	CompareTo	Compare Data if equal return 0
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)

	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetLength	Get Data Length, Data in word format
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToBytes	Convert to byte array
	ToShorts	Convert to short array
	ToString	Convert to HexString (Overrides Object.::.ToString().)
	ToUshorts	Convert to ushort array

7.1.4.10 S_EPC Class

Methods

	Name	Description
	CompareTo	Compare Data if equal return 0
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetLength	Get Data Length, Data in word format
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToBytes	Convert to byte array
	ToShorts	Convert to short array
	ToString	Convert to HexString (Overrides Object.::.ToString().)
	ToUshorts	Convert to ushort array

7.1.4.11 S_MASK Class

Methods

	Name	Description
	Equals	Indicates whether this instance and a specified object are equal. (Inherited from ValueType.)

	GetHashCode	Returns the hash code for this instance. (Inherited from ValueType.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	ToString	return PC in string format (Overrides ValueType..::.ToString().)

Properties

	Name	Description
	Length	total byte length of DATA
	ToBytes	Data Value in Byte

7.1.4.12 S_PC Class

Methods

	Name	Description
	CompareTo	Compare Data if equal return 0
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetLength	Get Data Length, Data in word format
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToBytes	Convert to byte array
	ToShorts	Convert to short array
	ToString	Convert to HexString (Overrides Object..::.ToString().)
	ToUshorts	Convert to ushort array

Properties

	Name	Description
	EPCLength	Get 16bit EPC Length from current PC value
	UMI	User Memory Indicator, true if user memory contains data. Notes: Not all tags support this function

	XI	An XPC_W1 Indicator, true if XPC_W1 is non-zero value Notes: Not all tags support this function

7.1.4.13 S_PWD Class

Methods

	Name	Description
	CompareTo	Compare Data if equal return 0
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetLength	Get Data Length, Data in word format
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToBytes	Convert to byte array
	ToShorts	Convert to short array
	ToString	Convert to HexString (Overrides Object.::.ToString().)
	ToUshorts	Convert to ushort array

7.1.4.14 S_TID Class

Methods

	Name	Description
	CompareTo	Compare Data if equal return 0
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetLength	Get Data Length, Data in word format
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToBytes	Convert to byte array

	ToShorts	Convert to short array
	ToString	Convert to HexString (Overrides Object.::.ToString().)
	ToUshorts	Convert to ushort array

Properties

	Name	Description
	GetACID	Get Allocation Class ID
	GetEpcID	Get EPC Mask Designer ID
	GetIsoID	Get ISO Mask Designer ID

7.1.4.15 SelectAction Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	action	Specifies the action that will be applied to the tag populations (i.e, the matching and non-matching tags).
	enableTruncate	Specifies if, during singulation, a tag will respond to a subsequent inventory operation with its entire Electronic Product Code (EPC) or will only respond with the portion of the EPC that immediately follows the bit pattern (as long as the bit pattern falls within the EPC – if the bit pattern does not fall within the tag's EPC, the tag ignores the tag partitioning operation2). If this parameter is non-zero: • bank must be EPC. • target must be SELECTED. This action must correspond to the last tag select operation issued before the inventory operation or access command.
	target	Specifies what flag, selected (i.e., SL) or one of the four inventory flags (i.e., S0, S1, S2, or S3), will be modified by the action.

7.1.4.16 SelectCriteria Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	countCriteria	The number of selection criteria in the array pointed to by the pCriteria field. This field must be greater than or equal to zero and less than or equal to the maximum number of selection criteria as specified in [MAC- EDS]. Calling RFID_18K6CSetSelectCriteria with this field set to zero results in disabling all selection criteria (i.e., even if the SELECT flag is provided to the appropriate RFID_18K6CTag* function, no selects will be issued). If this field is zero, pCriteria may be NULL.
	pCriteria	A pointer to an array, containing countCriteria entries, of selection criterion structures that are to be applied sequentially, beginning with pCriteria[0], to the tag population. If this field is NULL, countCriteria must be zero.

7.1.4.17 SelectCriterion Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)

	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	action	The action that is to be applied to matching and/or non-matching tags (as defined by the mask field).
	mask	The mask that will be applied to a tag to determine if it is matching or non-matching.

7.1.4.18 SelectMask Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	bank	The memory bank that contains the bits that will be compared against the bit pattern specified in mask. For a tag mask, RESERVED is not a valid value.
	count	The number of bits in the mask. A length of zero will cause all tags to match. If (offset+count) falls beyond the end of the memory bank, the tag is considered non-matching. Valid values are 0 to 255, inclusive.
	m_mask	A buffer that contains a left-justified bit array that represents that bit pattern to match
	offset	The offset, in bits, from the start of the memory bank, of the first bit that will be matched against the mask. If offset falls beyond the end of the

		memory bank, the tag is considered non-matching.

Properties

	Name	Description
	mask	A buffer that contains a left-justified bit array that represents that bit pattern to match – i.e., the most significant bit of the bit array appears in the most-significant bit (i.e., bit 7) of the first byte of the buffer (i.e., mask[0]). All bits beyond count are ignored. For example, if the application wished to find tags with the following 12 bits 1000.1100.1101, starting at offset 16 in the EPC memory bank, then the fields would be set as follows: bank = RFID_18K6C_MEMORY_BANK_EPC offset = 16 count = 12 mask[0] = 0x8C (1000.1100) mask[1] = 0xD? (1101.????)

7.1.4.19 SingulationAlgorithmParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	length_	Structure size

Properties

	Name	Description
	length_	Structure size

7.1.4.20 SingulationCriteria Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	countCriteria	The number of singulation criteria in the array pointed to by the pCriteria field. This field must be greater than or equal to zero and less than or equal to the maximum number of post-singulation criteria as specified in [MAC-EDS]. Calling SetPostMatchCriteria(SingulationCriterion[]) with this field set to zero results in disabling all post-singulation criteria (i.e., even if the POST_MATCH flag is provided to the appropriate RFID_18K6CTag* function, no post-singulation matching will be performed – the result is that all tags are considered matching). If this field is zero, pCriteria may be NULL.
	pCriteria	A pointer to an array, containing countCriteria entries, of post- singulation criterion structures that are to be applied sequentially, beginning with pCriteria[0], to singulated tags. This must not be NULL. If this field is NULL, countCriteria must be zero.

7.1.4.21 SingulationCriterion Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)

	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	mask	The mask that will be applied to the tag's Electronic Product Code to determine if it is matching or non-matching.
	match	Determines if the associated tag-protocol operation will be applied to tags that match the mask or not. A non-zero value indicates that the tag-protocol operation should be applied to tags that match the mask. A value of zero indicates that the tag-protocol operation should be applied to tags that do not match the mask.

7.1.4.22 SingulationMask Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	count	The number of bits in the mask. A length of zero will cause all tags to match. If (offset+count) falls beyond the end of the EPC, the tag is considered non-matching. Valid values are 0 to 496, inclusive.
	m_mask	
	offset	The offset in bits, from the start of the Electronic Product Code (EPC), of the first bit that will be matched against the mask. If offset falls beyond the end of EPC, the tag is considered non-matching.

Properties

	Name	Description
	mask	A buffer that contains a left-justified bit array that represents that bit pattern to match – i.e., the most significant bit of the bit array appears in the most-significant bit (i.e., bit 7) of the first byte of the buffer (i.e., mask[0]). All bits beyond count are ignored. For example, if the application wished to find tags with the following 16 bits 1011.1111.1010.0101, starting at offset 20 in the Electronic Product Code, then the fields would be set as follows: offset = 20 count = 16 mask[0] = 0xBF (1011.1111) mask[1] = 0xA5 (1010.0101)

7.1.4.23 TAG_ACCESS_PKT Class

Methods

	Name	Description
	Equals	Indicates whether this instance and a specified object are equal. (Inherited from ValueType.)
	GetHashCode	Returns the hash code for this instance. (Inherited from ValueType.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	ToString	Returns the fully qualified type name of this instance. (Inherited from ValueType.)

Fields

	Name	Description
	cmd	ISO 18000-6C access command
	data	Tag Access data
	error_code	Please check success flags first If the tag backscattered an error (i.e. the tag backscatter error flag is set), this value is the error code that the tag backscattered. Values are: 0x00 – general error (catch-all for errors not covered by codes) 0x03 – specified memory location does not exist of the PC value is not supported by the tag 0x04 – specified memory location is locked and/or permalocked = and is not writeable 0x0B – tag has insufficient power to perform the memory write 0x0F – tag does not support error-specific codes
	ms_ctr	Current millisecond timer/counter

Properties

	Name	Description
	IsAckTimeout	ACK timeout flag: false = Tag responded within timeout.

		true = Tag failed to respond within timeout.
	IsBackscatterError	Tag backscatter error flag: false = Tag did not backscatter an error. true = Tag backscattered an error. See error_code field.
	IsCRCInvalid	CRC invalid flag: false = CRC was valid true = CRC was invalid
	IsError	false = Access operation succeeded true = An error occurred. If one of the following error-specific bit fields does not indicate an error, the error code appears in the data field.

7.1.4.24 TagBlockPermalockParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	Count	
	Flags	Flag - Zero or combination of Select or Post-Match
	Length	Structure size
	Mask	
	Offset	
	retryCount	Number of retries attemp to read. This field must be between 0 and 15,

		inclusive.
	setPermalock	True to set permalock, otherwise read it state

7.1.4.25 TagCallbackInfo Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	Count	total count
	crcInvalid	Crc error flag
	Epc	EPC Data
	Index	Index number, First come with small number.
	Name	RFID Device name
	Pc	PC Data
	rssi	The Receive Signal Strength Indicator (RSSI).

7.1.4.26 TagGroup Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance.

		(Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	Selected	Specifies the state of the selected (SL) flag for tags that will have the operation applied to them.
	Session	Specifies which inventory session flag (i.e., S0, S1, S2, or S3) will be matched against the inventory state specified by target.
	target	Specifies the state of the inventory session flag (i.e., A or B), specified by session, for tags that will have the operation applied to them.

7.1.4.27 TagInventoryParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	flags	Flag - Zero or combination of Select or Post-Match
	tagStopCount	The maximum number of tags to which the tag-protocol operation will be applied. If this number is zero, then the operation is applied to all tags that match the selection, and optionally post-singulation, match criteria. If this number is non-zero, the antenna-port dwell-time and inventory-round-count constraints still apply, however the operation will be prematurely terminated if the maximum number of tags have the

		tag-protocol operation applied to them.

7.1.4.28 TagKillParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	extCommand	Extended Kill command
	flags	Flag - Zero or combination of Select or Post-Match
	killPassword	The kill password for the tags. A value of zero indicates no kill password.
	Length	Structure size
	retryCount	Number of retries attemp to read. This field must be between 0 and 15, inclusive.

7.1.4.29 TagLockParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)

	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	accessPasswordPermissions	The access permissions for the tag's access password.
	epcMemoryBankPermissions	The access permissions for the tag's EPC memory bank.
	flags	Flag - Zero or combination of Select or Post-Match
	killPasswordPermissions	The access permissions for the tag's kill password.
	length	Structure size
	retryCount	Number of retries attemp to read. This field must be between 0 and 15, inclusive.
	tidMemoryBankPermissions	The access permissions for the tag's TID memory bank.
	userMemoryBankPermissions	The access permissions for the tag's user memory bank.

7.1.4.30 TagPerm Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPasswordPermissions	The access permissions for the tag's access password.
	epcMemoryBankPermissions	The access permissions for the tag's EPC memory bank.
	killPasswordPermissions	The access permissions for the tag's kill password.
	tidMemoryBankPermissions	The access permissions for the tag's TID memory bank.

	userMemoryBankPermissions	The access permissions for the tag's user memory bank.

7.1.4.31 TagPostMatchParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	count	The number of bits in the mask. A length of zero will cause all tags to match. If (offset+count) falls beyond the end of the EPC, the tag is considered non-matching. Valid values are 0 to 496, inclusive.
	mask	A buffer that contains a left-justified bit array that represents that bit pattern to match – i.e., the most significant bit of the bit array appears in the most-significant bit (i.e., bit 7) of the first byte of the buffer (i.e., mask[0]). All bits beyond count are ignored. For example, if the application wished to find tags with the following 16 bits 1011.1111.1010.0101, starting at offset 20 in the Electronic Product Code, then the fields would be set as follows: offset = 20 count = 16 mask[0] = 0xBF (1011.1111) mask[1] = 0xA5 (1010.0101)
	offset	The offset in bits, from the start of the Electronic Product Code (EPC), of the first bit that will be matched against the mask. If offset falls beyond the end of EPC, the tag is considered non-matching.
	toggleTarget	A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e., A to B or B to A) and another inventory cycle run. A non- zero value indicates that the target should be toggled. A zero value indicates that the target should not be toggled.

7.1.4.32 TagRangingParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	flags	Flag - Zero or combination of Select or Post-Match
	tagStopCount	The maximum number of tags to which the tag-protocol operation will be applied. If this number is zero, then the operation is applied to all tags that match the selection, and optionally post-singulation, match criteria. If this number is non-zero, the antenna-port dwell-time and inventory-round-count constraints still apply, however the operation will be prematurely terminated if the maximum number of tags have the tag-protocol operation applied to them.

7.1.4.33 TagRangingParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)

	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	count	The number of 16-bit words that will be read. This field must be between 1 and 31, inclusive.
	length	Structure size
	m_epc	An EPC to the 16-bit values to write to the tag's memory bank.
	offset	The offset, in the memory bank, of the first 16-bit word to read.
	retryCount	Number of retrial will retry if write failure

Properties

	Name	Description
	Epc	An EPC to the 16-bit values to write to the tag's memory bank.

7.1.4.34 TagReadPcParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	length	Structure size
	retryCount	Number of retrial will retry if write failure

Properties

	Name	Description
	Pc	A PC to the 16-bit values to read from the tag's memory bank.

7.1.4.35 TagReadProtectParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	flags	Flag - Zero or combination of Select or Post-Match
	retryCount	Max retry count can't excess 15

7.1.4.36 TagReadPwdParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)

	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	length	Structure size
	retryCount	Number of retrial will retry if read failure

Properties

	Name	Description
	password	A password to the 32-bit values to read from the tag's memory bank.

7.1.4.37 TagReadTidParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	count	The number of 16-bit words that will be read. This field must be between 1 and 31, inclusive.
	length	Structure size
	offset	The offset, in the memory bank, of the first 16-bit word to read.
	pData	A pointer to the 16-bit values to read from the tag's memory bank.
	retryCount	Number of retrial will retry if read failure

Properties

	Name	Description
	tid	An array to the 16-bit values to read from tag's memory bank.

7.1.4.38 TagReadUserParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	count	The number of 16-bit words that will be read. This field must be between 1 and 31, inclusive.
	length	Structure size
	offset	The offset, in the memory bank, of the first 16-bit word to read.
	retryCount	Number of retrial will retry if read failure

Properties

	Name	Description
	pData	An array to the 16-bit values to read from the tag's memory bank.

7.1.4.39 TagSearchingParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)

	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	avgRssi	averaging the RSSI value during search

7.1.4.40 TagSelectedParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	epcMask	A buffer that contains a left-justified bit array that represents that bit pattern to match – i.e., the most significant bit of the bit array appears in the most-significant bit (i.e., bit 7) of the first byte of the buffer (i.e., mask[0]). All bits beyond count are ignored. For example, if the application wished to find tags with the following 16 bits 1011.1111.1010.0101, starting at offset 20 in the Electronic Product Code, then the fields would be set as follows: offset = 20 count = 16 mask[0] = 0xBF (1011.1111) mask[1] = 0xA5 (1010.0101)

	epcMaskLength	epc mask length in bit, e.g. epc = 0x3000, length = 4 * 8 = 32 bits, epc = 0x300, length = 3 * 8 = 24 bits
	epcMaskOffset	epc mask offset in bit, note : if enable PC mask, this parameter is ignored.
	flags	A mask that indicates current parameters enalbe or not.

7.1.4.41 TagWriteEpcParms Class

Methods

	Name	Description
	Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString	Returns a String that represents the current Object. (Inherited from Object.)

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	count	The number of 16-bit words that will be written. This field must be between 1 and 31, inclusive.
	epc	A new epc to the 16-bit values to write to the tag's memory bank.
	length	Structure size
	offset	The offset, in the memory bank, of the first 16-bit word to write.
	retryCount	Number of retrial will retry if write failure

Properties

	Name	Description
	Length	Structure size

7.1.4.42 TagWritePcParms Class

Fields

	Name	Description

	accessPassword	The access password for the tags. A value of zero indicates no access password.
	length	Structure size
	pc	A new pc to the 16-bit values to write to the tag's memory bank.
	retryCount	Number of retrial will retry if write failure

7.1.4.43 TagWritePwdParms Class

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	length	Structure size
	password	A new password to the 32-bit values to write to the tag's memory bank.
	retryCount	Number of retrial will retry if write failure

7.1.4.44 TagWriteUserParms Class

Fields

	Name	Description
	accessPassword	The access password for the tags. A value of zero indicates no access password.
	count	The number of 16-bit words that will be written.
	length	Structure size
	offset	The offset, in the memory bank, of the first 16-bit word to write.
	pData	A array to the 16-bit values to write to the tag's memory bank.
	retryCount	Number of retrial will retry if write failure

Fields

	Name	Description
	Length	Structure length

7.1.4.45 Version Class

Fields

	Name	Description
	major	Major
	minor	Minor
	patch	Patch

7.1.4.46 WriteParmsBase Class

Fields

	Name	Description
	length_	Structure size

Properties

	Name	Description
	length_	Structure size

7.1.4.47 WriteRandomParms Class

Fields

	Name	Description
	bank	The memory bank from which to write. Valid values are: MemoryBank.RESERVED MemoryBank.EPC MemoryBank.TID MemoryBank.USER
	count	The number of 16-bit words to be written to the tag's specified memory bank. For version 1.1 of the RFID Reader Library, this parameter must contain a value between 1 and 8, inclusive.
	length_	Structure size (Inherited from WriteParmsBase.)
	pData	A buffer of count 16-bit values to be written to the tag's specified memory bank. The high-order byte of pData[n] will be written to the tag's memory-bank byte at pOffset[n]. The low-order byte will be written to the next byte. For example, if pOffset[n] is 2 and pData[n] is 0x1122, then the tag-memory byte at 16-bit offset 2 (byte offset 4) will have 0x11 written to it and the next byte (byte offset 5) will have 0x22 written to it. This field must not be NULL.
	pOffset	An array of count 16-bit values that specify 16-bit tag- memory-bank offsets, with zero being the first 16-bit word in the memory bank, where the corresponding 16-bit words in the pData array will be written. i.e., the 16-bit word in pData[n] will be written to the 16-bit tag-memory-bank offset contained in pOffset[n]. This field must not be NULL.
	reserved	Reserved for future use. Set to zero.

Properties

	Name	Description
	length_	Structure size (Inherited from WriteParmsBase.)

7.1.4.48 WriteSequentialParms Class

Fields

	Name	Description

	bank	The memory bank from which to write. Valid values are: MemoryBank.RESERVED MemoryBank.EPC MemoryBank.TID MemoryBank.USER
	count	The number of 16-bit words to be written to the tag's specified memory bank. For version 1.1 of the RFID Reader Library, this parameter must contain a value between 1 and 8, inclusive.
	length_	Structure size (Inherited from WriteParmsBase.)
	offset	The offset of the first 16-bit word, where zero is the first 16-bit word in the memory bank, to write in the specified memory bank.
	pData	A buffer of count 16-bit values to be written sequentially to the tag's specified memory bank. The high-order byte of pData[n] will be written to the tag's memory-bank byte at 16-bit offset (offset+n). The low-order byte will be written to the next byte. For example, if offset is 2 and pData[0] is 0x1122, then the tag-memory byte at 16-bit offset 2 (byte offset 4) will have 0x11 written to it and the next byte (byte offset 5) will have 0x22 written to it. This field must not be NULL.

Properties

	Name	Description
	length	Structure size (Inherited from WriteParmsBase.)

7.1.5 CSLibrary.barcode Namespace

7.1.5.1 Classes Overview

Classes

	Class	Description
	Start	Start barcode scanning.
	Stop	Stop barcode scanning

7.1.6 CSLibrary.notification Namespace

7.1.6.1 Classes Overview

Classes

	Class	Description

7.2 Error Messages

Every function will return an enumerated result. The error messages are listed below.


```
/// <summary>
/// function result value definitions
/// </summary>
public enum Result : int
{
    /// <summary>
    /// Success
    /// </summary>
    OK            =  0,

    /// <summary>
    /// Attempted to open a radio that is already open
    /// </summary>
    ALREADY_OPEN = -9999,

    /// <summary>
    /// Buffer supplied is too small
    /// </summary>
    BUFFER_TOO_SMALL,

    /// <summary>
    /// General failure
    /// </summary>
    FAILURE,

    /// <summary>
    /// Failed to load radio bus driver
    /// </summary>
    DRIVER_LOAD,

    /// <summary>
    /// Library cannot use version of radio bus driver present on system
    /// </summary>
    DRIVER_MISMATCH,

    /// <summary>
    /// Operation cannot be performed while library is in emulation mode
    /// </summary>
    EMULATION_MODE,

    /// <summary>
    /// Antenna number is invalid
    /// </summary>
    INVALID_ANTENNA,
```



```
/// <summary>
/// Radio handle provided is invalid
/// </summary>
INVALID_HANDLE,

/// <summary>
/// One of the parameters to the function is invalid
/// </summary>
INVALID_PARAMETER,

/// <summary>
/// Attempted to open a non-existent radio
/// </summary>
NO_SUCH_RADIO,

/// <summary>
/// Library has not been successfully initialized
/// </summary>
NOT_INITIALIZED,

/// <summary>
/// Function not supported
/// </summary>
NOT_SUPPORTED,

/// <summary>
/// Op cancelled by cancel op func, close radio, or library shutdown
/// </summary>
OPERATION_CANCELLED,

/// <summary>
/// Library encountered an error allocating memory
/// </summary>
OUT_OF_MEMORY,

/// <summary>
/// The operation cannot be performed because the radio is currently busy
/// </summary>
RADIO_BUSY,

/// <summary>
/// The underlying radio module encountered an error
/// </summary>
```



```
RADIO_FAILURE,

/// <summary>
/// The radio has been detached from the system
/// </summary>
RADIO_NOT_PRESENT,

/// <summary>
/// The RFID library function is not allowed at this time.
/// </summary>
CURRENTLY_NOT_ALLOWED,

/// <summary>
/// The radio module's MAC firmware is not responding to requests.
/// </summary>
RADIO_NOT_RESPONDING,

/// <summary>
/// The MAC firmware encountered an error while initiating the nonvolatile
/// memory update.  The MAC firmware will return to its normal idle state
/// without resetting the radio module.
/// </summary>
NONVOLATILE_INIT_FAILED,

/// <summary>
/// An attempt was made to write data to an address that is not in the
/// valid range of radio module nonvolatile memory addresses.
/// </summary>
NONVOLATILE_OUT_OF_BOUNDS,

/// <summary>
/// The MAC firmware encountered an error while trying to write to the
/// radio module's nonvolatile memory region.
/// </summary>
NONVOLATILE_WRITE_FAILED,

/// <summary>
/// The underlying transport layer detected that there was an overflow
/// error resulting in one or more bytes of the incoming data being
/// dropped.  The operation was aborted and all data in the pipeline was
/// flushed.
/// </summary>
RECEIVE_OVERFLOW,
```



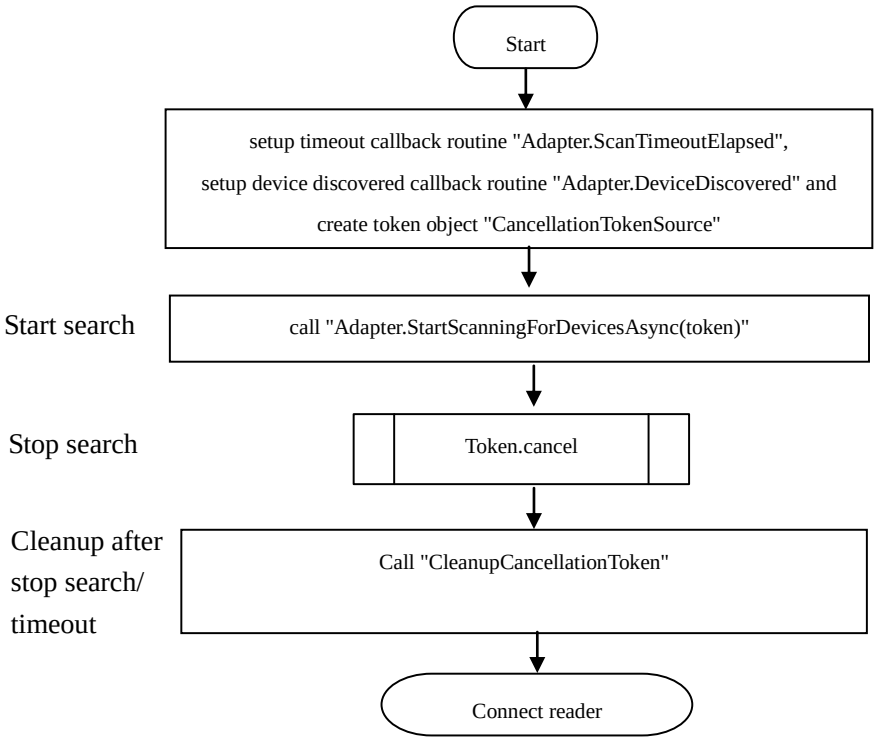
```
        /*------------------WaterYu define----------------*/
                NET_RESET,

                NET_DISCONNECT,

                NET_NOT_CONNECTED,

                /// <summary>
                /// Null object exception
                /// </summary>
                NULL_OBJECT,

                /// <summary>
                /// Invalid radio index
                /// </summary>
                INVALID_RADIO_INDEX,

                /// <summary>
                /// Invalid index
                /// </summary>
                INVALID_INDEX,

                /// <summary>
                ///
                /// </summary>
                COUNTRY_NOT_SUPPORTED,

                /// <summary>
                /// Can't connect to device
                /// </summary>
                CONNECT_DEVICE_FAILED,
        };
```


8 Programming Flow

This chapter describes the detail programming flow for normal operations on CS108, including Bluetooth search, Connect/Disconnect CS108, Setting Frequency/Country/Parameters, Inventory, Read, Write, Lock, Kill, Search.

All sample code in this chapter can be found in the under "*CS108 Example Code*"

8.1 Bluetooth Search

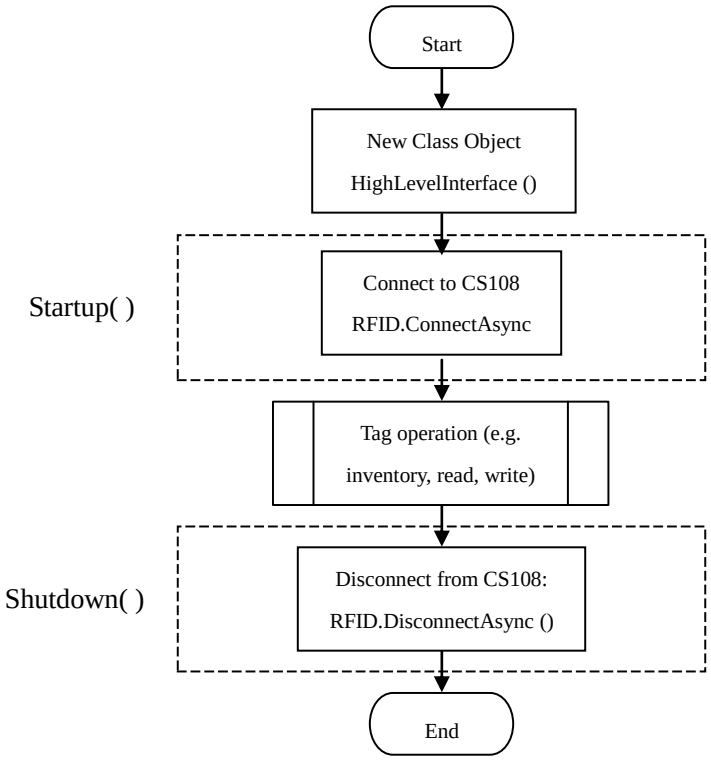


8.2 Connect/Disconnect CS108

This section describes the flow for Bluetooth connecting to and disconnecting from CS108.

8.2.1 Flow-chart

The standard connect and disconnect sequence for CS108 is as below.



8.2.2 Sample Code

The following is a piece of sample code that shows how to initiate the CS108 integrated reader and also how to startup the reader.

---------- Code (ViewModelMainMenu.cs) ----------

```
private async void Connect ()
```



```
        {
            var _services = await
_device.GetServiceAsync(Guid.Parse("00009800-0000-1000-8000-00805f9b34fb"));

            if (_services == null)
            {
                Close(this);
            }

            await BleMvxApplication._reader.ConnectAsync(_services);
        }
```

8.3 RFID Country/Frequency Setting

Refer to Appendix F for model and regulatory region details.

8.3.1 Get device country code

```
/// <summary>
/// GetCountryCode
/// </summary>
/// <returns>Result</returns>
public Result GetCountryCode(ref uint code)
```

8.3.2 Check device only hopping can be set

```
/// <summary>
/// If true, it can only set to hopping channel.
/// </summary>
public bool IsHoppingChannelOnly
```

8.3.3 Check device only fixed can be set

```
/// <summary>
/// If true, it can only set to fixed channel.
/// Otherwise, both fixed and hopping can be set.
/// </summary>
public bool IsFixedChannelOnly
```

8.3.4 Check current selected fixed channel

```
/// <summary>
/// Get Fixed frequency channel
/// </summary>
public bool IsFixedChannel
```

8.3.5 Get active region code

```
/// <summary>
/// Available region you can use
/// </summary>
public List<RegionCode> GetActiveRegionCode()
```

8.3.6 Get current selected region code

```
/// <summary>
```

```
        /// Get Current Region Profile
        /// </summary>
        public RegionCode SelectedRegionCode
```

8.3.7 Get current selected frequency channel

```
        /// <summary>
        /// Get Current Selected Frequency Channel
        /// </summary>
        public uint SelectedChannel
```

8.3.8 Set fixed frequency channel

```
        /// <summary>
        /// Set Fixed Frequency Channel
        /// All region can be used to set a fixed channel
        /// </summary>
        /// <param name="prof">Region Code</param>
        /// <param name="channel">Channel number start from zero, you can get the available channels
        ///                                                                 from CSLibrary.HighLevelInterface.AvailableFrequencyTable(CSLibrary.Constants.RegionCode)</param>
        public Result SetFixedChannel(RegionCode prof = RegionCode.CURRENT, uint channel = 0)
```

8.3.9 Set frequency to the specific order

```
        /// <summary>
        /// Set frequency to the specific order
        /// </summary>
        /// <param name="prof">Country Profile</param>
        /// <returns>Result</returns>
        public Result SetHoppingChannels(RegionCode prof)
```

8.3.10 Set to hopping channels

```
        /// <summary>
        /// Reset current frequency profile
        /// </summary>
        /// <returns></returns>
        public Result SetHoppingChannels()
```


8.3.11 Set to frequency agile mode

```
/// <summary>
/// Set to frequency agile mode
/// </summary>
/// <param name="prof">Country Profile</param>
/// <returns>Result</returns>
public Result SetAgileChannels(RegionCode prof)
```

8.3.12 Reset frequency setting to power up default

```
/// <summary>
/// Reset frequency order to power up default
/// </summary>
/// <returns></returns>
public Result SetDefaultChannel()
```

8.4 RFID Operation Parameters Setting

There are several RFID parameters can be configured on CS108:

- Power Level
- Link Profile
- Operation Mode
- Tag Group
- Q Parameter
- Selection Criteria
- PostMatch Criteria

8.4.1 Setting Power Level

The transmit power level can be configured by the function `SetPowerLevel(uint pwrlevel)` as below. The configurable pwrlevel is in the range from 0 to 300, that is the ten times of the transmit power in dBm. For example, pwrlevel = 300 means the transmit power is 30dBm.

```
/// <summary>
/// Set Power Level(Max 300)...
/// </summary>
/// <param name="pwrlevel">Power Level Max. 300</param>
/// <returns></returns>
public Result SetPowerLevel(uint pwrlevel)
```

8.4.2 Setting Link Profile

The link profile can be configured by the function `SetCurrentLinkProfile(uint profile)` as below. The configurable profile is in the range from 0 to 3.

```
/// <summary>
/// Set Current Link Profile (0 to 3)...
/// </summary>
/// <param name="profile">Link Profile 0,1,2,3</param>
/// <returns></returns>
public Result SetCurrentLinkProfile(uint profile)
```

8.4.3 Setting Operation Mode

The operation mode of the reader can be configured by function `SetOperationMode(RadioOperationMode mode)`. The mode value could be configured to CONTINUOUS (continuous mode) or NONCONTINUOUS (non-continuous mode). The default

value is NONCONTINUOUS.

```
/// <summary>
/// Sets the operation mode of RFID radio module.  By default, when
/// an application opens a radio, the RFID Reader Library sets the
/// reporting mode to non-continuous.  An RFID radio module's
/// operation mode will remain in effect until it is explicitly changed
/// via RFID_RadioSetOperationMode, or the radio is closed and re-
/// opened (at which point it will be set to non-continuous mode).
/// The operation mode may not be changed while a radio module is
/// executing a tag-protocol operation.
/// </summary>
/// <param name="mode">The operation mode for the radio module.</param>
/// <returns></returns>
public Result SetOperationMode(RadioOperationMode mode)
```

8.4.4 Setting Tag Group

The tag-protocol operation can be configured to be applied to any or one of the tag groups. This tag group is set by the function SetTagGroup(TagGroup tagGroup).

```
/// <summary>
/// Once the tag population has been partitioned into disjoint groups, a subsequent
/// tag-protocol operation (i.e., an inventory operation or access command) is then
/// applied to one of the tag groups.
/// </summary>
/// <param name="tagGroup">Tag Interest</param>
/// <returns></returns>
public Result SetTagGroup(TagGroup tagGroup)
```

8.4.5 Setting Q Parameter

There are 4 main inventory algorithms (one fixed Q and three variable Q) on CS108 to support the ISO18000-6C (or Gen2) protocol. The four algorithms are Fixed Q Algorithm, Dynamic Q Algorithm, Dynamic Q Adjustment Algorithm and Dynamic Q Adjustment Threshold Algorithm. You could configure the reader to follow one of the algorithms. Please refer to Chapter 10 for the details about the inventory algorithms and Q parameter.

Fixed Q Algorithm:

```
/// <summary>
/// The parameters for the fixed-Q algorithm, MAC singulation algorithm 0
/// If running a same operation, it only need to config once times
```



```
        /// </summary>
        /// <returns></returns>
        public Result SetFixedQParms(FixedQParms fixedQParm)
```

Dynamic Q Algorithm:

```
        /// <summary>
        /// The parameters for the dynamic-Q algorithm, MAC singulation algorithm 1
        /// </summary>
        /// <returns></returns>
        public Result SetDynamicQParms(DynamicQParms dynParm)
```

8.4.6 Setting Selection Criteria

You could configure the tag selection criteria for the ISO 18000-6C Select command on the reader by function SetSelectCriteria(SelectCriterion[] critlist).

```
        /// <summary>
        /// Configures the tag-selection criteria for the ISO 18000-6C select
        /// command.  The supplied tag-selection criteria will be used for any
        /// tag-protocol operations (i.e., Inventory, etc.) in
        /// which the application specifies that an ISO 18000-6C select
        /// command should be issued prior to executing the tag-protocol
        /// operation (i.e., the FLAGS.SELECT flag is provided to
        /// the appropriate RFID_18K6CTag* function).  The tag-selection
        /// criteria will stay in effect until the next call to
        /// SetSelectCriteria.  Tag-selection criteria may not
        /// be changed while a radio module is executing a tag-protocol
        /// operation.
        /// </summary>
        /// <param name="critlist">
        /// SelectCriteria array, containing countCriteria entries, of selection
        /// criterion structures that are to be applied sequentially, beginning with
        /// pCriteria[0], to the tag population.  If this field is NULL,
        /// countCriteria must be zero.
        ///</param>
        /// <returns></returns>
        public Result SetSelectCriteria(SelectCriterion[] critlist)
```

or

```
        public Result SetSelectCriteria(uint index, SelectCriterion[] critlist)
```

8.4.7 Cancel All Selection Criteria

You could cancel all Select command on the reader by function CancelAllSelectCriteria().

```
public Result CancelAllSelectCriteria()
```

8.4.8 Setting PostMatch Criteria

You could configure the post-singulation match criteria to be used by the reader by function SetPostMatchCriteria(SingulationCriterion[] postmatch).

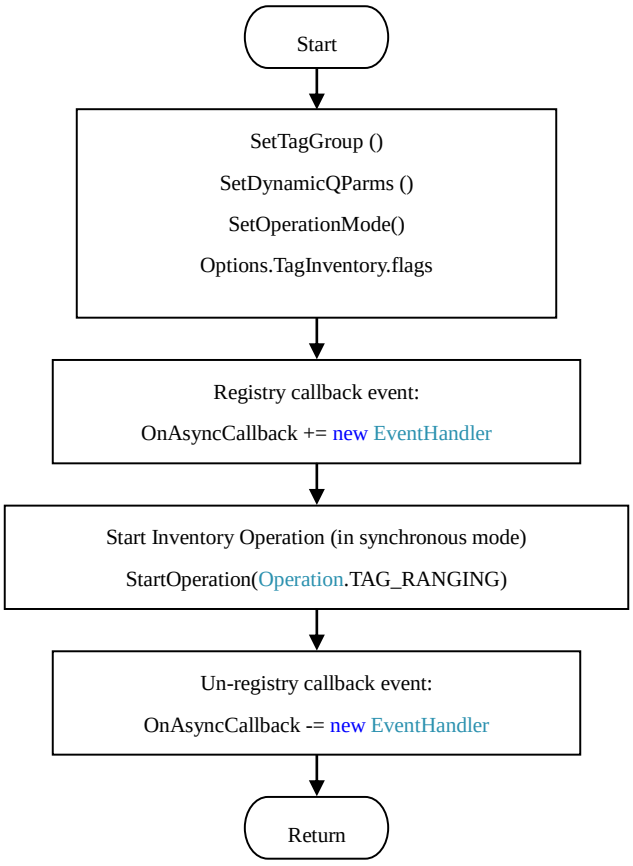
```
/// <summary>
/// Configures the post-singulation match criteria to be used by the
/// RFID radio module.  The supplied post-singulation match criteria
/// will be used for any tag-protocol operations (i.e.,
/// Inventory, etc.) in which the application specifies
/// that a post-singulation match should be performed on the tags
/// that are singulated by the tag-protocol operation (i.e., the
/// FLAGS.POST_MATCH flag is provided to the
/// appropriate RFID_18K6CTag* function).  The post-singulation
/// match criteria will stay in effect until the next call to
/// SetPostMatchCriteria.  Post-singulation match
/// criteria may not be changed while a radio module is executing a
/// tag-protocol operation.
/// </summary>
/// <param name="postmatch"> An array that specifies the post-
/// singulation match criteria that are to be
/// applied to the tag's Electronic Product Code
/// after it is singulated to determine if it is to
/// have the tag-protocol operation applied to it.
/// If the countCriteria field is zero, all post-
/// singulation criteria will be disabled.  This
/// parameter must not be NULL. </param>
/// <returns></returns>
public Result SetPostMatchCriteria(SingulationCriterion[] postmatch)
```

8.5 RFID Inventory

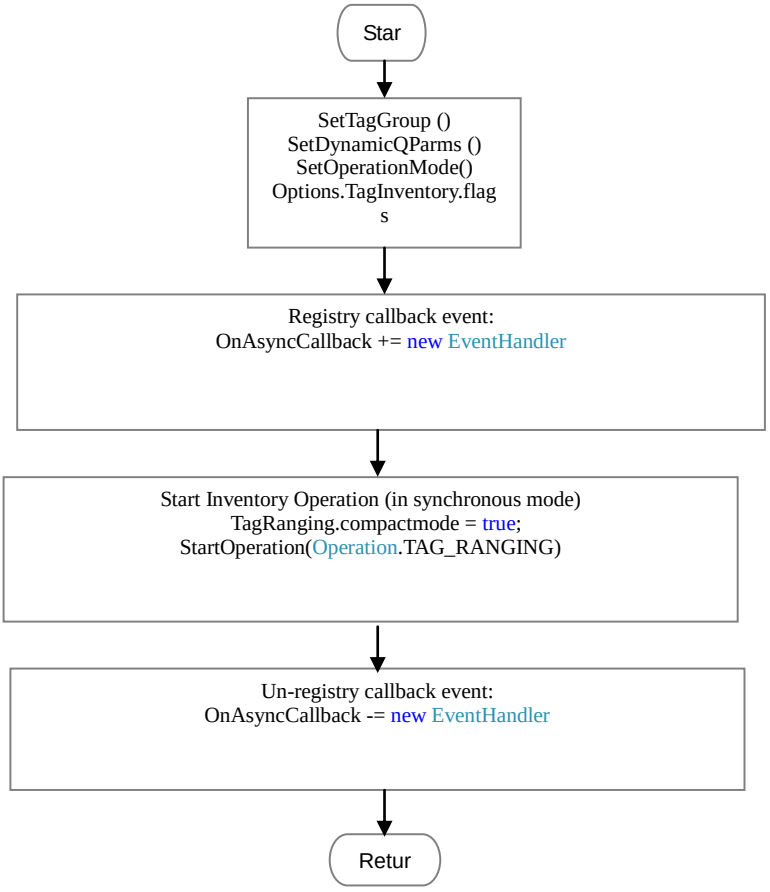
8.5.1 Flow-chart

TagInventory is the methods for doing inventory on CS108.

1) Normal Tag Inventory: This method allows custom settings of Tag Group, Inventory Algorithm (Dynamic Q) and Q parameter



2) Tag Inventory Compact Mode : This is fast inventory mode (only included EPC and RSSI). The method allows custom settings of Tag Group, Inventory Algorithm (Dynamic Q) and Q parameter



8.5.2 Sample Code (Compact Mode)

This example shows how to configure the reader to run in non-continuous / continuous mode inventory to read any tag with Dynamic Q algorithm (starting Q value is 7).

- TagInventory: allow custom settings of Tag Group, Inventory Algorithm (Dynamic Q) and Q parameter

---------- Code (ViewModelInventorynScan.cs) ----------

```
        void StartInventory()
        {
            EPCData.Clear();
            TagData.Clear();

            StartTagCount();
            if (BleMvxApplication._config.RFID_OperationMode ==
CSLibrary.Constants.RadioOperationMode.CONTINUOUS)
```



```
            {
                _startInventory = false;
                _startInventoryButtonText = "Stop Inventory";
            }

            // Setting 1

BleMvxApplication._reader.rfid.SetInventoryTimeDelay((uint)BleMvxApplication._config.RF
ID_InventoryDelayTime);

BleMvxApplication._reader.rfid.SetInventoryDuration((uint)BleMvxApplication._config.RFI
D_DWellTime);

BleMvxApplication._reader.rfid.SetPowerLevel((uint)BleMvxApplication._config.RFID_Power
);

            // Setting 2

BleMvxApplication._reader.rfid.SetOperationMode(BleMvxApplication._config.RFID_Operatio
nMode);

BleMvxApplication._reader.rfid.SetTagGroup(BleMvxApplication._config.RFID_TagGroup);

BleMvxApplication._reader.rfid.SetCurrentSingulationAlgorithm(BleMvxApplication._config
.RFID_Algorithm);

BleMvxApplication._reader.rfid.SetCurrentLinkProfile(BleMvxApplication._config.RFID_Pro
file);

            // Setting 3

BleMvxApplication._reader.rfid.SetDynamicQParms(BleMvxApplication._config.RFID_DynamicQ
Parms);

            // Setting 4

BleMvxApplication._reader.rfid.SetFixedQParms(BleMvxApplication._config.RFID_FixedQParm
s);

            // Select Criteria filter
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK("709999");

            CSLibrary.Structures.SelectCriterion[] critlist = new
```

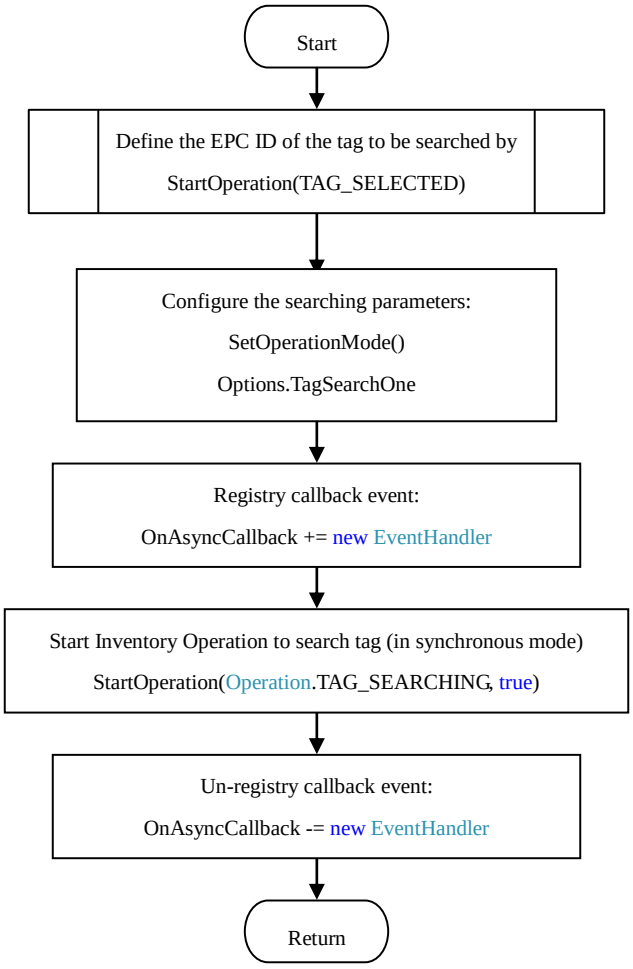


```
CSLibrary.Structures.SelectCriterion[1];
            critlist[0] = new CSLibrary.Structures.SelectCriterion();
            critlist[0].mask = new
CSLibrary.Structures.SelectMask(CSLibrary.Constants.MemoryBank.EPC, 0x20, 24,
BleMvxApplication._reader.rfid.Options.TagSelected.epcMask.ToBytes());
            critlist[0].action = new
CSLibrary.Structures.SelectAction(CSLibrary.Constants.Target.SELECTED,
CSLibrary.Constants.Action.ASLINVA_DSLINVB, 0);

            BleMvxApplication._reader.rfid.SetSelectCriteria(critlist);

            // Post Match Criteria filter
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(BleMvxApplication._config.MASK_EPC);

            CSLibrary.Structures.SingulationCriterion[] sel = new
CSLibrary.Structures.SingulationCriterion[1];
            sel[0] = new CSLibrary.Structures.SingulationCriterion();
            sel[0].match = BleMvxApplication._config.MASK_Enable ? 0U : 1U;
            sel[0].mask = new
CSLibrary.Structures.SingulationMask(BleMvxApplication._config.MASK_Offset,
(uint)(BleMvxApplication._config.MASK_EPC.Length * 4),
BleMvxApplication._reader.rfid.Options.TagSelected.epcMask.ToBytes());
            BleMvxApplication._reader.rfid.SetPostMatchCriteria(sel);

            BleMvxApplication._reader.rfid.Options.TagRanging.compactmode = true; //
Set to false if use normal inventory
            // Start Inventory

BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_RANGING
);
        }
```

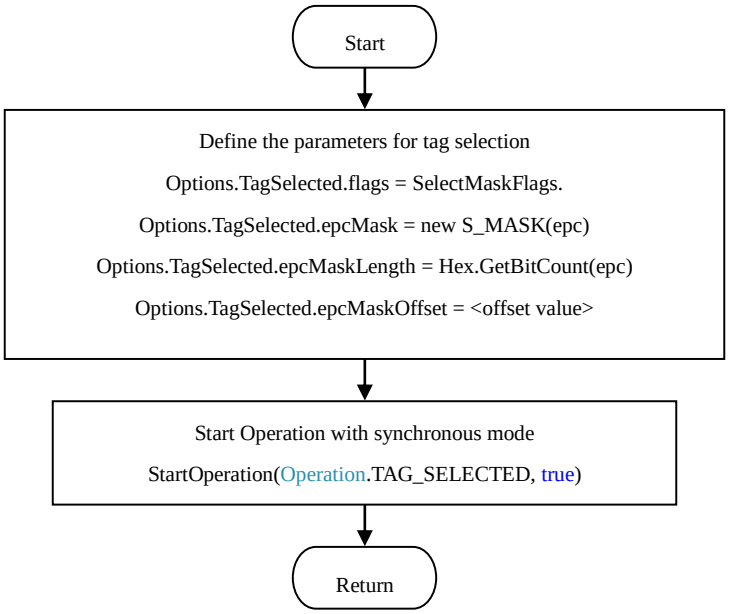
8.6 Search a tag

8.6.1 Flow-chart

TAG_SELECTED: This method allows searching for tag with the default Tag Inventory parameters.



TagSelected.Run()



8.6.2 Sample Code

---------- Code (ViewModelGeiger.cs) ----------

```
        void StartGeigerButtonClick()
        {
            RaisePropertyChanged(() => entryEPC);

            BleMvxApplication._reader.rfid.Options.TagSelected.flags =
CSLibrary.Constants.SelectMaskFlags.ENABLE_TOGGLE;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(_entryEPC);
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskOffset = 0;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskLength =
(uint)(_entryEPC.Length) * 4;

BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_SELECTE
D);
```

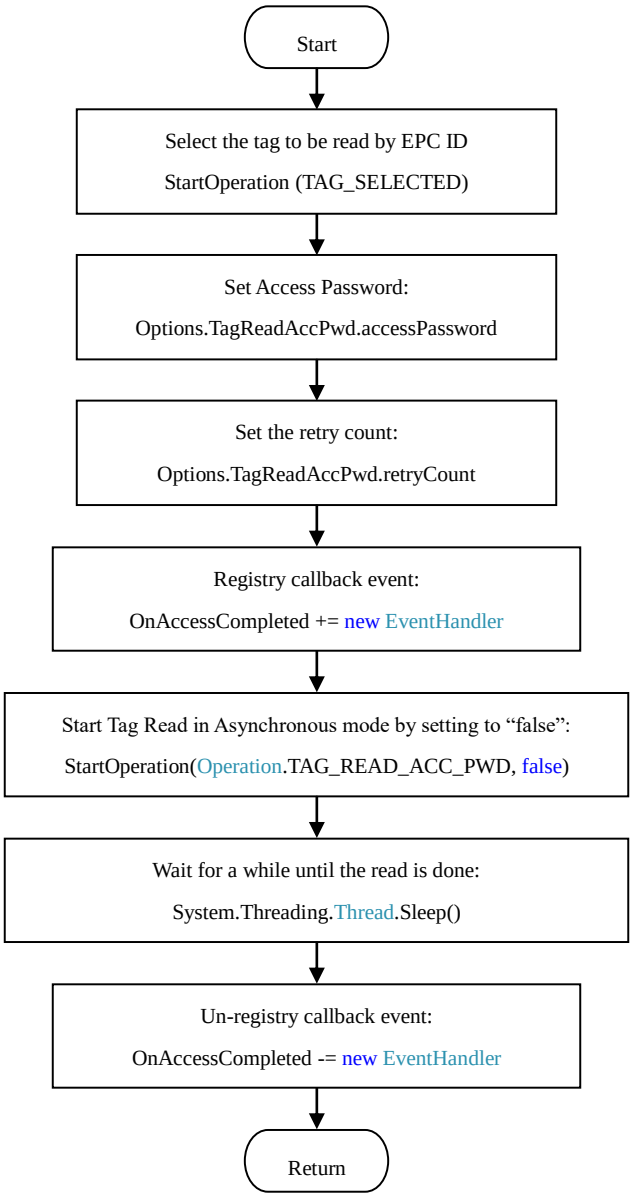


```
BleMvxApplication._reader.rfid.SetOperationMode(CSLibrary.Constants.RadioOperationMode.
CONTINUOUS);
            //BleMvxApplication._reader.rfid.Options.TagSearchOne.avgRssi =
cb_averaging.Checked;

BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_SEARCHI
NG);

            RaisePropertyChanged(() => startGeigerButtonText);
        }
```

8.7 Read a Tag

8.7.1 Flow-chart

Asynchronous Mode: This method allows reading a tag's memory with data returned in asynchronous mode.



8.7.2 Sample Code

This example shows how to read a tag's memory data by in synchronous or asynchronous methods.

- RunASyncReadAccPwd: read the Access Password of a tag using "RFID.StartOperation(Operation.TAG_READ_ACC_PWD)" command in asynchronous mode

---------- Code (ViewModelReadWrite.cs) ----------

```
        void OnReadButtonButtonClick()
        {
            uint m_retry_cnt = 7;       // Max 7

            RaisePropertyChanged(() => entrySelectedEPC);
            RaisePropertyChanged(() => entrySelectedPWD);

            RaisePropertyChanged(() => switchPCIsToggled);
            RaisePropertyChanged(() => switchEPCIsToggled);
            RaisePropertyChanged(() => switchACCPWDIsToggled);
            RaisePropertyChanged(() => switchKILLPWDIsToggled);
            RaisePropertyChanged(() => switchTIDUIDIsToggled);
            RaisePropertyChanged(() => switchUSERIsToggled);

            uint accessPwd = Convert.ToUInt32(entrySelectedPWD, 16);


            if (BleMvxApplication._reader.rfid.State !=
CSLibrary.Constants.RFState.IDLE)
            {
                //MessageBox.Show("Reader is busy now, please try later.");
                return;
            }


//BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_RANGI
NG);

            if (entrySelectedEPC.Length > 64)
            {
                //MessageBox.Show("EPC too long, only selecte first 256 bit");
                BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(/*m_record.pc.ToString() + */entrySelectedEPC.Substring(0,
64));
            }
            else
                BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(/*m_record.pc.ToString() + */entrySelectedEPC);

            BleMvxApplication._reader.rfid.Options.TagSelected.flags =
```

```
CSLibrary.Constants.SelectMaskFlags.ENABLE_TOGGLE;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskOffset = 0;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskLength =
(uint)BleMvxApplication._reader.rfid.Options.TagSelected.epcMask.Length * 8;

BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_SELECTE
D);

            if (switchPCIsToggled)
            {
                //lb_ReadInfo.Text = "Start reading PC";

                BleMvxApplication._reader.rfid.Options.TagReadPC.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagReadPC.retryCount =
m_retry_cnt;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_PC
);
            }

            if (switchEPCIsToggled)
            {
                //lb_ReadInfo.Text = "Start reading EPC";

                BleMvxApplication._reader.rfid.Options.TagReadEPC.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagReadEPC.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagReadEPC.count = 6;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_EP
C);
            }

            //if access bank is checked, read it.
            if (switchACCPWDIsToggled)
            {
                //lb_ReadInfo.Text = "Start reading access pwd";

                BleMvxApplication._reader.rfid.Options.TagReadAccPwd.accessPassword =
accessPwd;
```



```
                BleMvxApplication._reader.rfid.Options.TagReadAccPwd.retryCount =
m_retry_cnt;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_AC
C_PWD);
            }

            //if kill bank is checked, read it.
            if (switchKILLPWDIsToggled)
            {
                // lb_ReadInfo.Text = "Start reading kill pwd";

                BleMvxApplication._reader.rfid.Options.TagReadKillPwd.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagReadKillPwd.retryCount =
m_retry_cnt;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_KI
LL_PWD);
            }

            //if tid bank is checked, read it.
            if (switchTIDUIDIsToggled)
            {
                // lb_ReadInfo.Text = "Start reading TID";

                BleMvxApplication._reader.rfid.Options.TagReadTid.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagReadTid.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagReadTid.offset = 0; //
uint.Parse(m_readAllBank.OffsetTid);
                BleMvxApplication._reader.rfid.Options.TagReadTid.count = 2; //
m_readAllBank.WordTid;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_TI
D);
            }

            //if user bank is checked, read it.
            if (switchUSERIsToggled)
```

```
            {
                //lb_ReadInfo.Text = "Start reading user memory";

                BleMvxApplication._reader.rfid.Options.TagReadUser.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagReadUser.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagReadUser.offset = 0; //
m_readAllBank.OffsetUser;
                BleMvxApplication._reader.rfid.Options.TagReadUser.count = 2; //
m_readAllBank.WordUser;


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_READ_US
ER);
            }
        }


        void TagCompletedEvent(object sender,
CSLibrary.Events.OnAccessCompletedEventArgs e)
        {
            if (e.access == CSLibrary.Constants.TagAccess.READ)
            {
                switch (e.bank)
                {
                    case CSLibrary.Constants.Bank.PC:
                        if (e.success)
                        {
                            entryPC =
BleMvxApplication._reader.rfid.Options.TagReadPC.pc.ToString();
                            RaisePropertyChanged(() => entryPC);
                        }
                        break;

                    case CSLibrary.Constants.Bank.EPC:
                        if (e.success)
                        {
                            entryEPC =
BleMvxApplication._reader.rfid.Options.TagReadEPC.epc.ToString();
                            RaisePropertyChanged(() => entryEPC);
                        }
                        break;
```

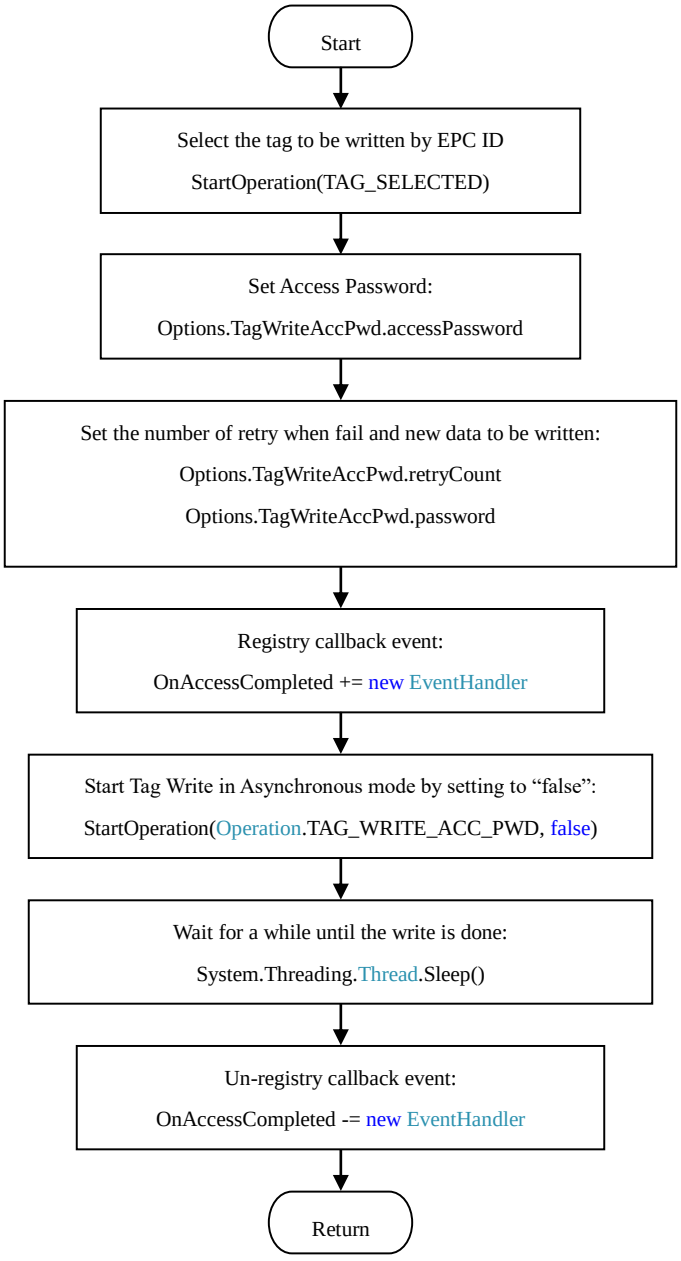


```
                    case CSLibrary.Constants.Bank.ACC_PWD:
                        if (e.success)
                        {
                            entryACCPWD =
BleMvxApplication._reader.rfid.Options.TagReadAccPwd.password.ToString();
                            RaisePropertyChanged(() => entryACCPWD);
                        }
                        break;

                    case CSLibrary.Constants.Bank.KILL_PWD:
                        if (e.success)
                        {
                            entryKILLPWD =
BleMvxApplication._reader.rfid.Options.TagReadKillPwd.password.ToString();
                            RaisePropertyChanged(() => entryKILLPWD);
                        }
                        break;

                    case CSLibrary.Constants.Bank.TID:
                        if (e.success)
                        {
                            entryTIDUID =
BleMvxApplication._reader.rfid.Options.TagReadTid.tid.ToString();
                            RaisePropertyChanged(() => entryTIDUID);
                        }
                        break;

                    case CSLibrary.Constants.Bank.USER:
                        if (e.success)
                        {
                            entryUSER =
BleMvxApplication._reader.rfid.Options.TagReadUser.pData.ToString();
                            RaisePropertyChanged(() => entryUSER);
                        }
                        break;
                }
            }
        }
```


8.8 Write a tag

8.8.1 Flow-chart

Asynchronous Mode: This method allows writing a tag's memory with data returned in asynchronous mode. Below is an example of writing access password.



8.8.2 Sample Code

This example shows how to write a tag's memory data by in asynchronous methods. For example,

- RunASyncWriteAccPwd: write new data to the Access Password field of a tag using "RFID.StartOperation(Operation.TAG_WRITE_ACC_PWD, false)" command in asynchronous mode

---------- Code (ViewModelReadWrite.cs) ----------

```
        void OnWriteButtonButtonClick()
        {
            uint m_retry_cnt = 7;       // Max 7

            RaisePropertyChanged(() => switchPCIsToggled);
            RaisePropertyChanged(() => switchEPCIsToggled);
            RaisePropertyChanged(() => switchACCPWDIsToggled);
            RaisePropertyChanged(() => switchKILLPWDIsToggled);
            RaisePropertyChanged(() => switchTIDUIDIsToggled);
            RaisePropertyChanged(() => switchUSERIsToggled);

            RaisePropertyChanged(() => entrySelectedEPC);
            RaisePropertyChanged(() => entrySelectedPWD);
            RaisePropertyChanged(() => entryPC);
            RaisePropertyChanged(() => entryEPC);
            RaisePropertyChanged(() => entryACCPWD);
            RaisePropertyChanged(() => entryKILLPWD);
            RaisePropertyChanged(() => entryTIDUID);
            RaisePropertyChanged(() => entryUSER);

            uint accessPwd = Convert.ToUInt32(entrySelectedPWD, 16);

            if (BleMvxApplication._reader.rfid.State !=
CSLibrary.Constants.RFState.IDLE)
            {
                //MessageBox.Show("Reader is busy now, please try later.");
                return;
            }

            // Can not write TID bank
            if (switchTIDUIDIsToggled)
            {

            }
```

```
            if (!(switchPCIsToggled | switchEPCIsToggled | switchACCPWDIsToggled |
switchKILLPWDIsToggled | switchUSERIsToggled))
            {
                //All unchecked
                //MessageBox.Show("Please check at least one item to write", "Warning!",
MessageBoxButtons.OK, MessageBoxIcon.Question, MessageBoxDefaultButton.Button3);
                return;
            }

            if (entrySelectedEPC.Length > 64)
            {
                //MessageBox.Show("EPC too long, only selecte first 256 bit");
                BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(/*m_record.pc.ToString() + */entrySelectedEPC.Substring(0,
64));
            }
            else
                BleMvxApplication._reader.rfid.Options.TagSelected.epcMask = new
CSLibrary.Structures.S_MASK(/*m_record.pc.ToString() + */entrySelectedEPC);

            BleMvxApplication._reader.rfid.Options.TagSelected.flags =
CSLibrary.Constants.SelectMaskFlags.ENABLE_TOGGLE;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskOffset = 0;
            BleMvxApplication._reader.rfid.Options.TagSelected.epcMaskLength =
(uint)BleMvxApplication._reader.rfid.Options.TagSelected.epcMask.Length * 8;

BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_SELECTE
D);

            //if access bank is checked, read it.
            if (switchACCPWDIsToggled)
            {
                //lb_WriteInfo.Text = "Start writing access pwd";

                BleMvxApplication._reader.rfid.Options.TagWriteAccPwd.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagWriteAccPwd.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagWriteAccPwd.password =
Convert.ToUInt32(entryACCPWD, 16);


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_WRITE_A
CC_PWD);
```



```
            }

            //if kill bank is checked, read it.
            if (switchKILLPWDIsToggled)
            {
                //lb_WriteInfo.Text = "Start writing kill pwd";

                BleMvxApplication._reader.rfid.Options.TagWriteKillPwd.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagWriteKillPwd.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagWriteKillPwd.password =
Convert.ToUInt32(entryKILLPWD, 16);


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_WRITE_K
ILL_PWD);
            }

            //if user bank is checked, read it.
            if (switchUSERIsToggled)
            {
                //lb_WriteInfo.Text = "Start writing user memory";

                BleMvxApplication._reader.rfid.Options.TagWriteUser.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagWriteUser.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagWriteUser.offset = 0; //
m_writeAllBank.OffsetUser;
                BleMvxApplication._reader.rfid.Options.TagWriteUser.count = 2; //
m_writeAllBank.WordUser;
                BleMvxApplication._reader.rfid.Options.TagWriteUser.pData =
CSLibrary.Tools.Hex.ToUshorts(entryUSER);


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_WRITE_U
SER);
            }

            if (switchPCIsToggled)
            {
                //lb_WriteInfo.Text = "Start writing PC";
```

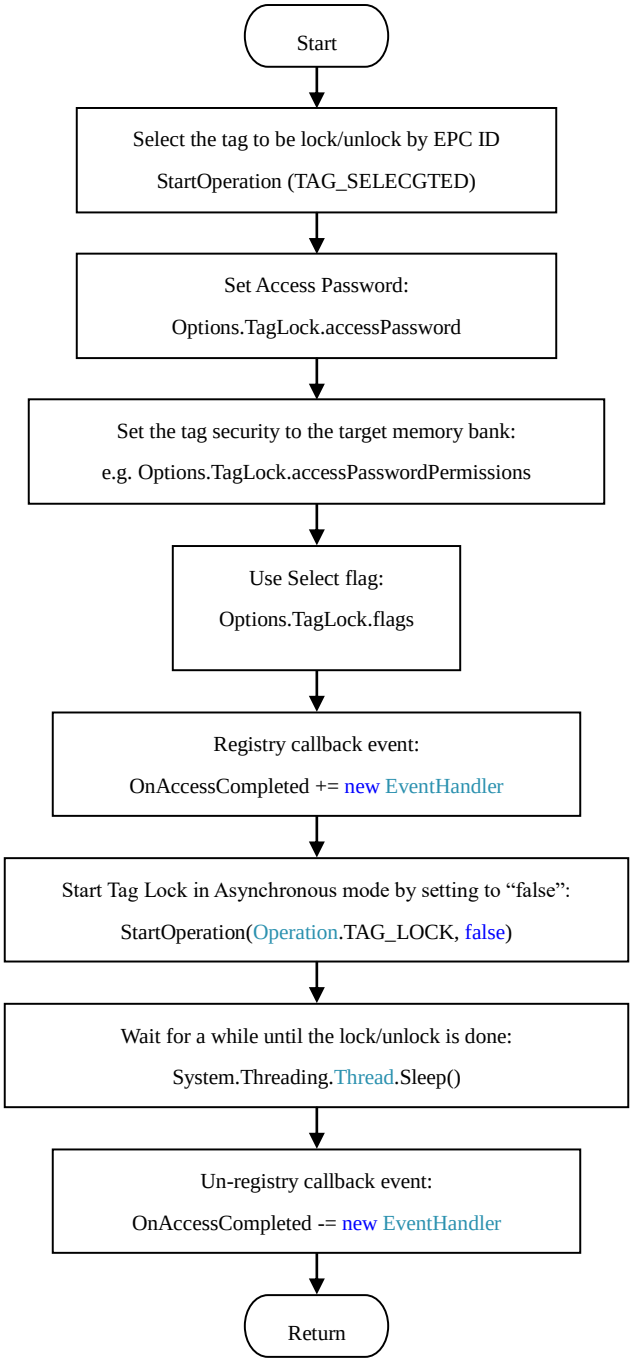
```
                BleMvxApplication._reader.rfid.Options.TagWritePC.retryCount =
m_retry_cnt;
                BleMvxApplication._reader.rfid.Options.TagWritePC.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagWritePC.pc =
CSLibrary.Tools.Hex.ToUshort(entryPC);


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_WRITE_P
C);
            }

            //Write EPC must put in last order to prevent it get lost
            if (switchEPCIsToggled)
            {
                //lb_WriteInfo.Text = "Start writing EPC";

                BleMvxApplication._reader.rfid.Options.TagWriteEPC.retryCount = 0;
                BleMvxApplication._reader.rfid.Options.TagWriteEPC.accessPassword =
accessPwd;
                BleMvxApplication._reader.rfid.Options.TagWriteEPC.offset = 0;
                BleMvxApplication._reader.rfid.Options.TagWriteEPC.count =
CSLibrary.Tools.Hex.GetWordCount(entryEPC);
                BleMvxApplication._reader.rfid.Options.TagWriteEPC.epc = new
CSLibrary.Structures.S_EPC(entryEPC);


BleMvxApplication._reader.rfid.StartOperation(CSLibrary.Constants.Operation.TAG_WRITE_E
PC);
            }
        }
```

8.9 Lock/Unlock a tag

8.9.1 Flow-chart

There is method for locking/unlocking a tag's memory on CS108.

Asynchronous Mode: This method allows locking/un-locking a tag's memory with data returned in asynchronous mode.



8.9.2 Sample Code

This example shows how to lock and unlock a tag's memory data.

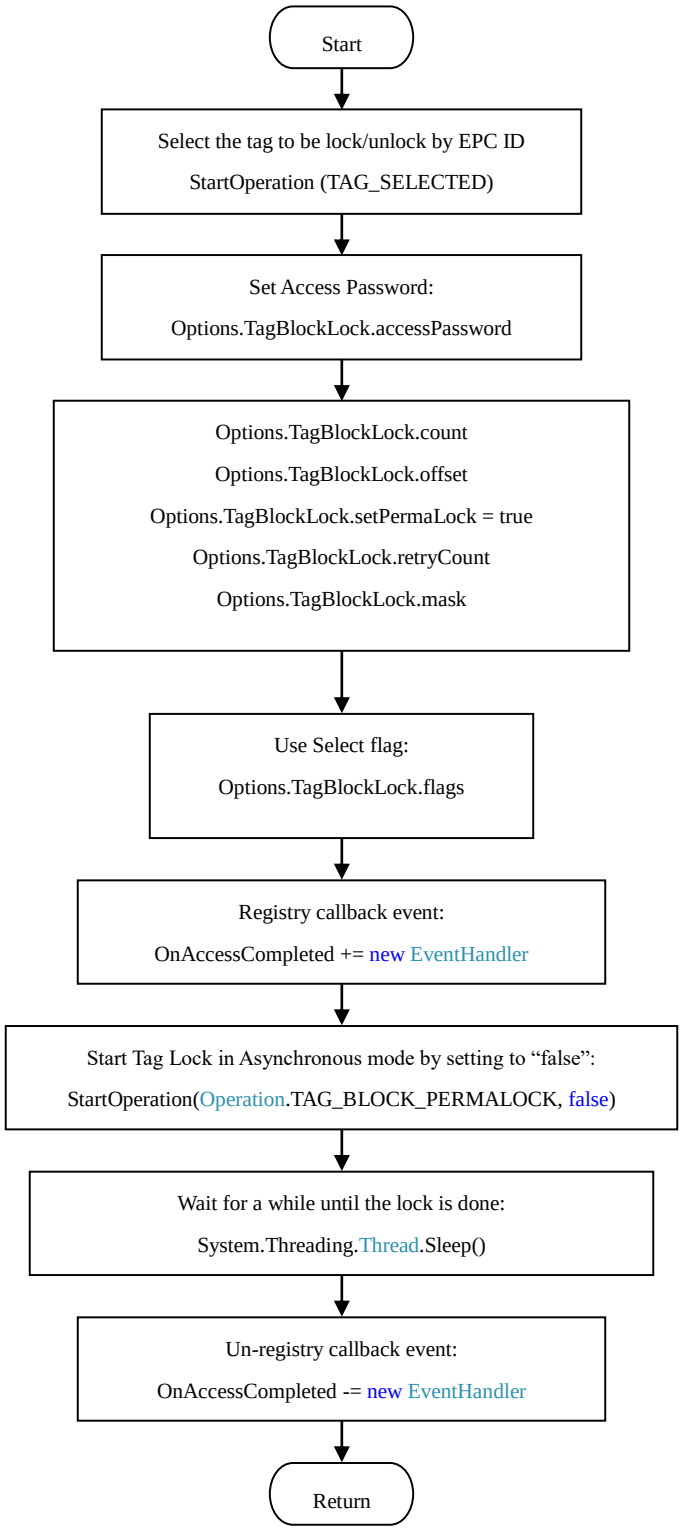
- RunSyncUnlockAccPwd: Unlock the Access Password field of a tag using "RFID.StartOperation(Operation.TAG_LOCK)" command in synchronous mode

8.10 Block PermaLock a tag

8.10.1 Flow-chart

Below is the method for Block PermaLock a tag's User Memory.

Asynchronous Mode: This method allows block permalock a tag's user memory with data returned in asynchronous mode.



8.10.2 Sample Code

This example shows how to block perma-lock a tag's memory data.

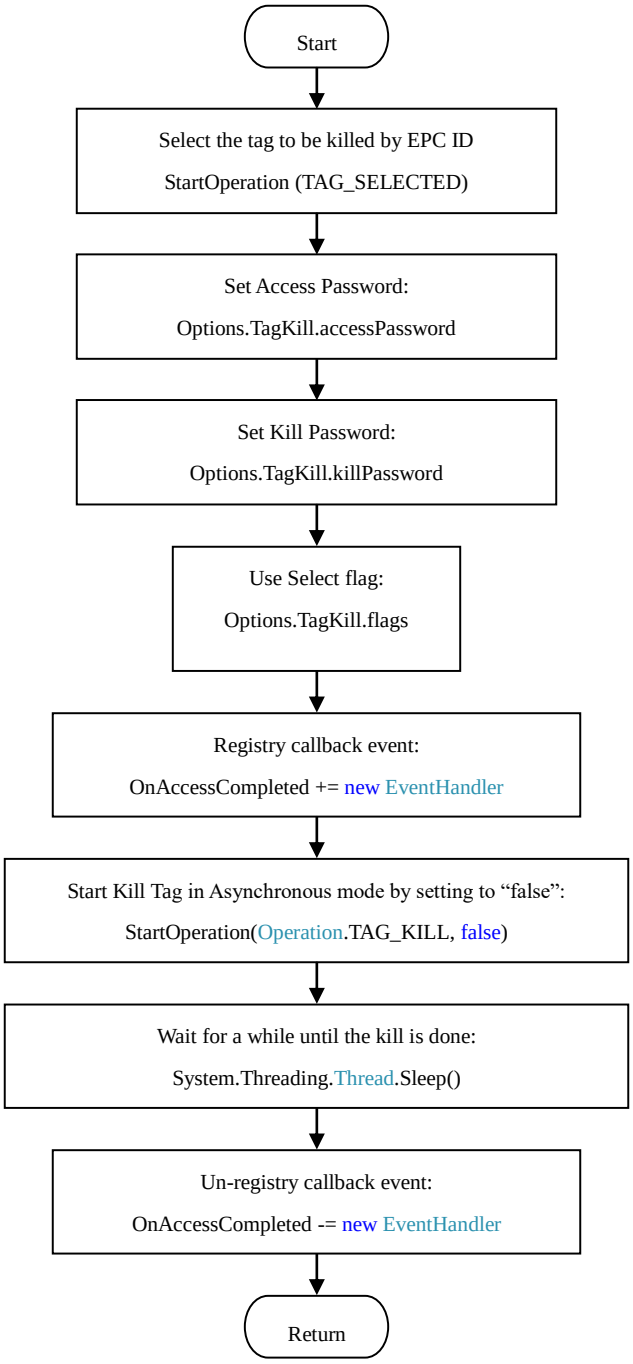
- Lock a User memory block (64 bits) of a tag using "RFID.StartOperation(Operation.TAG_BLOCK_PERMALOCK)" command

8.11 Kill a tag

8.11.1 Flow-chart

There is methods for killing a tag on CS108.

Asynchronous Mode: This method allows killing a tag with data returned in asynchronous mode.



8.11.2 Sample Code

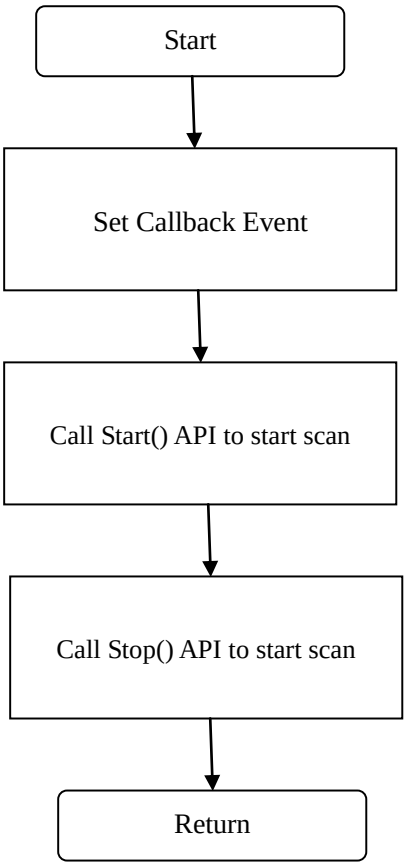
This example shows how to kill a tag.

- RunSyncKillAccPwd: Kill a tag using "RFID.StartOperation(Operation.TAG_KILL)" command in synchronous mode with access password "22222222" and kill password "11111111".

8.12 Scan barcode

8.12.1 Flow-chart

There is method for scanning barcode.



8.12.2 Sample Code

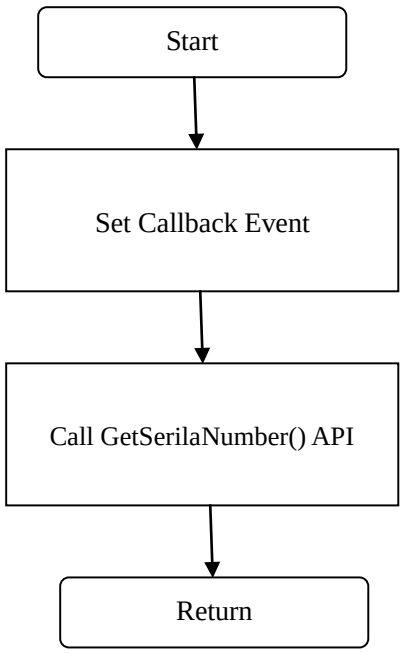
This example shows how to read barcode

```
        public override void main()
        {
                BleMvxApplication._reader.barcode.OnCapturedNotify += new
EventHandler<CSLibrary.Barcode.BarcodeEventArgs>(Linkage_CaptureCompleted);
        }


        void Linkage_CaptureCompleted(object sender,
CSLibrary.Barcode.BarcodeEventArgs e)
        {
            InvokeOnMainThread(() =>
            {
                switch (e.MessageType)
                {
                    case CSLibrary.Barcode.Constants.MessageType.DEC_MSG:

AddOrUpdateBarcodeData((CSLibrary.Barcode.Structures.DecodeMessage)e.Message);

Xamarin.Forms.DependencyService.Get<ISystemSound>().SystemSound(1);
                        //UpdateUI((DecodeMessage)e.Message,           "Barcode
Captured...");
                        break;
                    case CSLibrary.Barcode.Constants.MessageType.ERR_MSG:
                        //UpdateUI(null, String.Format("Barcode Returned: {0}",
e.ErrorMessage));
                        break;
                }
            });
        }

        public async void btnStartScanBarcode(object sender, EventArgs e)
        {
             BleMvxApplication._reader.barcode.Start();
        }
```


8.13 Get device serial number

8.13.1 Flow-chart

There is method to get device serial number.



8.13.2 Sample Code

This example shows how to get device serial number.

```
        public override void main()
        {
            BleMvxApplication._reader.siliconlabIC.OnAccessCompleted += new
EventHandler<CSLibrary.SiliconLabIC.Events.OnAccessCompletedEventArgs>(OnAccess
CompletedEvent);
        }


        void OnAccessCompletedEvent(object sender,
CSLibrary.SiliconLabIC.Events.OnAccessCompletedEventArgs e)
        {
```



```
            {
                switch (e.type)
                {
                    case
CSLibrary.SiliconLabIC.Constants.AccessCompletedCallbackType.SERIALNUMBER:
                        _userDialogs.Alert("Serial Number : " + (string)e.info);
                        break;
                }
            }
        }

        public async void btnGetSerialNumber(object sender, EventArgs e)
        {
            BleMvxApplication._reader.siliconlabIC.GetSerialNumber();
        }
```

9 Building and Deploying DemoApp on Visual Studio 2017

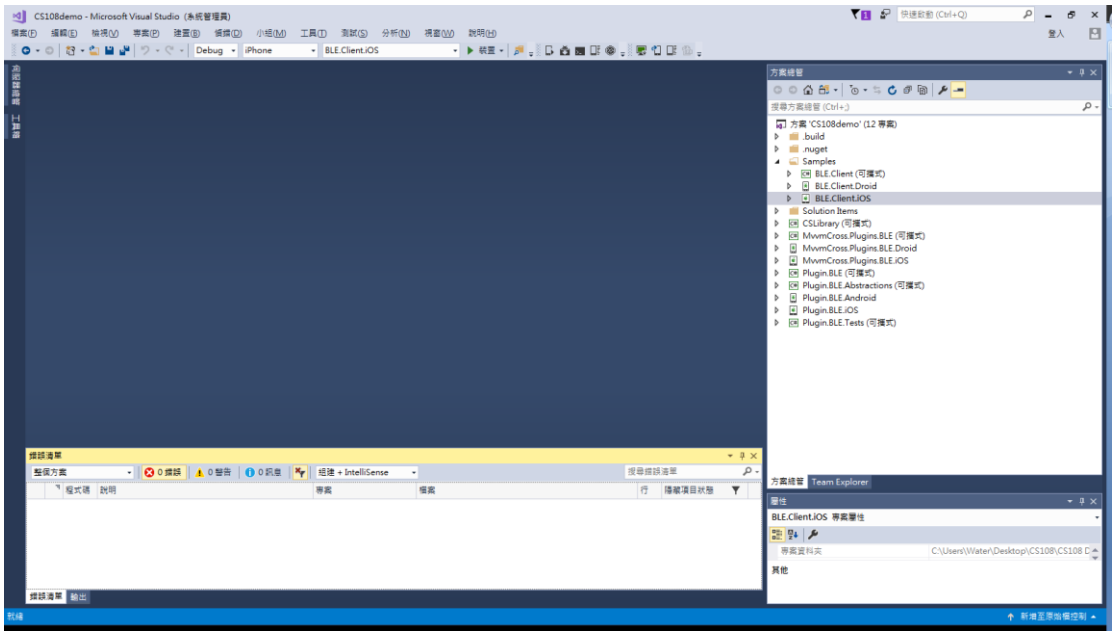
9.1 Callback-based API DemoApp

The file structure of the Callback-based API Demo Application Program includes the following content:

Folder/Files	Content
CS108	Main Folder
CS108\CSLibrary	CS108 library files
CS108 Demo	Source code of the CS108 sample program

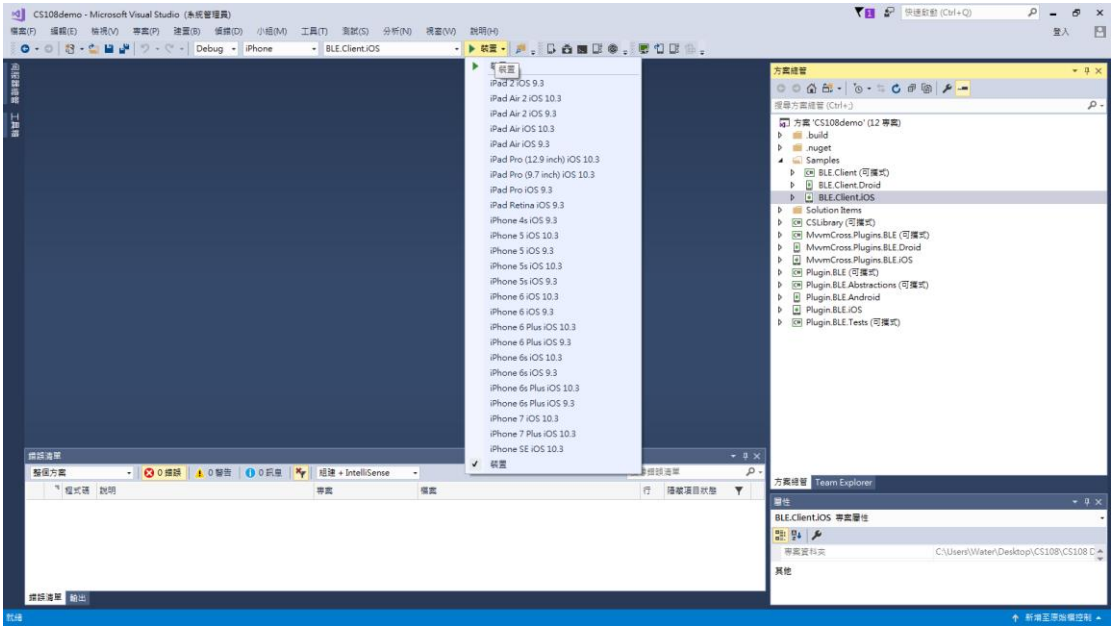
9.2 Building the DemoApp program

1. Open the DemoApp project file (CS108demo.sln) on the development platform by Visual Studio 2017.

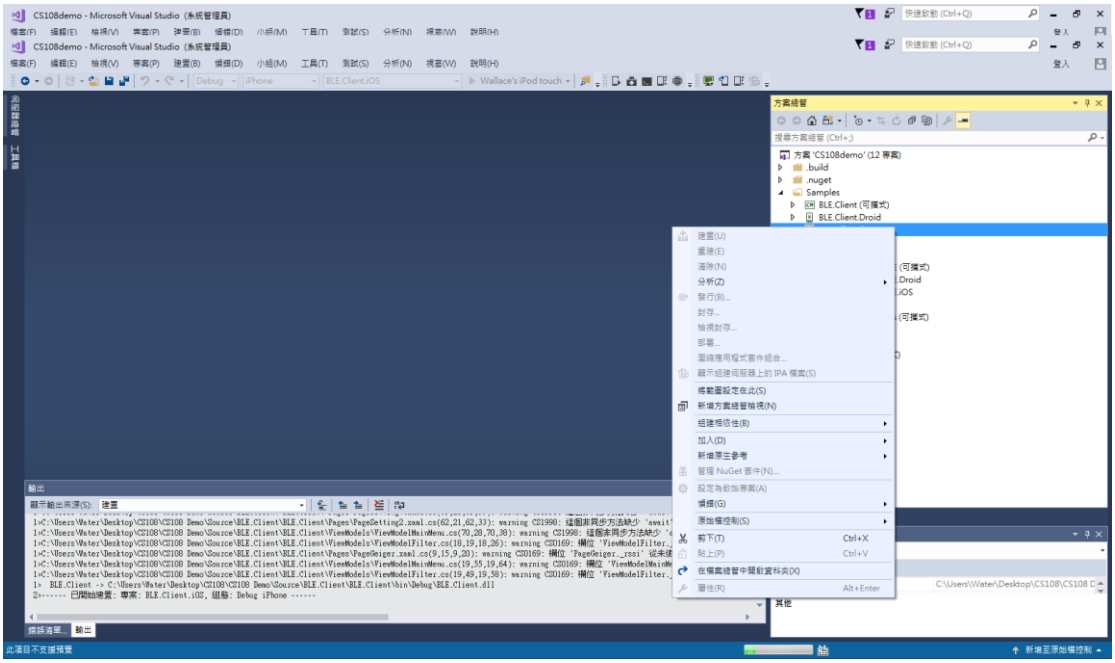


2. Click "Device" on the taskbar of Visual Studio 2017 to select your testing machine

.



3. Select "BLE.Client.iOS" → "Rebuild Solution" to rebuild the project on Visual Studio 2017. Wait until the rebuild success.



4. After the build success, the run debug mode.

10 Appendix A: Link Profiles

There are 4 link profiles in CS108: 0, 1, 2, 3. Only 1 profile is active at any time in CS108. The purpose of each link profile is explained below. These purposes correspond to different business and physical scenarios. The user should try out each profile to see which one gives best performance.

Link Profile	0	1	2	3
Purpose	Best Multipath Fading Resistance	Longest Read Range, Dense Reader Mode	Read Range and Throughput, Dense Reader Mode	Maximum Throughput
R-T Modulation	DSB-ASK	PR-ASK	PR-ASK	DSB-ASK
Tari (μs)	25.00	25.00	25.00	6.25
X	1.00	0.50	0.50	0.50
PW (Pulse Width in usec)	12.50	12.50	12.50	3.13
RTcal (usec)	75.00	62.50	62.50	15.63
TRcal (usec)	200.00	85.33	71.11	20.00
DR (Divide Ratio)	8	64/3	64/3	8
T-R Modulation	FM0	Miller-4	Miller-4	FM0
TRExt	1	1	1	1
Link Frequency(LF) (KHz)	40	250	300	400
Data Rate (Kbps)	40	62.5	75	400

11 Appendix B: Sessions

Session is a concept of EPC to allow a tag to respond to multiple readers inventorying it at the same time, each using a different session number.

There are 4 possible sessions: S0, S1, S2, S3.

The user however has to be careful because these 4 sessions have different behavior, notably how the tag flag "persist" in time. A tag, before inventory or when just after power on, has a flag of State A. When it is inventoried, the flag will go to State B. The tag flag will stay in State B until the tag powers off or the persistence time is up.

A reader can declare it only wants to inventory flag A, so that after a tag is inventoried and its flag gone to State B, it will no longer respond to further inventory rounds – until the end of the persistence time.

Now for S0, S1, S2 and S3, the persistence times are DIFFERENT! Because of that, one has to be very careful in choosing which session to use.

Session	**Tag Flags Persistence Time**
S0	Tag Energized: indefinite Tag Not Energized: none
S1	Tag Energized: 0.5 second < Persistence Time < 5 seconds Tag Not Energized: 0.5 second < Persistence Time < 5 seconds
S2	Tag Energized: indefinite Tag Not Energized: 2 seconds < Persistence Time
S3	Tag Energized: indefinite Tag Not Energized: 2 seconds < Persistence Time

12 Appendix C: Tag Population and Q

Tag Population is the RFID tag population that is to be inventoried. To be more precise, it is the population of tags that can be "seen" by the RFID reader.

Q is an EPC concept related to the way a group of tags is inventoried. When a reader broadcast its desire to inventory tags, it sends out a Q value. The tag will, based on that Q, calculate a certain number and define that as the number of repeated inventories the reader will do. Basically, the relationship of Inventory Repeats and Q is:

Inventory Repeats = 2^Q

The tag will then choose by random a certain number less than this Inventory Repeats. When the reader starts doing inventory, the tag will then respond at that repeat number.

In other words, the Inventory Repeats should correspond to Tag Population:

Tag Population = Inventory Repeats = 2^Q

For example, if there are 8 tags, then in theory the Q can be 3, and if each tag chooses a number different from that of the other 7 (miraculously, of course), then the 8 tags will be inventoried in an orderly manner in turn.

Of course this will never happen, as the tags will easily choose a number the same as that of another one, and a collision will happen.

Therefore, it is a normal practice to have a bigger Q, such as 4 in this case, so that the 8 tags would have a lower chance of choosing the same number.

Therefore, reversing the equation, ideally, we can have:

$Q = INTEGER(LOG_2(Tag\ Population))$

But in reality, we need some headroom, so that:

$Q = INTEGER(LOG_2(Tag\ Population \times 2)+1)$

13 Appendix D: Query Algorithm

There are 2 types of Query Algorithm: Fixed Q and Dynamic Q.

For Fixed Q, the Q value does not change. In other words, the expected Tag Population does not change.

For Dynamic Q, the Q value changes adaptively: when there are a lot of inventory repeats where no tags respond, the reader will interpret that there are not that many RFID tags in the front, and hence it is more efficient to change the Q to a smaller value. When there are a lot of inventory repeats where the reader receive data but they do not satisfy checksum, meaning there is heavy collision, then the reader will interpret that there are too many RFID tags in the front of the reader, and hence it is better to increase the value of Q. Dynamic Q algorithm is a way to allow the RFID reader to adapt to different amount of RFID tags being seen by the reader. The idea is that if there are not so many tags, then the Q can be reduced and the reader can collect all the tag data faster.

14 Appendix E: Target

Target here actually refers to the target flag that the reader wants to inventory. There are 2 possible flags of an RFID tag: State A and State B.

When an RFID tag is first powered up, it has a flag of State A. After it is inventoried, the state of the flag becomes State B.

The tag will only go back to State A if either it is powered off and powered on again, or if its persistence time has run up (See Appendix B).

For each round of inventory, the reader sends out notification to the world which tag flag state it wants to inventory. It can keep on inventory State A, or it can inventory State A and State B alternatively from one round of inventory to the next round of inventory.

In theory, it is a good thing to inventory only State A. The reason being that those tags that have been inventoried should not respond again, and will hence quickly reduce the amount of collision between tags. So in general if you set inventory to State A only, the inventory of large amount of tags can be very fast.

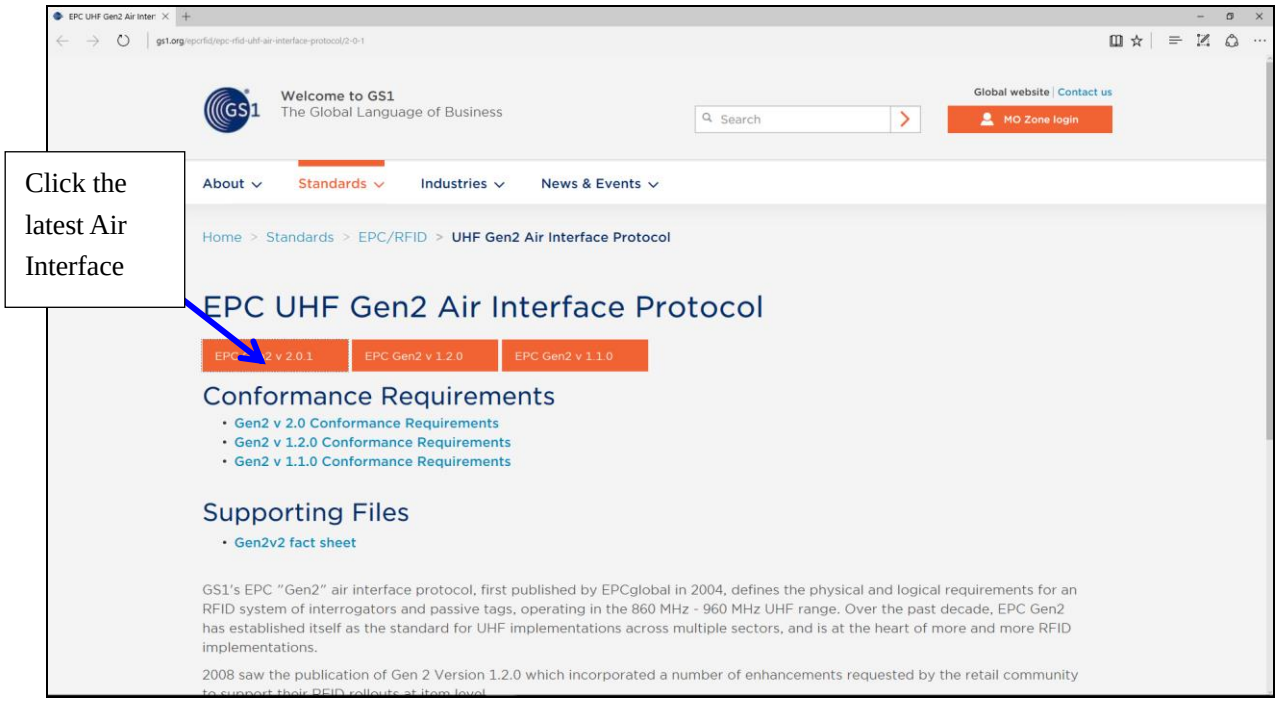
The only catch is that when a tag responds to the reader, it does not know another tag is colliding with it. It sends out the response and thinks it has done the job, hence transitioning to flag State B. So in such case, the tag will not respond to further inventory, even though its response has been lost due to collision. Because of that, sometimes the user will set the inventory to target State A in one inventory round, and then State B in the next round, and vice versa, and so on. This is called A/B Toggle or A & B Dual Target or simply Dual Target.

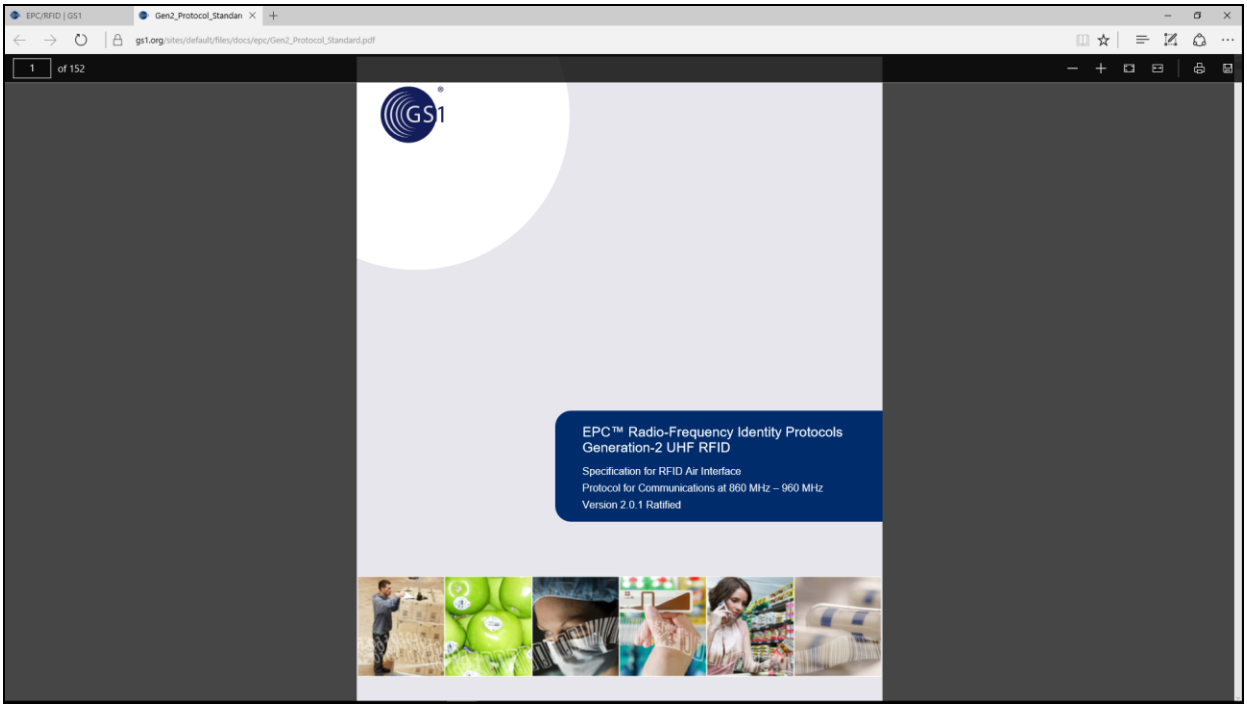
15 Appendix F: Security

There are 4 actions you can apply on the memory inside an RFID tag:

1) Lock
2) Unlock
3) Permanent Lock
4) Permanent Unlock

EPC document which can be found from EPC website:
https://www.gs1.org/epcrfid/epc-rfid-uhf-air-interface-protocol/2-0-1.
Once there, press the button showing the latest air interface and mouse click to get the pdf file.





For Access Password and Kill Password the security locking affects both reading and writing.

For EPC bank and User bank, the security locking affects only writing.

For TID bank, since we are the user and not the manufacturing vendor, security action has no effect. It has been permanently unlocked in the factory anyway.

16 Appendix G: Models & Regulatory Region

There are various models, denoted by the alphanumeric right after the "CS108-", here denoted by "**N**". The applicable regulatory regions for each model are described to the right of it below:

N=1:	865-868 MHz for Europe ETSI, Russia, Mid-East countries, 865-867 MHz for India
N=2:	902-928 MHz, FCC, for USA, Canada and Mexico. Hopping frequencies locked
N=2 AS:	920-926 MHz, Australia. Hopping frequencies locked
N=2 NZ:	921.5-928 MHz, New Zealand. Hopping frequencies locked
N=2 OFCA:	920-925 MHz, Hong Kong. Hopping frequencies locked
N=2 RW:	920-928 MHz, Rest of the World, e.g. Philippines, Brazil, Peru, Uruguay, etc.
N=4:	922-928 MHz, Taiwan
N=7:	920-925 MHz, China
N=8:	916.7-920.9 MHz, Japan
N=9:	915-921 MHz, Europe Upper Band

17 Appendix H: Technical Support

All technical support should be sent to the following email:

info@convergence.com.hk

