CSL CS203 Integrated Reader Callback-based API Programmer's Manual

Version 0.91

CSL: The One-Stop-Shop for RFID Solutions

Table of Contents

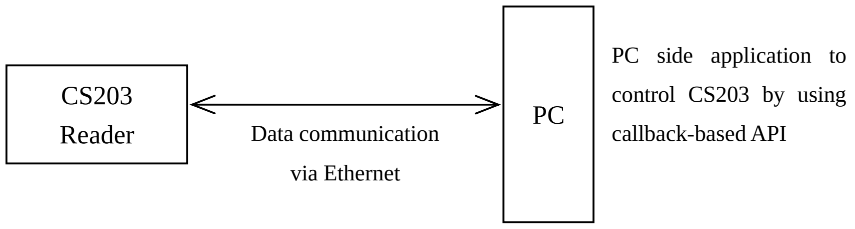
1 Introduction

This Programmer’s Guide provides a comprehensive guideline for the CS203 Callback-based API programming and software development. This document provides detailed information about the programming interface made available by the CS203 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 Dot Net C# programming language.

2 Pre-requisite

The build environment consists of tools and the corresponding configurations of the Visual Studio. It is expected that the system integrator or the software system programming house will be developing the applications on Visual Studio. With this tool, typically he has to write programs on the PC. The following are needed to set up the build environment. Most of the software is also included in the folders "Software Development Environment on PC" on the Document CD.

2.1 System Diagram


CS203 Reader
Data communication via Ethernet
PC
PC side application to control CS203 by using callback-based API

2.2 Basic configuration on PC side:

Operating System requirement:
- Microsoft Windows XP SP2 (English)

Software package required:
- Microsoft Visual Studio 2005 Professional Edition with SP1
- Microsoft .Net Framework 3.5 or above (in Document CD or Free download from Microsoft)
- Visual C++ 2005 Redistributable package (in Document CD or Free download from Microsoft)

Optional software package:
- Microsoft SQL Server 2005 Express Edition (Free download from Microsoft)
- Microsoft SQL Server Management Studio Express Edition (Free download from Microsoft)

To build demo application successfully, you need to install Microsoft Visual Studio 2005 (with Visual C# component and SP1 patch) or above. For more detailed information, please go to Microsoft webpage (http://msdn.microsoft.com/en-us/vstudio/default.aspx).
Visual Studio 2005 SP1 -
http://www.microsoft.com/downloads/details.aspx?familyid=bb4a75ab-e2d4-4c96-b39d-37baf6b5b1dc&displaylang=en

3 Callback-based API Library

3.1 API Library Files

The CS203 Callback-based API Library consists of the following files.

File	**Location of source code**	**Remarks**
RFID_XP.dll	Inside folder "*Intel Binary V1300 for CS203*" of the CALLBACK API Demo Code	CSL Callback-based API Class Library
Interop.NetFwTypeLib.dll	Inside folder "*Intel Binary V1300 for CS203*" of the CALLBACK API Demo Code	Windows firewall library for plug-&-play function
rfid.dll	Inside folder "*Intel Binary V1300 for CS203*" of the CALLBACK API Demo Code	Library for RFID module
rfidtx.dll	Inside folder "*Intel Binary V1300 for CS203*" of the CALLBACK API Demo Code	Library for RFID module
cpl.dll	Inside folder "*Intel Binary V1300 for CS203*" of the CALLBACK API Demo Code	Library for RFID module

3.2 Version History

- The information provided in this document is for Callback-based API 1.0.

4 Useful Document

The below document are useful reference for CS203 Callback-based API programming. They can be found on the Document CD.

- CSL CS203 Integrated Reader User's Manual
- CS203 Callback-based API DemoApp User Guide
- CS203 Callback-based API Library CHM Help Document (RFID Class Library.chm)

5 CS203 API: Theory of Operation

The CS203 Application Programming Interface (API) provides a programming interface for controlling CS203 integrated reader. The interface is loaded by a host 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 application uses the CS203 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 CS203 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 CS203 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.

6 Callback-based API Classes, Methods and Events

6.1 Overview

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

1. MyRunningStateEvent – 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. MyTagAccessEvent – report tag access data to user.

   This operation is configurable through SetTagGroup and SetFixedQParms , SetDynamicQParms, SetDynamicAdjustParms, SetDynamicThresholdParms.

4. MyErrorEvent – reoprt the error message result to user.

6.2 Classes, Methods and Events in Callback-based API

For detail description of all Classes, Methods, Events, Fields and Parameters, please refer to the CS203 Callback-based API CSLibrary CHM Help Document (RFID Class Library.chm).

6.2.1 Reader Namespace

6.2.1.1 Classes Overview

Classes

	Class	Description
	HighLevelInterface	Reader HighLevelInterface

6.2.1.2 HighLevelInterface Class

Methods for Class HighLevelInterface

	Name	Description
	CreateDynamicQAdjustParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	Create DynamicQ Adjust parameter
	CreateDynamicQParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	Create DynamicQ parameter
	CreateDynamicQThresholdParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	Create DynamicQ Threshold parameter
	CreateFixedQParms(UInt32, UInt32, UInt32, UInt32)	Create FixedQ parameter
	CreatePostMatchCriteria(UInt32, UInt32, Boolean, Byte[])	Create PostMatchCriteria parms
	CreateSelectCriteria(UInt32, UInt32, MemoryBank, Action, Target, Boolean, Byte[])	Create SelectCriteria parms
	CreateTagGroup(Selected, Session, SessionTarget)	Create TagGroup parameter
	Dispose()	Destructor
	GetAllReaderSetting(Boolean, String)	Get all reader setting to file or string buffer
	GetAntennaPortConfiguration(AntennaPort Config)	Allows an application to retrieve a single logical antenna port's configuration parameters e.g., dwell time, power level, and number of inventory cycles. Even if the logical antenna port is disabled, an application is allowed to retrieve these

		configuration parameters. Retrieving configuration parameters does not cause a logical antenna port to be automatically enabled; the application must still enable the logical antenna port via RFID_AntennaPortSetState. The antenna-port configuration cannot be retrieved while a radio module is executing a tag- protocol operation.
	GetAntennaPortStatus(AntennaPortStatus)	Retrieves the status of the requested logical antenna port for a particular radio module. The antenna-port status cannot be retrieved while a radio module is executing a tag-protocol operation.
	GetBootLoaderVersion()	Get Silicon Lab Bootloader Version
	GetC51AppVersion()	Get Silicon Lab Application Version Notes:Not support in future version
	GetC51BootLoaderVersion()	Get Silicon Lab Bootloader Version Notes:Not support in future version
	GetCountryCode(UInt32)	GetCountryCode
	GetCurrentLinkProfile(UInt32)	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(SingulationAlgorithm)	Get Current Singulation Algorithm
	GetCurrentTemperature(TemperatureParms)	Get Current System Temperature Don't get temperature during operation starting
	GetDriverVersion()	Get rfid Driver Version
	GetDynamicQAdjustParms(DynamicQAdjustParms)	Get DynamicQ Adjust Singulation Algorithm
	GetDynamicQParms(DynamicQParms)	Get DynamicQ Singulation Algorithm
	GetDynamicQThresholdParms(DynamicQThresholdParms)	Get DynamicQ Threshold Singulation Algorithm
	GetFirmwareVersion()	GetFirmwareVersion
	GetFixedQParms(FixedQParms)	Get FixedQ Singulation Algorithm
	GetFrequencyBand(Int32, UInt32,	Get Frequency band - Basic function

	FrequencyBandParms)	
	GetFrequencyTable(Region)	Get current frequency table
	GetGPI0(Int32)	Get GPI0 status
	GetGPI1(Int32)	Get GPI1 status
	GetGPO0(Int32)	Get GPO0 status
	GetGPO1(Int32)	Get GPO1 status
	GetImageVersion()	Get Silicon Lab Application Version
	GetLED(Int32)	Get LED status
	GetLibraryVersion()	Get Library Version
	GetMacErrorCode(UInt32)	Get Mac Error Code
	GetMacRegion(MacRegion)	Retrieves the region of operation for which the radio module is configured. The region of operation may not be retrieved while a radio module is executing a tag-protocol operation.
	GetOperationMode(RadioOperationMode)	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(UInt32)	GetPowerLevel
	GetPowerState(RadioPowerState)	Allows the application to retrieve the current power state of the radio module. The radio power state cannot be retrieved while a radio module is executing a tag-protocol operation.
	GetRadioConfigurationParameter(UInt16, UInt32)	Allows the application to retrieve the value for an RFID radio module's low-level configuration parameter (i.e., MAC virtual register). Radio configuration parameters may not be retrieved while a radio module is executing a tag-protocol operation.
	GetRadioResponseDataMode(ResponseMode)	Allows the application to retrieve the mode of data reporting for tag-access operations. The data-reporting mode may not be retrieved while a radio module is executing a tag-protocol operation.
	GetReaderDataFormat(ResponseMode)	Allows the application to retrieve the mode of data reporting

		for tag-access operations. The data-reporting mode may not be retrieved while a radio module is executing a tag-protocol operation.
	GetSingulationAlgorithmParms(SingulationAlgorithm, SingulationAlgorithmParms)	GetSingulationAlgorithmParms
	GetTagGroup(TagGroup)	Get Tag Group
	GetThresholdTemperature(ThresholdTemperatureParms)	GetThresholdTemperature
	MacBypassReadRegister(UInt16, UInt16)	Reads directly from a radio-module hardware register. The radio module's hardware registers may not be read while a radio module is executing a tag-protocol operation.
	MacBypassWriteRegister(UInt16, UInt16)	Writes directly to a radio-module hardware register. The radio module's hardware registers may not be written while a radio module is executing a tag-protocol operation.
	MacClearError()	Attempts to clear the error state for the radio module's MAC firmware. The MAC's error state may not be cleared while a radio module is executing a tag-protocol operation.
	MacErrorIsFatal(UInt32)	Mac Error Is Fatal Error
	MacErrorIsNegligible(UInt32)	Mac Error Is Negligible
	MacErrorIsOverheat(UInt32)	Check Mac error code
	MacGetRegion(MacRegion)	Retrieves the region of operation for which the radio module is configured. The region of operation may not be retrieved while a radio module is executing a tag-protocol operation.
	MacReadOemData(UInt32, UInt32, UInt32[])	Reads one or more 32-bit values from the MAC's OEM configuration data area. Note that the the 32-bit values read from the OEM configuration data area are in the R1000 Firmware- processor endian format and it is the responsibility of the application to convert to the endian format of the host processor. The MAC's OEM configuration data area may not be read while a radio module is executing a tag-protocol operation.
	MacReset()	Causes the MAC to perform the specified reset. Note that any currently-executing operations are aborted and unconsumed data is discarded. The MAC resets itself to a well-known

		initialization state and the radio is placed in an idle state. If the radio module is executing a tag-protocol operation, the tag-protocol operation is aborted and the tag-protocol operation function returns with an error code of RFID_ERROR_OPERATION_CANCELLED. Upon successful reset of the MAC, the connection to the radio module is lost and the RFID Reader Library invalidates the radio _radioIndex so that it may not longer be used by the application. The application must re-enumerate the radio modules, via RFID_RetrieveAttachedRadiosList, in the system and request control of the radio module again via RFID_RadioOpen.
	MacSetRegion(MacRegion)	Configures the radio module's region of operation as specified. The region of operation may not be changed while a radio module is executing a tag-protocol operation.
	MacUpdateFirmware(UInt32, NonVolatileMemoryBlock[])	Writes the specified data to the radio module's nonvolatile-memory block(s). After a successful update, the RFID radio module resets itself and the RFID Reader Library closes and invalidates the radio _radioIndex so that it may no longer be used by the application. To obtain control of the radio again, the application must re-enumerate, via RFID_RetrieveAttachedRadiosList, the radio modules in the system and request control of the radio again via RFID_RadioOpen. In the case of an unsuccessful update and depending upon the underlying cause for the returned failure status, the radio module's nonvolatile memory may be left in an undefined state, which means that the radio module may be in an unusable state. In this situation, the RFID Reader Library does not invalidate the radio _radioIndex – i.e., it is the application's responsibility to close the _radioIndex. Alternatively, an application can perform the update in "test" mode. An application uses the "test" mode, by checking the returned status, to verify that the update would succeed before performing the destructive update of the radio module's nonvolatile memory. When a "test" update has completed, either successfully or unsuccessfully, the MAC firmware returns to its normal idle state and the radio _radioIndex remains valid (indicating that the application is still responsible for closing it). The radio module's nonvolatile memory may not be updated while a radio module is executing a tag-protocol operation.

	MacWriteOemData(UInt32, UInt32, UInt32[])	Writes one or more 32-bit values to the MAC's OEM configuration data area. Note that it is the responsibility of the application programmer to ensure that the 32-bit values written to the OEM configuration data area areconverted from the host-processor endian format to the MAC-processor endian format before they are written. The MAC's OEM configuration data area may not be written while a radio module is executing a tag-protocol operation.
	PostMatchTag(UInt32, UInt32, Boolean, Byte[])	Select a tag by PostMatchCriteria
	RadioCloseAndReopen()	Reset Radio
	ResetReader(Operation)	Reset reader
	SelectTag(UInt32, UInt32, MemoryBank, Action, Target, Boolean, Byte[])	Select a tag by SelectCriteria
	SetAntennaPortConfiguration(AntennaPortConfig)	Allows an application to configure several parameters for a single logical antenna port e.g., dwell time, power level, and number of inventory cycles. Even if the logical antenna port is disabled, an application is allowed to set these configuration parameters. Setting configuration parameters does not cause a logical antenna port to be automatically enabled; the application must still enable the logical antenna port via RFID_AntennaPortSetState. The antenna-port configuration cannot be set while a radio module is executing a tag-protocol operation. NOTE: Since RFID_AntennaPortSetConfiguration sets all of the configuration parameters that are present in the RFID_ANTENNA_PORT_CONFIG structure, if an application wishes to leave some parameters unchanged, the application should first call RFID_AntennaPortGetConfiguration to retrieve the current settings, update the values in the structure that are to be changed, and then call RFID_AntennaPortSetConfiguration.
	SetCurrentLinkProfile(UInt32)	Allows the application to set the current link profile for the radio module. A link profile will remain in effect until changed by a subsequent call to RFID_RadioSetCurrentLinkProfile. The current link profile cannot be set while a radio module is executing a tag-protocol operation.
	SetCurrentLinkProfile(UInt32, Boolean)	Allows the application to set the current link profile for the radio module. A link profile will remain in effect until changed by a subsequent call to RFID_RadioSetCurrentLinkProfile. The

		current link profile cannot be set while a radio module is executing a tag-protocol operation.
	SetCurrentSingulationAlgorithm(Singulation Algorithm)	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.
	SetDynamicQAdjustParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	The parameters for the dynamic-Q algorithm that uses the ISO 18000-6C Query Adjust command, MAC singulation algorithm 2
	SetDynamicQAdjustParms()	Set System Default DynamicQAdjust
	SetDynamicQParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	The parameters for the dynamic-Q algorithm, MAC singulation algorithm 1
	SetDynamicQParms()	Set System Default DynamicQ
	SetDynamicQThresholdParms(UInt32, UInt32, UInt32, UInt32, UInt32, UInt32)	The parameters for the dynamic-Q algorithm with application-controlled Q-adjustment-threshold, MAC singulation algorithm 3
	SetFixedChannel(Region, UInt32, LBT)	Set Fixed Frequency Channel Only ETSI, IDA , G800 or JPN can be used to set a fixed channel
	SetFixedChannel(UInt32, LBT)	Set Fixed Frequency Channel on current select region
	SetFixedQParms(UInt32, UInt32, UInt32, UInt32)	The parameters for the fixed-Q algorithm, MAC singulation algorithm 0 If running a same operation, it only need to config once times
	SetFixedQParms()	Set System Default FixedQ
	SetFrequencyBand(Int32, UInt32, FreqCfgRegistor, UInt32, UInt32)	Set Frequency Band - Basic Function
	SetGPI0(Int32)	Set GPI0 status
	SetGPI1(Int32)	Set GPI1 status
	SetHoppingChannels(Region)	Set to the specific frequency profile
	SetHoppingChannels()	Reset current frequency profile

	SetLED(Int32)	Set Led status
	SetMultiPostMatchCrit(List<ReaderPostMatchCrit>)	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., RFID_18K6CTagInventory, 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 RFID_FLAG_PERFORM_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 RFID_18K6CSetPostMatchCriteria. Post-singulation match criteria may not be changed while a radio module is executing a tag-protocol operation.
	SetMultiSelectCriteria(List<ReaderSelectCriteria>)	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., RFID_18K6CTagInventory, 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 RFID_FLAG_PERFORM_SELECT flag is provided to the appropriate RFID_18K6CTag* function). The tag-selection criteria will stay in effect until the next call to RFID_18K6CSetSelectCriteria. Tag-selection criteria may not be changed while a radio module is executing a tag-protocol operation.
	SetPowerLevel(UInt32)	Set Power Level(Max 300)...
	SetPowerState(RadioPowerState)	Allows the application to set the power state of the radio module. The radio power state cannot be set while a radio module is executing a tag-protocol operation.
	SetRadioConfigurationParameter(UInt16, UInt32)	Allows an application to set an RFID radio module's low-level configuration parameter (i.e., MAC virtual register). Radio configuration parameters may not be set while a radio module is executing a tag-protocol operation.
	SetRadioResponseDataMode(ResponseMode)	Allows the application to control the mode of data reporting for tag-access operations. By default, when an application opens a radio, the RFID Reader Library sets the reporting mode to "Compact". The data-reporting mode will remain in effect until

		a subsequent call to RFID_RadioSetResponseDataMode, or the radio is closed and re-opened (at which point the data mode is set to normal). The data-reporting mode may not be changed while a radio module is executing a tag-protocol operation.
	SetReaderDataFormat(ResponseMode)	Allows the application to control the mode of data reporting for tag-access operations. By default, when an application opens a radio, the RFID Reader Library sets the reporting mode to "normal". The data-reporting mode will remain in effect until a subsequent call to RFID_RadioSetResponseDataMode, or the radio is closed and re-opened (at which point the data mode is set to normal). The data-reporting mode may not be changed while a radio module is executing a tag-protocol operation.
	SetRegulatoryAntennaPortState(AntennaPortState)	Allows an application to specify whether or not a radio module's logical antenna port is enabled for subsequent tag operations. The antenna-port state cannot be set while a radio module is executing a tag-protocol operation.
	SetSingulationAlgorithmParms(SingulationAlgorithm, SingulationAlgorithmParms)	SetSingulationAlgorithmParms
	SetTagGroup(Selected, Session, SessionTarget)	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.
	SetThresholdDynamicQParms()	Set System Default ThresholdDynamicQ
	SetThresholdTemperature(UInt32, UInt32, UInt32, UInt32)	No Use, use system default value instead SetThresholdTemperature
	Setup(Operation)	Basic operation setup configuration. eg, Inventory, Read, Write, Kill and Lock
	ShutdownReader()	Deinit the Radio Device
	Sleep(UInt32)	Sleep milliseconds by using WaitForSingleObject
	Start()	Start Operation with threading
	StartupReader(String, UInt32)	Startup Reader and configure TCP Port This Port number must match to your reader port number you set before
	Stop()	Stop Operation

	Write(ReaderFlag, MemoryBank, UInt16, UInt16, UInt32, UInt16[], UInt32, UInt32, UInt32)	Write data to tag by using main thread

Fields

	Name	Description
	AUS_CHN_CNT	Australia Frequency Channel number
	BR1_CHN_CNT	Brazil1 Frequency Channel number
	BR2_CHN_CNT	Brazil2 Frequency Channel number
	CN_CHN_CNT	China Frequency Channel number
	ETSI_CHN_CNT	ETSI Frequency Channel number
	FCC_CHN_CNT	FCC Frequency Channel number
	HK_CHN_CNT	Hong Kong Frequency Channel number
	IDA_CHN_CNT	India Frequency Channel number
	JPN_CHN_CNT	Japan Frequency Channel number
	KR_CHN_CNT	Korea Frequency Channel number
	MYS_CHN_CNT	Malaysia Frequency Channel number
	TW_CHN_CNT	Taiwan Frequency Channel number
	ZA_CHN_CNT	FCC Frequency Channel number

Properties

	Name	Description
	AvailableLinkProfile	Available Link Profile you can use
	AvailableMaxPower	Available Maximum Power you can set
	AvailableRegion	Available frequency profiles you can use
	Instance	HighLevelInterface Instance, Thread Safe
	IsFixedChannel	Get Fixed frequency channel
	IsFixedChannelOnly	If true, it can only set to fixed channel. Otherwise, both fixed and hopping can be set.
	LastResultCode	get last function return code
	LBT_ON	Get Current LBT status
	MacErrOccur	Get Mac Error happen in Inventory
	MissingStartupCall	Startup the Reader?
	MyState	Current Operation State
	NewInstance	HighLevelInterface Instance
	RadioIndex	Current Operation Radio Index
	RdrOpParms	Reader Operation Parmemter
	RunCycle	-1 run forever, use this if you set RunOnce to true.
	SelectedFreqChannel	Get Current Selected Frequency Channel
	SelectedFreqProfile	Get Frequency Channel Profile
	SelectedFreqProfileIndex	Return the index of the current selected frequency profile Fail return -1
	SelectedFrequency	Current selected frequency

	SelectedLinkProfileIndex	Get Selected LinkProfile Index Fail return (0xff)
	TotalNumberOfRadios	Get the total attached radio, for windoes CE version, it at least return 1;

Events

	Name	Description
	MyErrorEvent	Raise Error Event if any Error Ocurrs
	MyInventoryEvent	Inventory Event Callback
	MyRunningStateEvent	Reader Operation State Event
	MyTagAccessEvent	Tag Access Event Callback

6.2.2 Reader.Constants Namespace

6.2.2.1 Enumerations Overview

Enumerations

	Name	Description
	BandState	Frequency Band State
	BarcodeState	Barcode Operation State
	EPCBitNumber	EPC Length in Bit
	ErrorCode	Operation Error Code
	ErrorType	Error Tyoe
	FreqCfgRegistor	Frequency Channel configuration register 0, Disable this frequency channel 1, Enalbe this Channel 2, Disable channel and Guard Bands Checked 3, Enable channel and Guard Bands Checked
	LBT	LBT Config
	Operation	RFID Operation Mode
	PktErrorCode	Tag Access Backscatter Error Code
	ReaderFlag	RFID Flags
	ReaderOperationMode	Reader Operation State
	Region	Frequency Profile
	TagAccResult	Tag Access Result: Sometime tag partial write happen, it also return OK

6.2.3 Reader.DebugF Namespace

6.2.3.1 Classes Overview

Classes

	Class	Description
	DebugF	Debug File

6.2.3.2 DebugF Class

Methods for Constants

	Name	Description
	Dispose()	Destructor
	Write(String)	Write Msg to file
	Write(String, Result)	
	Write(String, String)	Write Msg to file

6.2.4 Reader.Events Namespace

6.2.4.1 Classes Overview

Classes

	Class	Description
	ErrorEventArg	CRCErrorEventArgs
	EventExtensions	EventExtensions1
	InventoryEventArgs	InventoryEventArgs
	ReaderOperationModeEventArgs	ReaderOperationModeEventArgs
	TagAccessEventArgs	TagAccessEventArgs

6.2.4.2 ErrorEventArg Class

Properties

	Name	Description
	ErrorCode	Error Code
	ErrorType	ErrorType

6.2.4.3 EventExtensionArgs Class

Constructors

	Name	Description
	EventExtensions()	Initializes a new instance of the EventExtensions class

6.2.4.4 InventoryEventArgs Class

Properties

	Name	Description
	InventoryInformation	Inventory Information

6.2.4.5 ReaderOperationModeEventArgs Class

Properties

	Name	Description

	State	State

6.2.4.6 TagAccessEventArgs Class

Properties

	Name	Description
	TagAccessInformation	Tag Access Information

6.2.5 Reader.Net Namespace

6.2.5.1 Classes Overview

Classes

	Class	Description
	DeviceFinderArgs	Device Finder Argument
	NETDISPLAY_ENTRY	Netfinder information return from device
	NetFinder	Search device on ethernet
	ResultArgs	Result Argument
	UpdatePercentArgs	
	UpdateResultArgs	

Structures

	Name	Description
	TimeEvent	Time Event

Enumerations

	Name	Description
	Mode	Netfinder Mode
	RecvOperation	Current Packet Recevice mode
	Resul	Result
	UpdateResul	Update result

6.2.5.2 DeviceFinderArgs Class

Properties

	Name	Description
	Found	Device finder information

6.2.5.3 NETDISPLAY_ENTRY Class

Fields

	Name	Description

	NETDISPLAY_ENTRY()	Initializes a new instance of the NETDISPLAY_ENTRY class
	description	Mode discription
	device_name	Device name, user can change it.
	DHCP	enable or disable DHCP
	ip	IP address
	mac	MAC address
	mode	Reserved for future use
	port	UDP Port
	serverport	Reserved for future use, Server mode port
	tcptimeout	Tcp timeout
	time_on_network	Total time on network
	time_on_powered	Total time on power on device

6.2.5.4 NetFinder Class

Methods for NetFinder Class

	Name	Description
	AssignDevice(Byte[], Byte[])	Change IP Address
	AssignDevice(Byte[],Byte[], Byte)	Change IP Address and TCP timeout
	AssignDevice(Byte[],Byte[], String, Byte)	Change IP Address and TCP timeout and device name
	AssignDevice(Byte[],Byte[], String, Byte, Boolean)	Change IP Address and TCP timeout and device name
	AsyncUpdateEboot(String, String)	Async update Eboot
	AsyncUpdateImage(String, String)	Async update image
	ClearDeviceList()	Clear all device list
	Dispose()	Dispose resource
	ResearchDevice()	Start to re-search device on ethernet continuously until Stop function called.
	SearchDevice()	Start to search device on ethernet continuously until Stop function called.
	Stop()	Stop to search

Properties

	Name	Description
	LastError	
	Operation	Get current recevice mode operation

Events

	Name	Description
	OnAssignCompleted	Assign device callback event
	OnSearchCompleted	Search device callback event
	OnUpdateCompleted	Update eboot/image callback event
	OnUpdatePercent	Update total update percent

6.2.5.5 ResultArgs Class

Properties

	Name	Description
	Result	Result

6.2.5.6 UpdatePercentArgs Class

	Name	Description
	Percent	Total update Percent

6.2.5.7 UpdateResultArgs Class

	Name	Description
	UpdateResultArgs(UpdateResult)	Constructor
	Result	Update Result

6.2.6 Reader.Settings Namespace

6.2.6.1 Classes Overview

Classes

	Class	Description
	Settings	Reader Setting Parameters

6.2.6.2 Settings Class

Fields

	Name	Description
	AntennaPortConfig	AntennaPortConfig
	AntennaPortStatus	AntennaPortStatus
	CurrentLinkProfile	CurrentLinkProfile
	DynamicQAdjustParms	DynamicQAdjustParms
	DynamicQParms	DynamicQParms
	DynamicQThresholdParms	DynamicQThresholdParms
	FixedQParms	FixedQParms
	PinsConfiguration	PinsConfiguration
	QueryParms	QueryParms

	RadioLinkProfile	RadioLinkProfile
	RadioOperationMode	RadioOperationMode
	RadioPowerState	RadioPowerState
	ResponseMode	ResponseMode
	SelectCriteria	SelectCriteria
	SingulationAlgorithm	SingulationAlgorithm
	SingulationCriteria	SingulationCriteria
	TagGroup	TagGroup

6.2.7 Reader.Structures Namespace

6.2.7.1 Classes Overview

Classes

	Class	**Description**
	InventoryDataStruct	Inventory Data Callback Structures
	ReaderConfigParms	Reader Configuration Parms
	ReaderOperateParms	Operation Paramemter

Structures

	Name	**Description**
	Client	Client IP Address eg. RFID Reader IP address
	CRC	CRC
	EPC	Electronic Product Code
	ErrorDataStruct	Error Data Structure
	FrequencyBandParms	Frequency Band Parms
	PC	Protocol Control
	ReaderPostMatchCrit	Single Criteria for Post Match Criteria
	ReaderSelectCriteria	Single Criteria for Select Criteria
	RunBasicKillParms	The ISO 18000-6C tag-kill operation parameters
	RunBasicLockParms	The ISO 18000-6C tag-lock operation parameters
	RunBasicReadParms	The ISO 18000-6C tag-read operation parameters
	RunBasicSearchParms	The ISO 18000-6C tag-inventory operation parameters
	RunBasicWriteParms	The ISO 18000-6C tag-write operation parameters
	RunLockAccParms	
	RunLockEPCParms	
	RunLockKillParms	
	RunLockTIDParms	
	RunLockUSERParms	
	RunReadUserParms	The ISO 18000-6C tag-Write Bank0 operation parameters

	RunSearchAnyParms	
	RunSearchOneTagParms	
	RunWriteBank0Parms	The ISO 18000-6C tag-Write Bank0 operation parameters
	RunWriteEPCParms	The ISO 18000-6C tag-Write EPC operation parameters
	RunWritePCParms	The ISO 18000-6C tag-Write PC operation parameters
	RunWriteUserParms	The ISO 18000-6C tag-Write USER operation parameters
	Server	Server Address eg. PC IP address and Port number
	TagAccessDataStruct	Tag Access Data Callback Structures
	TemperatureParms	Temperature Parms
	ThresholdTemperatureParms	ThresholdTemperatureParms

6.2.7.2 InventoryDatStruct Class

Fields

	Name	**Description**
	Count	Receive Count
	CRC	Cyclic Redundancy Check 4 Bytes
	EPC	EPC Data
	EPCByteLength	EPC Bytes Length
	Index	Index Number FCFS
	ms_ctr	Millisecond Counter
	PacketTime	Packet come back time
	PC	PC Data 4 Bytes
	RSSI	Receive Signal Strength Indication

6.2.7.3 ReaderConfigParms Class

Fields

	Name	**Description**
	AntennaPortConfig	AntennaPortConfig
	AntennaPortStatus	AntennaPortStatus
	CurrentLinkProfile	CurrentLinkProfile
	DynamicQAdjustParms	DynamicQAdjustParms
	DynamicQParms	DynamicQParms
	DynamicQThresholdParms	DynamicQThresholdParms
	FixedQParms	FixedQParms
	PinsConfiguration	PinsConfiguration
	QueryParms	QueryParms
	RadioLinkProfile	RadioLinkProfile
	RadioOperationMode	RadioOperationMode
	RadioPowerState	RadioPowerState
	ResponseMode	ResponseMode

	SelectCriteria	SelectCriteria
	SingulationAlgorithm	SingulationAlgorithm
	SingulationAlgorithmParms	SingulationAlgorithmParms
	SingulationCriteria	SingulationCriteria
	TagGroup	TagGroup

6.2.7.4 ReaderOperateParms Class

Fields

	Name	Description
	Operation	Current Operation mode
	ReadRetryCount	Retry count for read operation
	RunBasicKillParms	Config this before kill
	RunBasicLockParms	Config this before lock
	RunBasicReadParms	Config this before read
	RunBasicSearchParms	Config this before search
	RunBasicWriteParms	Config this before write
	RunLockAccParms	Config this before lock access password
	RunLockEPCParms	Config this before lock EPC
	RunLockKillParms	Config this before lock kill password
	RunLockTIDParms	Config this before lock TID
	RunLockUSERParms	Config this before lock USER
	RunReadUserParms	Config this before read USER
	RunSearchAnyParms	Config this before search any tag
	RunSearchOneTagParms	Config this before search one tag
	RunWriteAccPwdParms	Config this before write access password
	RunWriteEPCParms	Config this before write EPC
	RunWriteKillPwdParms	Config this before write kill password
	RunWritePCParms	Config this before write PC
	RunWriteUserParms	Config this before write USER
	TargetAccessPasswor	Target Access Password
	TargetEPC	Use EPC to select a target Tag
	TargetMemBank	Target Memory Bank
	TargetPC	User PC to select a target tag
	WriteRetryCount	Retry count for write operation

6.2.8 Reader.Utility Namespace

6.2.8.1 Classes Overview

Classes

	Class	Description
	Counter	High Performance Counter
	GUID	Autogen a guid
	HexEncoding	Summary description for HexEncoding.
	Sound	PC Sound
	Version	Get Version

6.2.8.2 Counter Class

Methods for Counter Class

	Name	Description
	GetTickCount()	
	Start()	Start Timer
	Stop()	Stop Timer

Properties

	Name	Description
	Duration	Returns the duration of the timer (in seconds)

6.2.8.3 GUID Class

Methods for GUID Class

	Name	Description
	Gen12BitUID()	12 bit guid
	Gen24BitUID()	24 bit guid
	Gen48BitUID()	48 bit guid
	Gen96BitUID()	96 bit guid

6.2.8.4 HexEncoding Class

Methods for HexEncoding Class

	Name	Description
	Compare(UInt16[], Byte[])	Compare ushort to byte
	Compare(Byte[], Byte[])	Compare two array
	CompareHex(Byte[],Byte[])	Compare Hex
	CompareHex(Byte[],Byte[], Int32)	Compare Hex
	CompareHexString(String, String)	Compare Hex String, ie, source = "11ffaa", target = "11FFaa", it will return true
	Copy(Byte[], Int32, Int32)	Copy One array to another array
	GetByte(String)	return a byte from string
	GetByteCount(String)	GetByteCount
	GetBytes(String)	Creates a byte array from the hexadecimal string. Each two characters are

		combined to create one byte. First two hexadecimal characters become first byte in returned array. Non-hexadecimal characters are ignored.
	GetBytes(UInt16 [])	Convent ushort array to byte array
	GetUshort(Byte[])	return ushort array from byte array
	GetUshort(String)	return ushort array from string input
	IsHexDigit(Char)	Returns true if c is a hexadecimal digit (A-F, a-f, 0-9)
	IsHexFormat(String)	Determines if given string is in proper hexadecimal string format
	Parse_Int16(Byte[], Int32)	Convert 2 Bytes to one short
	Parse_UInt16(Byte[], Int32)	Convert 2 Bytes to one short
	Parse_UInt32(Byte[], Int32)	Convert 4 Bytes to one uint32
	Parse_UInt64(Byte[], Int32)	Convert 4 bytes to long
	ToString(Byte[])	Byte to String Conversion
	ToString(UInt16[])	ushort to String Conversion
	ToString(Byte[], Int32, Int32)	Byte to String Conversion

6.2.8.5 Sound Class

Methods for Sound Class

	Name	Description
	Beep(UInt32, UInt32)	Sound Beep

6.2.8.6 Version Class

Methods for Version Class

	Name	Description
	GetVersion()	Get RFID CSharp Library Version

6.2.9 rfidt Namespace

6.2.9.1 Classes Overview

Classes

	Class	Description
	Linkage	Native Class Library Linkage
	Native	Native class

Delegates

	Name	Description
	CallbackDelegate	Declare, instantiate and utilize for callbacks originating from inventory, read, write and similar operations...

6.2.9.2 Linkage Class

Methods for Linkage Class

	Name	Description
	AntennaPortGetConfiguration(Int32, UInt32, AntennaPortConfig)	Retrieves the configuration for a radio module's antenna port. The antenna port configuration may not be retrieved while a radio module is executing a tag-protocol operation.
	AntennaPortGetStatus(Int32, UInt32, AntennaPortStatus)	Retrieves the status of a radio module's antenna port. The antenna port status may not be retrieved while a radio module is executing a tag- protocol operation.
	AntennaPortSetConfiguration(Int32, UInt32, AntennaPortConfig)	Sets the configuration for a radio module's antenna port. The antenna port configuration may not be set while a radio module is executing a tag-protocol operation.
	AntennaPortSetState(Int32, UInt32, AntennaPortState)	Sets the state of a radio module's antenna port. The antenna port state may not be set while a radio module is executing a tag-protocol operation.
	Get18K6CCurrentSingulationAlgorithm(Int32, SingulationAlgorithm)	Allows the application to retrieve the currently-active singulation algorithm. The currently-active singulation algorithm may not be changed while a radio module is executing a tag-protocol operation.
	Get18K6CPostMatchCriteria(Int32, SingulationCriteria)	Retrieves the configured post-singulation match criteria to be used by the RFID radio module. The post-singulation match criteria is used for any tag-protocol operations (i.e., RFID_18K6CTagInventory, 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. Post- singulation match criteria may not be retrieved while a radio module is executing a tag-protocol operation. The post-singulation match criteria may not be retrieved while a radio module is executing a tag-protocol operation.
	Get18K6CQueryTagGroup(Int32, TagGroup)	Retrieves the tag group that will have subsequent tag-protocol operations (e.g., inventory, tag read, etc.) applied to it. The tag group may not be retrieved while a radio module is executing a tag-protocol operation.
	Get18K6CSelectCriteria(Int32, SelectCriteria)	Retrieves the configured tag-selection criteria for the ISO 18000-6C select command. The returned tag-selection

		criteria are used for any tag-protocol operations (i.e., RFID_18K6CTagInventory, etc.) in which the application specifies that an ISO 18000-6C select command should be issued prior to executing the tag-protocol operation. The select criteria may not be retrieved while a radio module is executing a tag-protocol operation.
	Get18K6CSingulationAlgorithmParameters(Int32, SingulationAlgorithm, SingulationAlgorithmParms)	Allows the application to retrieve the settings for a particular singulation algorithm. Singulation-algorithm parameters may not be retrieved while a radio module is executing a tag-protocol operation.
	GetQueryParameters(Int32, QueryParms)	Retrieves the parameters for the ISO 18000-6C query command. These are the query parameters that used for tag-protocol operations (i.e., RFID_18K6CTagInventory, etc.). Query parameters may not be retrieved while a radio module is executing a tag-protocol operation. The query parameters may not be retrieved while a radio module is executing a tag- protocol operation. NOTE: As of version 1.1 of the RFID Reader Library, this function has been deprecated and replaced by the combination of RFID_18K6CGetQueryTagGroup, RFID_18K6CGetCurrentSingulationAlgorithm, and RFID_18K6CGetSingulationAlgorithmParameters. This function remains for backwards compatibility, however new code should not use it as it will be removed in a future version.
	MacBypassReadRegister(Int32, UInt16, UInt16)	Allows for direct reading of registers on the radio (i.e., bypassing the MAC). The radio regsiters mode may not be read while a radio module is executing a tag-protocol operation.
	MacBypassWriteRegister(Int32, UInt16, UInt16)	Allows for direct writing of registers on the radio (i.e., bypassing the MAC). The radio registers may not be written while a radio module is executing a tag-protocol operation.
	MacClearError(Int32)	Attempts to clear the error state for the radio module 抯 MAC firmware. The MAC error may not be cleared while a radio module is executing a tag- protocol operation.
	MacGetRegion(Int32, MacRegion, IntPtr)	Retrieves the regulatory mode region for the MAC's operation. The region of operation may not be retrieved

		while a radio module is executing a tag- protocol operation.
	MacGetVersion(Int32, MacVersion)	Retrieves the radio module's MAC firmware version information. The MAC version may not be retrieved while a radio module is executing a tag- protocol operation.
	MacReadOemData(Int32, UInt32, UInt32, UInt32[])	Reads one or more 32-bit words from the MAC's OEM configuration data area. The OEM data are may not be read while a radio module is executing a tag-protocol operation.
	MacReset(Int32, MacResetType)	Instructs the radio module's MAC firmware to perform the specified reset. Any currently executing tag-protocol operations will be aborted, any unconsumed data will be discarded, and tag-protocol operation functions (i.e., RFID_18K6CTagInventory, etc.) will return immediately with an error of RFID_ERROR_OPERATION_CANCELLED. Upon reset, the connection to the radio module is lost and the handle to the radio is invalid. To obtain control of the radio module after it has been reset, the application must re-enumerate the radio modules, via RFID_RetrieveAttachedRadiosList, and request control via RFID_RadioOpen. NOTE: This function must not be called from the packet callback function.
	MacSetRegion(Int32, MacRegion, IntPtr)	Sets the regulatory mode region for the MAC's operation. The region of operation may not be set while a radio module is executing a tag-protocol operation.
	MacUpdateNonvolatileMemory(Int32, UInt32, NonVolatileMemoryBlock[], UInt32)	Writes the specified data to the radio module 抯 nonvolatile-memory block(s). After a successful update, the RFID radio module resets itself and the RFID Reader Library closes and invalidates the radio handle so that it may no longer be used by the application. In the case of an unsuccessful update the RFID Reader Library does not invalidate the radio handle ?i.e., it is the application 抯 responsibility to close the handle. Alternatively, an application can perform the update in 搕est?mode. An application uses the 搕est?mode, by checking the returned status, to verify that the update would succeed before performing the destructive update of the radio

		module 抯 nonvolatile memory. When a 搕est?update has completed, either successfully or unsuccessfully, the MAC firmware returns to its normal idle state and the radio handle remains valid (indicating that the application is still responsible for closing it). The radio module 抯 nonvolatile memory may not be updated while a radio module is executing a tag-protocol operation.
	MacWriteOemData(Int32, UInt32, UInt32, UInt32[])	Writes one or more 32-bit words to the MAC's OEM configuration data area. The OEM data area may not be written while a radio module is executing a tag-protocol operation.
	RadioAbortOperation(Int32, UInt32)	Stops a currently-executing tag-protocol operation on a radio module and discards all remaining command-reponse packets. NOTE: This function must not be called from the packet callback function.
	RadioCancelOperation(Int32, UInt32)	Stops a currently-executing tag-protocol operation on a radio module. The packet callback function will be executed until the command-end packet is received from the MAC or the packet callback returns a non-zero result. NOTE: This function must not be called from the packet callback function.
	RadioClose(Int32)	Release control of a previously-opened radio. On close, any currently- executing or outstanding requests are cancelled and the radio is returned to idle state. NOTE: This function must not be called from the packet callback function.
	RadioGetConfigurationParameter(Int32, UInt16, UInt32)	Retrieves a low-level radio module configuration parameter. Radio configuration parameters may not be retrieved while a radio module is executing a tag-protocol operation.
	RadioGetCurrentLinkProfile(Int32, UInt32)	Retrieves the current link profile for the radio module. The current link profile may not be retrieved while a radio module is executing a tag- protocol operation.
	RadioGetGpioPinsConfiguration(Int32, UInt32)	Retrieves the configuration for the radio module's GPIO pins. For version 1.0 of the library, only GPIO pins 0-3 are valid. The GPIO pin configuration may not be retrieved while a radio module is executing a tag- protocol

		operation.
	RadioGetLinkProfile(Int32, UInt32, RadioLinkProfile)	Retrieves the information for the specified link profile for the radio module. A link profile may not be retrieved while a radio module is executing a tag-protocol operation.
	RadioGetOperationMode(Int32, RadioOperationMode)	Retrieves the radio's operation mode. The operation mode may not be retrieved while a radio module is executing a tag-protocol operation.
	RadioGetPowerState(Int32, RadioPowerState)	Retrieves the radio module's power state (not to be confused with the antenna RF power). The power state may not be retrieved while a radio module is executing a tag-protocol operation.
	RadioGetResponseDataMode(Int32, ResponseType, ResponseMode)	Retrieves the operation response data reporting mode for tag-protocol operations. The data response mode may not be retrieved while a radio module is executing a tag-protocol operation.
	RadioOpen(UInt32, Int32, MacMode)	Requests explicit control of a radio.
	RadioReadGpioPins(Int32, UInt32, UInt32)	Reads the specified radio module's GPIO pins. Attempting to read from an output GPIO pin results in an error. For version 1.0 of the library, only GPIO pins 0-3 are valid. The GPIO pins may not be read while a radio module is executing a tag-protocol operation.
	RadioReadLinkProfileRegister(Int32, UInt32, UInt16, UInt16)	Retrieves the contents of a link-profile register for the specified link profile. A link-profile regsiter may not be read while a radio module is executing a tag-protocol operation.
	RadioSetConfigurationParameter(Int32, UInt16, UInt32)	Sets the low-level configuration parameter for the radio module. Radio configuration parameters may not be set while a radio module is executing a tag-protocol operation.
	RadioSetCurrentLinkProfile(Int32, UInt32)	Sets the current link profile for the radio module. The curren link profile may not be set while a radio module is executing a tag-protocol operation.
	RadioSetGpioPinsConfiguration(Int32, UInt32,	Configures the specified radio module's GPIO pins. For

	UInt32)	version 1.0 of the library, only GPIO pins 0-3 are valid. The GPIO pin configuration may not be set while a radio module is executing a tag-protocol operation.
	RadioSetOperationMode(Int32, RadioOperationMode)	Sets the radio's operation mode. An RFID radio module 抯 operation mode will remain in effect until it is explicitly changed via RFID_RadioSetOperationMode. The operation mode may not be set while a radio module is executing a tag-protocol operation.
	RadioSetPowerState(Int32, RadioPowerState)	Sets the radio module's power state (not to be confused with the antenna RF power). The power state may not be set while a radio module is executing a tag-protocol operation.
	RadioSetResponseDataMode(Int32, ResponseType, ResponseMode)	Sets the operation response data reporting mode for tag-protocol operations. By default, the reporting mode is set to "normal". The reporting mode will remain in effect until a subsequent call to RFID_RadioSetResponseDataMode. The mode may not be changed while the radio is executing a tag-protocol operation. The data response mode may not be set while a radio module is executing a tag-protocol operation.
	RadioTurnCarrierWaveOff(Int32)	
	RadioTurnCarrierWaveOn(Int32)	
	RadioWriteGpioPins(Int32, UInt32, UInt32)	Writes the specified radio module's GPIO pins. Attempting to write to an input GPIO pin results in an error. For version 1.0 of the library, only GPIO pins 0-3 are valid. The GPIO pins may not be written while a radio module is executing a tag-protocol operation.
	RadioWriteLinkProfileRegister(Int32, UInt32, UInt16, UInt16)	Writes a valut to a link-profile register for the specified link profile. A link-profile regsiter may not be written while a radio module is executing a tag-protocol operation.
	RetrieveAttachedRadiosList(RadioEnumeration, UInt32)	Retrieves the list of radio modules attached to the system.
	Set18K6CCurrentSingulationAlgorithm(Int32, SingulationAlgorithm)	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.
	Set18K6CPostMatchCriteria(Int32, SingulationCriteria, UInt32)	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., RFID_18K6CTagInventory, 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. The post-singulation match criteria will stay in effect until the next call to RFID_18K6CSetPostMatchCriteria. The post-singulation match criteria may not be set while a radio module is executing a tag-protocol operation.
	Set18K6CQueryTagGroup(Int32, TagGroup)	Specifies which tag group will have subsequent tag-protocol operations (e.g., inventory, tag read, etc.) applied to it. The tag group may not be changed while a radio module is executing a tag-protocol operation.
	Set18K6CSelectCriteria(Int32, SelectCriteria, UInt32)	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., RFID_18K6CTagInventory, etc.) in which the application specifies that an ISO 18000-6C select command should be issued prior to executing the tag-protocol operation. The tag-selection criteria will stay in effect until the next call to RFID_18K6CSetSelectCriteria. The select criteria may not be set while a radio module is executing a tag-protocol operation.
	Set18K6CSingulationAlgorithmParameters(Int32, SingulationAlgorithm, SingulationAlgorithmParms)	Allows the application to configure the settings for a particular singulation algorithm. A singulation algorithm may not be configured while a radio module is executing a tag-protocol operation. NOTE: Configuring a singulation algorithm does not automatically set it as the current singulation algorithm (see RFID_18K6CSetCurrentSingulationAlgorithm).
	SetQueryParameters(Int32, QueryParms, Int32)	Configures the parameters for the ISO 18000-6C query command. The supplied query parameters will be used for any subsequent tag-protocol operations (i.e., RFID_18K6CTagInventory, etc.) and will stay in effect until the next call to RFID_18K6CSetQueryParameters. The query parameters may not be set while a radio

		module is executing a tag-protocol operation. NOTE: Failure to call RFID_18K6CSetQueryParameters prior to executing the first tag-protocol operation (i.e., RFID_18K6CTagInventory, etc.) will result in the RFID radio module using default values for the ISO 18000-6C query parameters. NOTE: As of version 1.1 of the RFID Reader Library, this function has been deprecated and replaced by the combination of RFID_18K6CSetQueryTagGroup, RFID_18K6CSetCurrentSingulationAlgorithm, and RFID_18K6CSetSingulationAlgorithmParameters. This function remains for backwards compatibility, however new code should not use it as it will be removed in a future version.
	Shutdown()	Shuts down RFID Reader Library, cleaning up all resources including closing all open radio handles and returning radios to idle. NOTE: This function must not be called from the packet callback function.
	Sleep(UInt32)	Sleep milliseconds
	Startup(LibraryVersion, LibraryMode)	Initializes the RFID Reader Library
	Tag18K6CInventory(Int32, InventoryParms, UInt32)	Executes a tag inventory for the tags of interest. If the RFID_FLAG_PERFORM_SELECT flag is specified, the tag population is partitioned (i.e., ISO 18000-6C select) prior to the inventory operation. If the RFID_FLAG_PERFORM_POST_MATCH flag is specified, the post-singulation match mask is applied to a singulated tag's EPC to determine if the tag will be returned to the application. The operation-response packets will be returned to the application via the application-supplied callback function. An application may prematurely stop an inventory operation by calling RFID_Radio{Cancel|Abort}Operation on another thread or by returning a non-zero value from the callback function. A tag inventory may not be issued while a radio module is executing a tag-protocol operation.
	Tag18K6CKill(Int32, KillParms, UInt32)	Executes a tag kill for the tags of interest. If the RFID_FLAG_PERFORM_SELECT flag is specified, the tag population is partitioned (i.e., ISO 18000-6C select) prior to the tag-kill operation. If the

		RFID_FLAG_PERFORM_POST_MATCH flag is specified, the post-singulation match mask is applied to a singulated tag's EPC to determine if the tag will be killed. The operation-response packets will be returned to the application via the application-supplied callback function. Each tag-kill record is grouped with its corresponding tag-inventory record. An application may prematurely stop a kill operation by calling RFID_Radio{Cancel|Abort}Operation on another thread or by returning a non- zerovalue from the callback function. A tag kill may not be issued while a radio module is executing a tag-protocol operation.
	Tag18K6CLock(Int32, LockParms, UInt32)	Executes a tag lock for the tags of interest. If the RFID_FLAG_PERFORM_SELECT flag is specified, the tag population is partitioned (i.e., ISO 18000-6C select) prior to the tag-lock operation. If the RFID_FLAG_PERFORM_POST_MATCH flag is specified, the post-singulation match mask is applied to a singulated tag's EPC to determine if the tag will be locked. The operation-response packets will be returned to the application via the application-supplied callback function. Each tag-lock record is grouped with its corresponding tag-inventory record. An application may prematurely stop a lock operation by calling RFID_Radio{Cancel|Abort}Operation on another thread or by returning a non- zero value from the callback function. A tag lock may not be issued while a radio module is executing a tag-protocol operation.
	Tag18K6CRead(Int32, ReadParms, UInt32)	Executes a tag read for the tags of interest. If the RFID_FLAG_PERFORM_SELECT flag is specified, the tag population is partitioned (i.e., ISO 18000-6C select) prior to the tag-read operation. If the RFID_FLAG_PERFORM_POST_MATCH flag is specified, the post-singulation match mask is applied to a singulated tag's EPC to determine if the tag will be read from. Reads may only be performed on 16-bit word boundaries and for multiples of 16-bit words. If one or more of the memory words specified by the offset/count combination do not exist or are read-locked, the read from the tag will fail and this failure will be reported through the operation response packet. The operation-response packets will be returned to the application via the

		application-supplied callback function. Each tag-read record is grouped with its corresponding tag- inventory record. An application may prematurely stop a read operation by calling RFID_Radio{Cancel|Abort}Operation on another thread or by returning a non-zero value from the callback function. A tag read may not be issued while a radio module is executing a tag-protocol operation. Note that read should not be confused with inventory. A read allows for reading a sequence of one or more 16-bit words starting from an arbitrary 16-bit location in any of the tag's memory banks.
	Tag18K6CWrite(Int32, WriteParms, UInt32)	Executes a tag write for the tags of interest. If the RFID_FLAG_PERFORM_SELECT flag is specified, the tag population is partitioned (i.e., ISO 18000-6C select) prior to the tag-write operation. If the RFID_FLAG_PERFORM_POST_MATCH flag is specified, the post-singulation match mask is applied to a singulated tag's EPC to determine if the tag will be written to. Writes may only be performed on 16-bit word boundaries and for multiples of 16-bit words. If one or more of the specified memory words do not exist or are write-locked, the write to the tag will fail and this failure will be reported through the operation-response packet. The operation-response packets will be returned to the application via the application-supplied callback function. Each tag-write record is grouped with its corresponding tag-inventory record. An application may prematurely stop a write operation by calling RFID_Radio{Cancel|Abort}Operation on another thread or by returning a non- zero value from the callback function. A tag write may not be issued while a radio module is executing a tag-protocol operation.
	TCP_POWER_DEVICE(String, Int32)	Send TCP Command to power up or power down RFID device
	TCPIsConnected()	Check whether TCP Connection is lost Notice : Does not use this during reader is busy
	TCPPowerDownDevice()	Send TCP Command to power down RFID device Notice : Does not use this during reader is busy

	TCPPowerUpDevice()	Send TCP Command to power up RFID device Notice : Does not use this during reader is busy
	TCPReconnect()	Reconnect TCP Notice : Does not use this during reader is busy
	TCPRecv(String, UInt32)	Recevice TCP data Notice : Does not use this during reader is busy
	TCPSend(String)	Send TCP data Notice : Does not use this during reader is busy
	TCPShutdown()	Shutdown TCP Connection Notice : Does not use this during reader is busy
	TCPStartup(String, UInt32)	Startup TCP connection with RFID reader

6.2.9.3 Native Class

Methods for Native Class

	Name	Description
	RFID_18K6CGetCurrentSingulationAlgorithm(Int32, SingulationAlgorithm)	
	RFID_18K6CGetPostMatchCriteria(Int32, IntPtr)	
	RFID_18K6CGetQueryTagGroup(Int32, TagGroup)	
	RFID_18K6CGetSelectCriteria(Int32, IntPtr)	
	RFID_18K6CGetSingulationAlgorithmParameters(Int32, SingulationAlgorithm, IntPtr)	
	RFID_18K6CSetCurrentSingulationAlgorithm(Int32, SingulationAlgorithm)	
	RFID_18K6CSetPostMatchCriteria(Int32, IntPtr, UInt32)	
	RFID_18K6CSetQueryTagGroup(Int32, TagGroup)	
	RFID_18K6CSetSelectCriteria(Int32, IntPtr, UInt32)	
	RFID_18K6CSetSingulationAlgorithmParameters(Int32, SingulationAlgorithm, IntPtr)	
	RFID_18K6CTagInventory(Int32, IntPtr, UInt32)	
	RFID_18K6CTagKill(Int32, IntPtr, UInt32)	
	RFID_18K6CTagLock(Int32, IntPtr, UInt32)	
	RFID_18K6CTagRead(Int32, IntPtr, UInt32)	
	RFID_18K6CTagWrite(Int32, IntPtr, UInt32)	
	RFID_AntennaPortGetConfiguration(Int32, UInt32,	

	AntennaPortConfig)	
	RFID_AntennaPortGetStatus(Int32, UInt32, AntennaPortStatus)	
	RFID_AntennaPortSetConfiguration(Int32, UInt32, AntennaPortConfig)	
	RFID_AntennaPortSetState(Int32, UInt32, AntennaPortState)	
	RFID_BTLOADER(String)	
	RFID_EBOOT(String)	
	RFID_GetBTLVersion(String, Byte[])	
	RFID_GetGPI0(String, Int32)	
	RFID_GetGPI1(String, Int32)	
	RFID_GetGPO0(String, Int32)	
	RFID_GetGPO1(String, Int32)	
	RFID_GetIMGVersion(String, Byte[])	
	RFID_GetLED(String, Int32)	
	RFID_GetMACAddress(String, Byte[])	
	RFID_MacBypassReadRegister(Int32, UInt16, UInt16)	
	RFID_MacBypassWriteRegister(Int32, UInt16, UInt16)	
	RFID_MacClearError(Int32)	
	RFID_MacGetRegion(Int32, MacRegion, IntPtr)	
	RFID_MacGetVersion(Int32, MacVersion)	
	RFID_MacReadOemData(Int32, UInt32, UInt32, IntPtr)	
	RFID_MacReset(Int32, MacResetType)	
	RFID_MacSetRegion(Int32, MacRegion, IntPtr)	
	RFID_MacUpdateNonvolatileMemory(Int32, UInt32, IntPtr, UInt32)	
	RFID_MacWriteOemData(Int32, UInt32, UInt32, IntPtr)	
	RFID_POWER(String, Int32)	
	RFID_PowerDown()	
	RFID_PowerUp()	
	RFID_RadioAbortOperation(Int32, UInt32)	
	RFID_RadioCancelOperation(Int32, UInt32)	
	RFID_RadioClose(Int32)	
	RFID_RadioGetConfigurationParameter(Int32, UInt16, UInt32)	
	RFID_RadioGetCurrentLinkProfile(Int32, UInt32)	
	RFID_RadioGetGpioPinsConfiguration(Int32, UInt32)	
	RFID_RadioGetLinkProfile(Int32, UInt32, IntPtr)	
	RFID_RadioGetOperationMode(Int32, RadioOperationMode)	
	RFID_RadioGetPowerState(Int32, RadioPowerState)	
	RFID_RadioGetResponseDataMode(Int32, ResponseType,	

	ResponseMode)	
	RFID_RadioOpen(UInt32, Int32, MacMode)	
	RFID_RadioReadGpioPins(Int32, UInt32, UInt32)	
	RFID_RadioReadLinkProfileRegister(Int32, UInt32, UInt16, UInt16)	
	RFID_RadioSetConfigurationParameter(Int32, UInt16, UInt32)	
	RFID_RadioSetCurrentLinkProfile(Int32, UInt32)	
	RFID_RadioSetGpioPinsConfiguration(Int32, UInt32, UInt32)	
	RFID_RadioSetOperationMode(Int32, RadioOperationMode)	
	RFID_RadioSetPowerState(Int32, RadioPowerState)	
	RFID_RadioSetResponseDataMode(Int32, ResponseType, ResponseMode)	
	RFID_RadioTurnCarrierWaveOff(Int32)	
	RFID_RadioTurnCarrierWaveOn(Int32)	
	RFID_RadioWriteGpioPins(Int32, UInt32, UInt32)	
	RFID_RadioWriteLinkProfileRegister(Int32, UInt32, UInt16, UInt16)	
	RFID_RetrieveAttachedRadiosList(IntPtr, UInt32)	
	RFID_SetGPO0(String, Int32)	
	RFID_SetGPO1(String, Int32)	
	RFID_SetLED(String, Int32)	
	RFID_Shutdown()	
	RFID_Sleep(UInt32)	
	RFID_Startup(LibraryVersion, LibraryMode)	
	RFID_TCPConnected()	
	RFID_TCPReconnect()	
	RFID_TCPRecv(String, UInt32)	
	RFID_TCPSend(String, UInt32)	
	RFID_TCPSetSocket(IntPtr)	
	RFID_TCPShutdown()	
	RFID_TCPStartup(String, UInt32)	

6.2.10 rfid.Constants Namespace

Enumerations

	Enumerations	Description
	Action	
	AntennaPortState	
	DataDifference	

	DivideRatio	
	GpioPin	
	LibraryMode	
	MacMode	
	MacRegion	
	MacResetType	
	MemoryBank	
	MemoryPermission	
	MillerNumber	
	ModulationType	
	PasswordPermission	
	RadioOperationMode	
	RadioPowerState	
	RadioProtocol	
	ResponseMode	
	ResponseType	
	Result	
	Selected	
	Session	
	SessionTarget	
	SingulationAlgorithm	
	Target	
	WriteType	

6.2.11 rfid.Structures Namespace

6.2.11.1 Classes Overview

Classes

	Class	**Description**
	AntennaPortConfig	
	AntennaPortStatu	
	CommonParms	
	DriverVersion	
	DynamicQAdjustParms	
	DynamicQParms	
	DynamicQThresholdParms	
	FixedQParms	
	InventoryParms	
	KillParms	
	LibraryVersion	
	LockParms	

	MacVersion	
	NonVolatileMemoryBlock	
	ProfileConfig	
	QueryParms	
	RadioEnumeration	
	RadioInformation	
	RadioLinkProfile	
	RadioLinkProfileConfig_ISO18K6C	
	ReadParms	
	SelectAction	
	SelectCriteria	
	SelectCriterion	
	SelectMask	
	SingulationAlgorithmParms	
	SingulationCriteria	
	SingulationCriterion	
	SingulationMask	
	TagGroup	
	TagPerm	
	Version	
	WriteParms	
	WriteParmsBase	
	WriteRandomParms	
	WriteSequentialParms	

6.3 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,
    };
```

7 Programming Flow

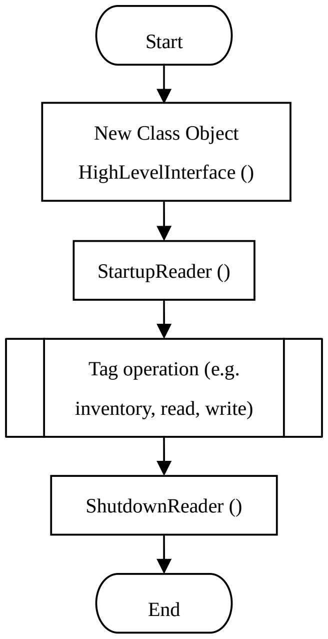
This chapter describes the detail programming flow for normal operations on CS203, including Startup/Shutdown RFID, Setting Frequency/Country/Parameters, Inventory, Read, Write, Lock, Kill, Search, Controlling LED.

7.1 Startup/Shutdown RFID Sequence

This section describes the flow for starting up and shutting down the RFID module.

7.1.1 Flow-chart

The standard RFID startup/shutdown sequence for CS203 is as below.



7.1.2 Sample Code

The following is a piece of sample code that shows how to initiate the CS203 handheld reader and also how to shutdown the reader

---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;

static class example1
{
    using Reader;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    static void StartUp()
    {
        //Startup Reader
        if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
        {
            MessageBox.Show("StartupReader Fail");
            return;
        }

        //Start to access any operation in library
        //...

        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();

    }
}
```

7.2 RFID Country/Frequency Setting

There are two functions to configure country and frequency profile:

- `SetFixedChannel()` : Configure the reader to operate in fixed frequency channel
- `SetHoppingChannels()` : Configure the reader to operate in frequency hopping

To operate in fixed frequency channel (for ETSI, India, G800 and Japan readers):

```
/// <summary>
/// Set Fixed Frequency Channel
/// </summary>
/// <param name="prof">Country Profile</param>
/// <param name="channel">Single Channel</param>
/// <param name="LBTcfg">This is only used when JPN is set</param>
/// <returns>Result</returns>
public Result SetFixedChannel(RegionCode prof, uint channel, LBT LBTcfg)
```

To operate in frequency hopping channels (for countries other than ETSI, India, G800 and Japan readers):

```
/// <summary>
/// Set to the specific frequency profile, all countries except ETSI, IDA , G800 and JPN
/// </summary>
/// <param name="prof">Country Profile</param>
/// <returns>Result</returns>
public Result SetHoppingChannels(RegionCode prof)
```

To get current selected region:

```
Region region = Program.RFID.SelectedFreqProfile;
```

To get current selected frequency and channel:

```
double freq = Program.RFID.SelectedFrequency;
uint channel = Program.RFID.SelectedFreqChannel;
```

To get current support region:

```
List<Region> regionList =  Program.RFID.AvailableRegion;
```

To get current available frequencies:

```
double[] freqTable = Program.RFID.GetFrequencyTable(Region freq);
```

It will return (for CS203-3 Japanese version reader)

```
952.40, <- channel 1, index start from 0
952.60,
952.80,
953.00,
953.20,
953.40,
953.60, <- last channel
```

e.g. If you have CS203-3 Japan region reader, you can use the below function to set current frequency to 953.0 MHz and turn on LBT.

```
SetFixedChannel(Region.JP, 3, LBT.ON);
```

The available Region for CS203 are listed below:

Member name	**Description**
FCC	USA
ETSI	Europe
CN	China
TW	Taiwan
KR	Korea
HK	Hong Kong
JP	Japan
AU	Australia
MY	Malaysia
SG	Singapore
IN	India
G800	G800 same as India
ZA	South Africa
BR1	Brazil
BR2	Brazil
UNKNOWN	Unknown Country

7.3 RFID Parameters Setting

There are several RFID parameters can be configured on CS203:

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

7.3.1 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)
```

7.3.2 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 5.

```
/// <summary>
/// Set Current Link Profile (0 to 5)...
/// </summary>
/// <param name="profile">Link Profile 0,1,2,3,4,5</param>
/// <returns></returns>
public Result SetCurrentLinkProfile(uint profile)
```

7.3.3 Setting Operation Mode

The operation mode of the reader can be configured by function `RadioSetOperationMode(uint handle, 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 RadioSetOperationMode(uint handle, RadioOperationMode mode)
```

7.3.4 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(Selected gpSelect, Session gpSession, SessionTarget
                        gpSessionTarget)
```

7.3.5 Setting Q Parameter

There are 4 main inventory algorithms (one fixed Q and three variable Q) on CS203 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)
```

Dynamic Q Adjustment Algorithm:

```
        /// <summary>
        /// The parameters for the dynamic-Q algorithm that uses the ISO 18000-6C Query Adjust
        /// command, MAC singulation algorithm 2
        /// </summary>
        public Result SetDynamicQAdjustParms(DynamicQAdjustParms dynParm)
```

Dynamic Q Adjustment Threshold Algorithm:

```
        /// <summary>
        /// The parameters for the dynamic-Q algorithm with application-controlled
        /// Q-adjustment-threshold, MAC singulation algorithm 3
        /// </summary>
        /// <returns></returns>
        public Result SetDynamicQThresholdParms(DynamicQThresholdParms dynParm)
```

7.3.6 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)
```

7.3.7 Setting PostMatch Criteria

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

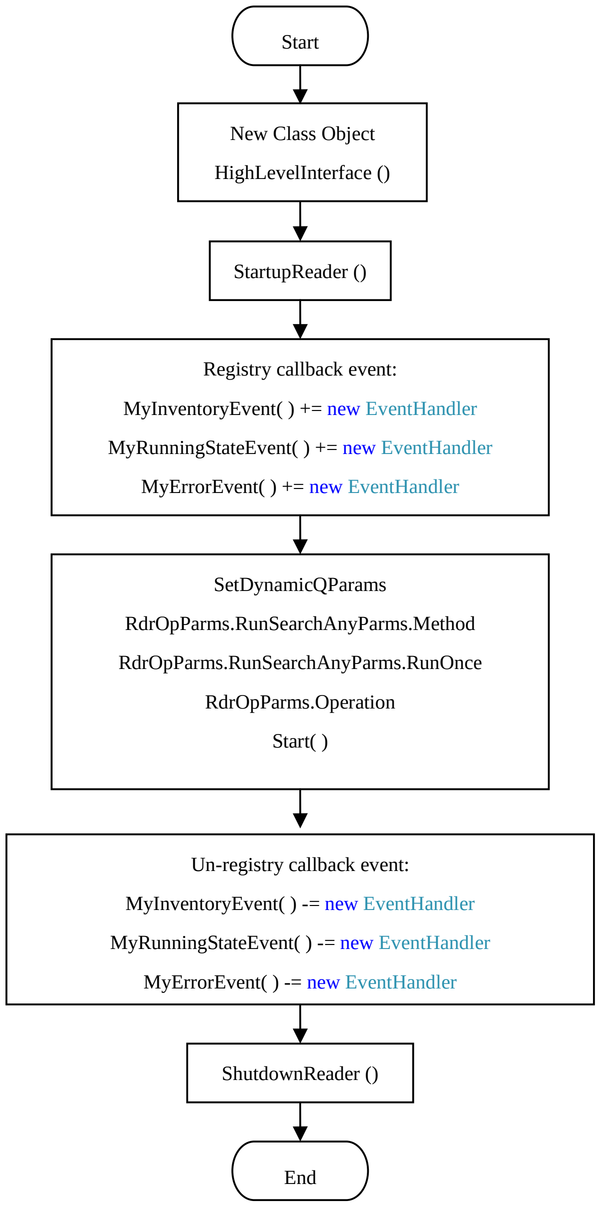
```
/// <summary>
/// Configures the post-singulation match criteria to be used by the
/// RFID radio module.  The supplied post-singulation match criteria
/// will be used for any tag-protocol operations (i.e.,
/// Inventory, etc.) in which the application specifies
/// that a post-singulation match 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)
```

7.4 Inventory

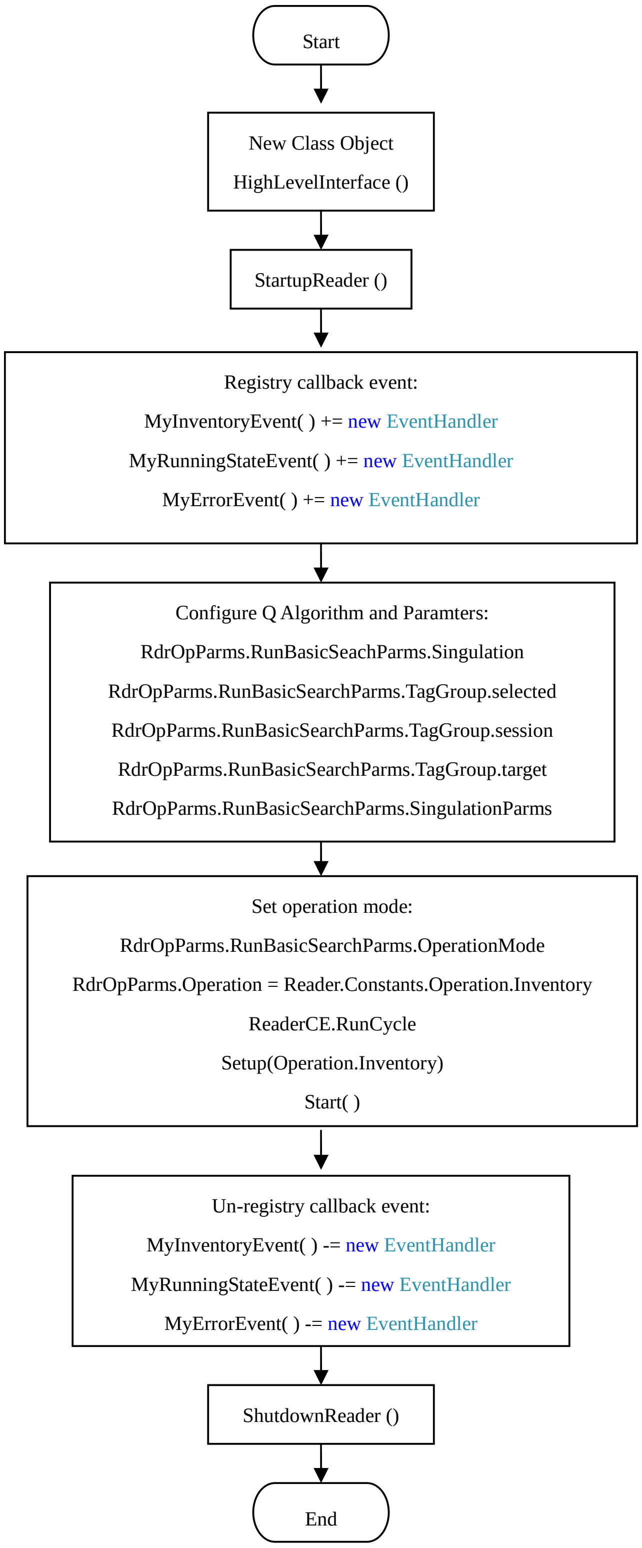
7.4.1 Flow-chart

Below are 2 different methods for doing inventory on CS203.

1) RunSearchAnyParms: This method is used for doing inventory with minimum inventory algorithm and Q parameter settings.


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
SetDynamicQParams
RdrOpParms.RunSearchAnyParms.Method
RdrOpParms.RunSearchAnyParms.RunOnce
RdrOpParms.Operation
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

2) RunBasicSearchParms: This method allows custom settings of detail inventory algorithm and parameters


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure Q Algorithm and Paramters:
RdrOpParms.RunBasicSeachParms.Singulation
RdrOpParms.RunBasicSearchParms.TagGroup.selected
RdrOpParms.RunBasicSearchParms.TagGroup.session
RdrOpParms.RunBasicSearchParms.TagGroup.target
RdrOpParms.RunBasicSearchParms.SingulationParms
Set operation mode:
RdrOpParms.RunBasicSearchParms.OperationMode
RdrOpParms.Operation = Reader.Constants.Operation.Inventory
ReaderCE.RunCycle
Setup(Operation.Inventory)
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

7.4.2 Sample Code

This example shows the 2 methods to configure the reader to run in non-continuous mode inventory with Dynamic Q algorithm (starting Q value is 7).

1) RunSearchAnyParms Method

---------------------------------- Code --

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example2
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    public static void Inventory()
    {
        //Startup Reader
        if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
        {
            MessageBox.Show("StartupReader Fail");
            return;
        }

        //Start to access any operation in library

        //attach all callback event first
        ReaderCE.MyInventoryEvent += new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent += new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyErrorEvent += new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

        //Configure Q Algorithm
       // SetDynamicQParms(StartQValue,MinQValue,MaxQValue,RetryCount,MaxQueryRepCount,
        //  ToggleTarget)
        ReaderCE.SetDynamicQParms(5, 0, 15, 0, 10, 1);
        //Apply current Q Algorithm
        ReaderCE.RdrOpParms.RunSearchAnyParms.Method = SingulationAlgorithm.DYNAMICQ;
        //non-continuous inventory
        ReaderCE.RdrOpParms.RunSearchAnyParms.RunOnce = true;
        //Operation type, current is searching any tag
        ReaderCE.RdrOpParms.Operation = Operation.SearchAnyTags;
        //start to search
        ReaderCE.Start();

        //Sleep 1 sec to wait inventort get started
        System.Threading.Thread.Sleep(1000);

        //Wait here until return to Idle state
        while (ReaderCE.MyState != ReaderOperationMode.Idle)
        {
            System.Threading.Thread.Sleep(1);
        }

        //remove all callback event
        ReaderCE.MyInventoryEvent -= new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent -= new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyErrorEvent -= new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);
        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();
    }

    static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
    }

    static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader State : {0}", e.State));
    }

    static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
e.InventoryInformation.PC,e.InventoryInformation.EPC, e.InventoryInformation.RSSI));
    }
}
```

2) RunBasicSearchParms Method

---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example2
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    public static void Inventory()
    {
        //Startup Reader
        if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
        {
            MessageBox.Show("StartupReader Fail");
            return;
        }
```

```
//Start to access any operation in library

//attach all callback event first
ReaderCE.MyInventoryEvent += new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
ReaderCE.MyRunningStateEvent += new
    EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
ReaderCE.MyErrorEvent += new
    EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

//Configure Q Algorithm
//Apply current Q Algorithm
ReaderCE.RdrOpParms.RunBasicSearchParms.Singulation = SingulationAlgorithm.DYNAMICQ;
ReaderCE.RdrOpParms.RunBasicSearchParms.TagGroup.selected = Selected.ALL;
ReaderCE.RdrOpParms.RunBasicSearchParms.TagGroup.session = Session.S0;
ReaderCE.RdrOpParms.RunBasicSearchParms.TagGroup.target = SessionTarget.A;

ReaderCE.RdrOpParms.RunBasicSearchParms.SingulationParms =
    singulation_dynamicQ.DynamicQ;
//non-continuous inventory
ReaderCE.RdrOpParms.RunBasicSearchParms.OperationMode =
    RadioOperationMode.NONCONTINUOUS;
//Operation type, current is inventory
ReaderCE.RdrOpParms.Operation = Reader.Constants.Operation.Inventory;
ReaderCE.RunCycle = 0; //set it to -1, otherwise it will stop inventory after some error,
                      // eg, network broke and reconnect
ReaderCE.Setup(Operation.Inventory);
//start to search
ReaderCE.Start();

//Wait here until return to Idle state
while (ReaderCE.MyState != ReaderOperationMode.Idle)
{
    System.Threading.Thread.Sleep(1);
}

//remove all callback event
ReaderCE.MyInventoryEvent -= new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
ReaderCE.MyRunningStateEvent -= new
    EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
ReaderCE.MyErrorEvent -= new
    EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);
//Shutdown Reader to release resources
```

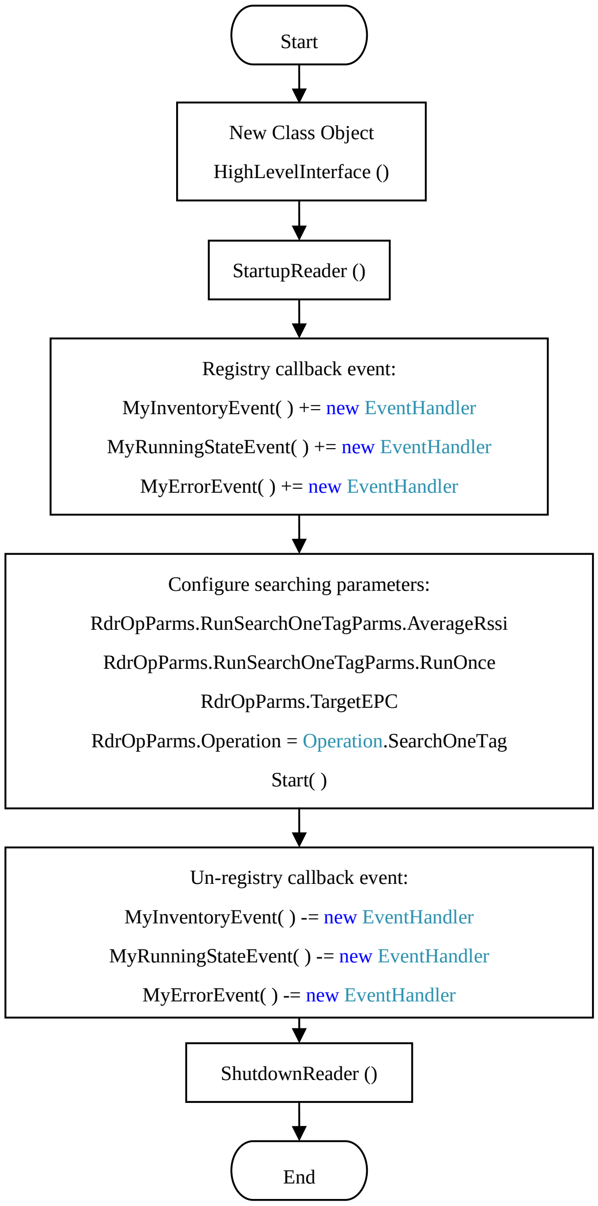
```
        ReaderCE.ShutdownReader();

    }

    static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
    }

    static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader State : {0}", e.State));
    }

    static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
e.InventoryInformation.PC,e.InventoryInformation.EPC, e.InventoryInformation.RSSI));
    }
}
```

7.5 Search a tag

7.5.1 Flow-chart

This part shows how to configure the CS203 to search a tag.

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


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure searching parameters:
RdrOpParms.RunSearchOneTagParms.AverageRssi
RdrOpParms.RunSearchOneTagParms.RunOnce
RdrOpParms.TargetEPC
RdrOpParms.Operation = Operation.SearchOneTag
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

7.5.2 Sample Code

This example shows how to configure the reader to search tag in non-continuous mode.

- SearchOneTag: searching for a tag with PC "3000" and EPC ID "800000000000000000000000"

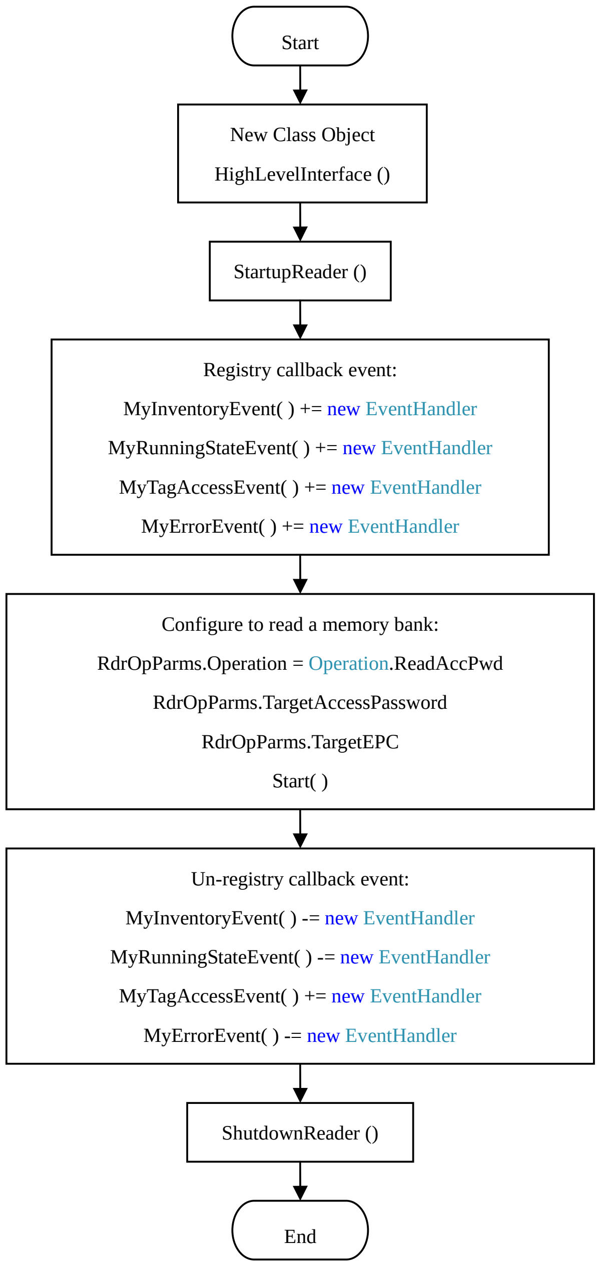
---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example3
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    public static void SearchOneTag()
    {
        //Startup Reader
        if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
        {
            MessageBox.Show("StartupReader Fail");
            return;
        }

        //Start to access any operation in library

        //attach all callback event first
        ReaderCE.MyInventoryEvent += new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent += new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyErrorEvent += new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);
        //Averaging RSSI during Searching
        ReaderCE.RdrOpParms.RunSearchOneTagParms.AveragingRssi = true;
        //Search 1 time, if search continuously set it to false
        ReaderCE.RdrOpParms.RunSearchOneTagParms.RunOnce = true;
        //Set Operation type to search one tag
        ReaderCE.RdrOpParms.Operation = Operation.SearchOneTag;
        //Searching value for tag’s PC and EPC
        ReaderCE.RdrOpParms.TargetPC = "3000";
        ReaderCE.RdrOpParms.TargetEPC = "800000000000000000000000";
        //start to search
        ReaderCE.Start();

        //Sleep 1 sec to wait inventort get started
        System.Threading.Thread.Sleep(1000);

        //Wait here until return to Idle state
        while (ReaderCE.MyState != ReaderOperationMode.Idle)
        {
            System.Threading.Thread.Sleep(1);
        }

        //remove all callback event
        ReaderCE.MyInventoryEvent -= new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent -= new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyErrorEvent -= new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);
        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();
    }

    static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
    }

    static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader State : {0}", e.State));
    }

    static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
e.InventoryInformation.PC, e.InventoryInformation.EPC, e.InventoryInformation.RSSI));
    }
}
```

7.6 Read a tag

7.6.1 Flow-chart

This part shows how to read the memory bank of a selected tag.


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure to read a memory bank:
RdrOpParms.Operation = Operation.ReadAccPwd
RdrOpParms.TargetAccessPassword
RdrOpParms.TargetEPC
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

The above flow-chart is an example for reading the Access Password data. For reading other memory bank of the tag, change the setting for the

```
RdrOpParms.Operation = Operation.<bank to read>;
```

The choice of `<bank to read>` includes:

`ReadAccPwd`: to read the access password

`ReadKillPwd`: to read the kill password

`ReadPC`: to read the PC field

`ReadEPC`: to read the EPC field

`ReadTID`: to read the TID bank

`ReadUser`: to read the User Memory bank

7.6.2 Sample Code

This example shows how to read a tag's Access Password memory data by "Operation ReadAccPwd". The target tag's EPC is "800000000000000000000000".

---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example5
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    public static void ReadTag()
    {
```

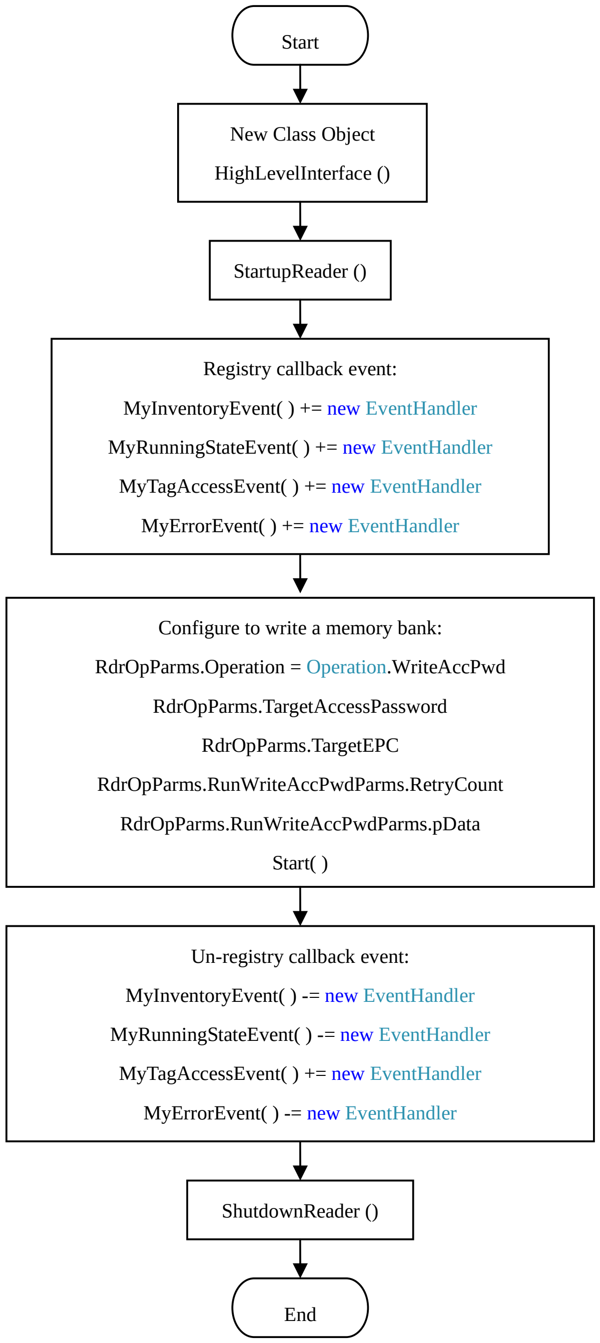
```
//Startup Reader
if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
{
    MessageBox.Show("StartupReader Fail");
    return;
}

//Start to access any operation in library

//attach all callback event first
ReaderCE.MyInventoryEvent += new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
ReaderCE.MyRunningStateEvent += new
    EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
ReaderCE.MyTagAccessEvent += new
    EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
ReaderCE.MyErrorEvent += new
    EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

//Set Operation type to read Access password
ReaderCE.RdrOpParms.Operation = Operation.ReadAccPwd;
//Input the access password to read if the bank is locked (0 means no password)
ReaderCE.RdrOpParms.TargetAccessPassword = 0x0;
//Target Tag to read
ReaderCE.RdrOpParms.TargetEPC = "800000000000000000000000";
//start to read
ReaderCE.Start();

//Sleep 1 sec to wait inventort get started
System.Threading.Thread.Sleep(1000);

//Wait here until return to Idle state
while (ReaderCE.MyState != ReaderOperationMode.Idle)
{
    System.Threading.Thread.Sleep(1);
}

//remove all callback event
ReaderCE.MyInventoryEvent -= new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
ReaderCE.MyRunningStateEvent -= new
    EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
ReaderCE.MyTagAccessEvent -= new
    EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
```

```
        ReaderCE.MyErrorEvent -= new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();
    }

    static void ReaderCE_MyTagAccessEvent(object sender, TagAccessEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Tag Access : Type = {0} Result = {1} ErrorCode =
              {2} Data = {3} Time = {4}", e.TagAccessInformation.accessType,
              e.TagAccessInformation.resultType,e.TagAccessInformation.errorCode,
              HexEncoding.ToString(e.TagAccessInformation.data),
              e.TagAccessInformation.ms_ctr));
    }

    static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
    }

    static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader State : {0}", e.State));
    }

    static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
              e.InventoryInformation.PC, e.InventoryInformation.EPC,
              e.InventoryInformation.RSSI));
    }
}
```

7.7 Write a tag

7.7.1 Flow-chart

This part shows how to write the memory bank of a selected tag.


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure to write a memory bank:
RdrOpParms.Operation = Operation.WriteAccPwd
RdrOpParms.TargetAccessPassword
RdrOpParms.TargetEPC
RdrOpParms.RunWriteAccPwdParms.RetryCount
RdrOpParms.RunWriteAccPwdParms.pData
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

The above flow-chart is an example for writing the Access Password data. For writing other memory bank of the tag, change the setting for the

```
RdrOpParms.Operation = Operation.<bank to write>;
```

The choice of `<bank to write>` includes:

`WriteAccPwd`: to write the access password

`WriteKillPwd`: to write the kill password

`WritePC`: to write the PC field

`WriteEPC`: to write the EPC field

`WriteUser`: to write the User Memory bank

7.7.2 Sample Code

This example shows how to write a tag's Access Password memory data to "01234567" by "Operation WriteAccPwd". The target tag's EPC is "800000000000000000000000".

---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example4
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    public static void WriteTag()
    {
        //Startup Reader
```

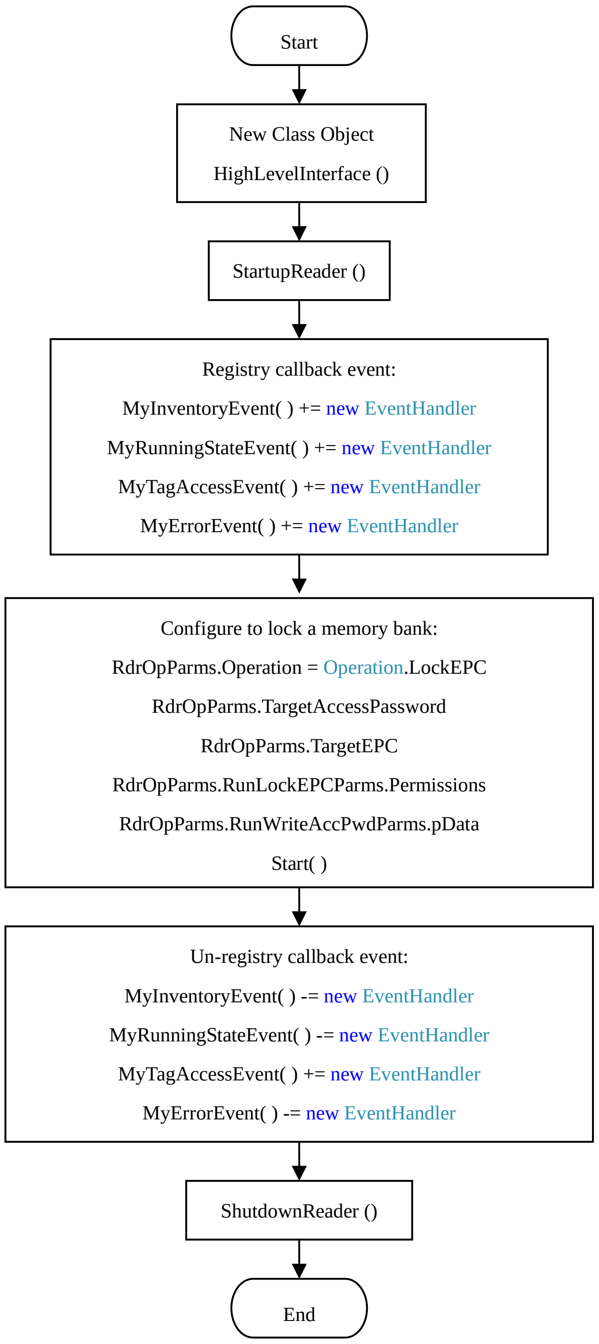
```
if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
{
    MessageBox.Show("StartupReader Fail");
    return;
}

//Start to access any operation in library

//attach all callback event first
ReaderCE.MyInventoryEvent += new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
ReaderCE.MyRunningStateEvent += new
    EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
ReaderCE.MyTagAccessEvent += new
    EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
ReaderCE.MyErrorEvent += new
    EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

//Set the Maximum retry count
ReaderCE.RdrOpParms.RunWriteAccPwdParms.RetryCount = 30;
//Set the new Data to be written to target tag
ReaderCE.RdrOpParms.RunWriteAccPwdParms.pData = 0x01234567;
//Input the current Access password if the bank is locked (0 means no password)
ReaderCE.RdrOpParms.TargetAccessPassword = 0;
//Set Operation type to write Access password
ReaderCE.RdrOpParms.Operation = Operation.WriteAccPwd;
//Target Tag to write
ReaderCE.RdrOpParms.TargetEPC = "800000000000000000000000";

//start to write
ReaderCE.Start();

//Sleep 1 sec to wait inventort get started
System.Threading.Thread.Sleep(1000);

//Wait here until return to Idle state
while (ReaderCE.MyState != ReaderOperationMode.Idle)
{
    System.Threading.Thread.Sleep(1);
}

//remove all callback event
ReaderCE.MyInventoryEvent -= new
    EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
```

```
        ReaderCE.MyRunningStateEvent -= new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyTagAccessEvent -= new
            EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
        ReaderCE.MyErrorEvent -= new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();
    }

    static void ReaderCE_MyTagAccessEvent(object sender, TagAccessEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Tag Access : Type = {0} Result = {1} ErrorCode =
              {2} Data = {3} Time = {4}", e.TagAccessInformation.accessType,
              e.TagAccessInformation.resultType, e.TagAccessInformation.errorCode,
              HexEncoding.ToString(e.TagAccessInformation.data),
              e.TagAccessInformation.ms_ctr));
    }

    static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
    }

    static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader State : {0}", e.State));
    }

    static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
    {
        Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
              e.InventoryInformation.PC, e.InventoryInformation.EPC,
              e.InventoryInformation.RSSI));
    }
}
```

7.8 Lock/Unlock a tag

7.8.1 Flow-chart

This part shows how to lock the memory bank of a selected tag.


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure to lock a memory bank:
RdrOpParms.Operation = Operation.LockEPC
RdrOpParms.TargetAccessPassword
RdrOpParms.TargetEPC
RdrOpParms.RunLockEPCParms.Permissions
RdrOpParms.RunWriteAccPwdParms.pData
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

The above flow-chart is an example for locking the EPC bank data. For locking other memory bank of the tag, change the setting for the

```
RdrOpParms.Operation = Operation.<bank to lock>;
```

The choice of `<bank to lock>` includes:

`LockAcc`: to set the security for the access password

`LockKill`: to set the security for the kill password

`LockEPC`: to set the security for the EPC bank

`LockTID`: to set the security for the TID bank

`LockUser`: to set the security for the User Memory bank

The choices of Lock Operation (`RdOpParms`) includes:

`RunLockAccParms`: Config this for locking/unlocking access password

`RunLockEPCParms`: Config this for locking/unlocking EPC

`RunLockKillParms`: Config this for locking/unlocking kill password

`RunLockTIDParms`: Config this for locking/unlocking TID

`RunLockUSERParms`: Config this for locking/unlocking User Memory

The choices of Memory Permission (`MemoryPermission`) includes:

`WRITEABLE`: The memory bank is writeable when the tag is in either the open or secured states (Unlock).

`ALWAYS_WRITEABLE`: The memory bank is writeable when the tag is in either the open or secured states and this access permission should be set permanently (Permanent Unlock).

`SECURED_WRITEABLE`: The memory bank is writeable only when the tag is in the secured state (Lock).

`ALWAYS_NOT_WRITEABLE`: The memory bank is not writeable and this access permission should be set permanently (Permanent Lock).

`NO_CHANGE`: The memory bank's access permission should remain unchanged.

7.8.2 Sample Code

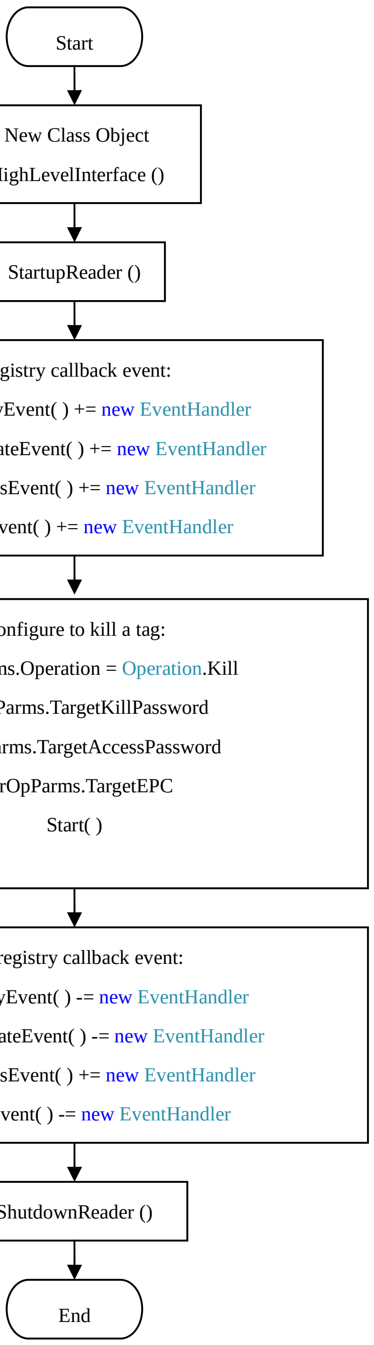
This example shows how to lock a tag's EPC memory bank by "Operation LockEPC". The target tag's EPC is "000000000000000000000001" and access password is "01234567".

---------------------------------- Code ---
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example7
{
    using rfid.Constants;
    using rfid.Structures;
    using Reader;
    using Reader.Constants;
    using Reader.Structures;
    using Reader.Utility;

    public static HighLevelInterface ReaderCE = new HighLevelInterface();
    /// <summary>
    /// Notes : You must set a non-zero lock password before lock a tag
    /// otherwise you cannot lock tag sucessfully
    /// </summary>
    public static void LockTag()
    {
        //Startup Reader
        if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
        {
            MessageBox.Show("StartupReader Fail");
            return;
        }

        //Start to access any operation in library

        //attach all callback event first
        ReaderCE.MyInventoryEvent += new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent += new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyTagAccessEvent += new
            EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
        ReaderCE.MyErrorEvent += new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

        //Set Operation type to Lock EPC
        ReaderCE.RdrOpParms.Operation = Operation.LockEPC;
        // Input the Target Access Password
        ReaderCE.RdrOpParms.TargetAccessPassword = 0x1234567;
        //Change EPC Permission to secured writeable
        ReaderCE.RdrOpParms.RunLockEPCParms.Permissions = MemoryPermission.SECURED_WRITEABLE;
        //Target Tag to lock
        ReaderCE.RdrOpParms.TargetEPC = "000000000000000000000001";

        //start to lock
        ReaderCE.Start();

        //Sleep 1 sec to wait inventort get started
        System.Threading.Thread.Sleep(1000);

        //Wait here until return to Idle state
        while (ReaderCE.MyState != ReaderOperationMode.Idle)
        {
            System.Threading.Thread.Sleep(1);
        }

        //remove all callback event
        ReaderCE.MyInventoryEvent -= new
            EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
        ReaderCE.MyRunningStateEvent -= new
            EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
        ReaderCE.MyTagAccessEvent -= new
            EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
        ReaderCE.MyErrorEvent -= new
            EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);
        //Shutdown Reader to release resources
        ReaderCE.ShutdownReader();
    }

        static void ReaderCE_MyTagAccessEvent(object sender, TagAccessEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader Tag Access : Type = {0} Result = {1} ErrorCode
                  = {2} Data = {3} Time = {4}", e.TagAccessInformation.accessType,
                  e.TagAccessInformation.resultType,
                e.TagAccessInformation.errorCode,
                HexEncoding.ToString(e.TagAccessInformation.data),
                e.TagAccessInformation.ms_ctr));
        }

        static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
        }

        static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs
                e)
        {
            Debug.WriteLine(string.Format("Reader State : {0}", e.State));
        }

        static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
                e.InventoryInformation.PC, e.InventoryInformation.EPC,
                e.InventoryInformation.RSSI));
        }
    }
```

7.9 Kill a tag

7.9.1 Flow-chart

This part shows how to kill a selected tag.


Start
New Class Object
HighLevelInterface ()
StartupReader ()
Registry callback event:
MyInventoryEvent( ) += new EventHandler
MyRunningStateEvent( ) += new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) += new EventHandler
Configure to kill a tag:
RdrOpParms.Operation = Operation.Kill
RdrOpParms.TargetKillPassword
RdrOpParms.TargetAccessPassword
RdrOpParms.TargetEPC
Start( )
Un-registry callback event:
MyInventoryEvent( ) -= new EventHandler
MyRunningStateEvent( ) -= new EventHandler
MyTagAccessEvent( ) += new EventHandler
MyErrorEvent( ) -= new EventHandler
ShutdownReader ()
End

7.9.2 Sample Code

This example shows how to kill a tag's EPC memory bank by "Operation Kill". The target tag's EPC is "000000000000000000000001", kill password is "01234567" and access password is "01234567".

---------------------------------- Code ---

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Threading;
using System.Diagnostics;

static class example6
    {
        using rfid.Constants;
        using rfid.Structures;
        using Reader;
        using Reader.Constants;
        using Reader.Structures;
        using Reader.Utility;

        public static HighLevelInterface ReaderCE = new HighLevelInterface();
        /// <summary>
        /// Notes : You must set a non-zero Kill password before kill a tag
        /// otherwise you will not kill tag sucessfully
        /// </summary>
        public static void KillTag()
        {
            //Startup Reader
            if (ReaderCE.StartupReader(IP, 0) != rfid.Constants.Result.OK)
            {
                MessageBox.Show("StartupReader Fail");
                return;
            }

            //Start to access any operation in library

            //attach all callback event first
            ReaderCE.MyInventoryEvent += new
                EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
            ReaderCE.MyRunningStateEvent += new
```

```
                EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
            ReaderCE.MyTagAccessEvent += new
                EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
            ReaderCE.MyErrorEvent += new
                EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

            //Set Operation type to kill tag
            ReaderCE.RdrOpParms.Operation = Operation.Kill;
            //Input the Target Kill Password of the tag which is non-zero password.
            ReaderCE.RdrOpParms.TargetKillPassword = "01234567";
            //Input the Target Access Password of the tag
            ReaderCE.RdrOpParms.TargetAccessPassword = 0x1234567;
            //Target Tag to kill
            ReaderCE.RdrOpParms.TargetEPC = "000000000000000000000001";

            //start to kill
            ReaderCE.Start();

            //Sleep 1 sec to wait inventort get started
            System.Threading.Thread.Sleep(1000);

            //Wait here until return to Idle state
            while (ReaderCE.MyState != ReaderOperationMode.Idle)
            {
                System.Threading.Thread.Sleep(1);
            }

            //remove all callback event
            ReaderCE.MyInventoryEvent -= new
                EventHandler<InventoryEventArgs>(ReaderCE_MyInventoryEvent);
            ReaderCE.MyRunningStateEvent -= new
                EventHandler<ReaderOperationModeEventArgs>(ReaderCE_MyRunningStateEvent);
            ReaderCE.MyTagAccessEvent -= new
                EventHandler<TagAccessEventArgs>(ReaderCE_MyTagAccessEvent);
            ReaderCE.MyErrorEvent -= new
                EventHandler<ErrorEventArgs>(ReaderCE_MyErrorEvent);

            //Shutdown Reader to release resources
            ReaderCE.ShutdownReader();
        }

        static void ReaderCE_MyTagAccessEvent(object sender, TagAccessEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader Tag Access : Type = {0} Result = {1} ErrorCode
                = {2} Data = {3} Time = {4}", e.TagAccessInformation.accessType,
                e.TagAccessInformation.resultType, e.TagAccessInformation.errorCode,
                HexEncoding.ToString(e.TagAccessInformation.data),
                e.TagAccessInformation.ms_ctr));
        }

        static void ReaderCE_MyErrorEvent(object sender, ErrorEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader error : {0} {1}", e.ErrorType, e.ErrorCode));
        }

        static void ReaderCE_MyRunningStateEvent(object sender, ReaderOperationModeEventArgs
                e)
        {
            Debug.WriteLine(string.Format("Reader State : {0}", e.State));
        }

        static void ReaderCE_MyInventoryEvent(object sender, InventoryEventArgs e)
        {
            Debug.WriteLine(string.Format("Reader Data : PC = {0} EPC = {1} RSSI = {2}",
                e.InventoryInformation.PC, e.InventoryInformation.EPC,
                e.InventoryInformation.RSSI));
        }
    }
```

8 Building and Deploying DemoApp on Visual Studio

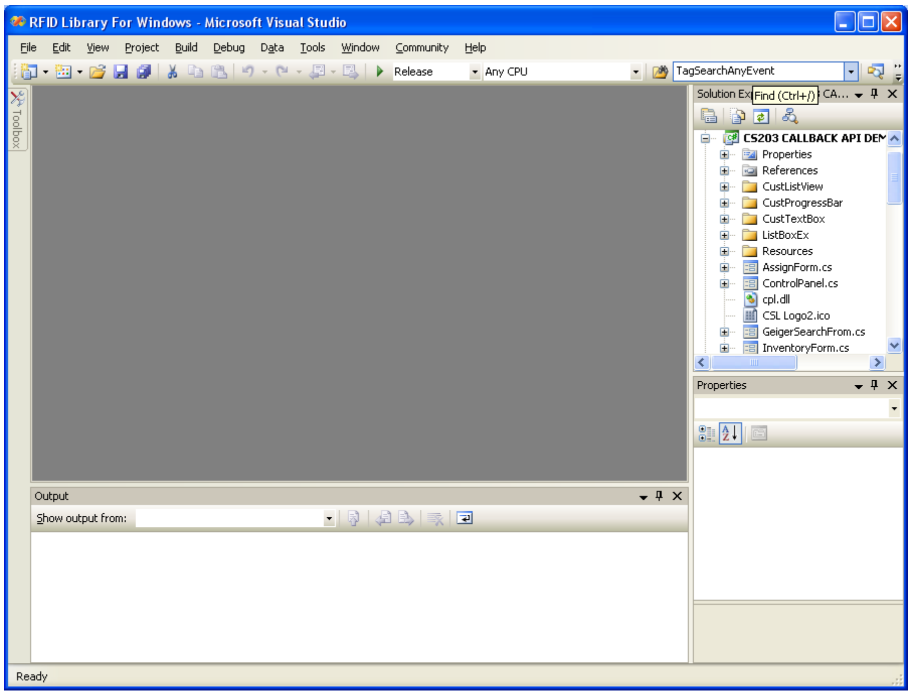
8.1 Callback-based API DemoApp

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

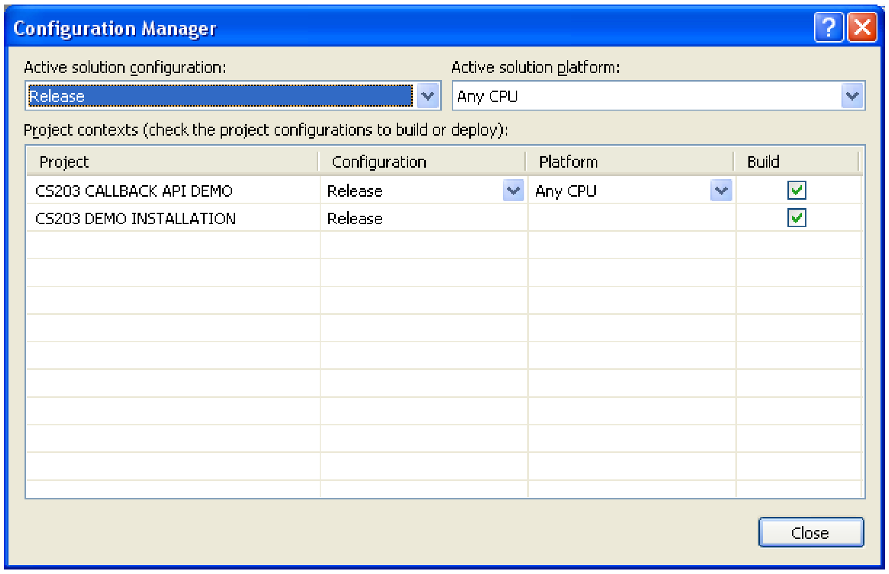
Folder/Files	Content
CS203 Callback API Cust	Source code of the CS203 sample program
CSLibrary	CS203 library files
DEMO	Installation files of the sample programs
Document	User Guides and API document
GPIO	Source code for CS203 GPIO control program
Start Stop	Source code of the CS203 Start-Stop test program

8.2 Building the DemoApp program

1. Open the DemoApp program (RFID Library For Windows.sln) on the development platform by Visual Studio 2005.



2. Click "Build" → "Configuration Manager" on the taskbar of Visual Studio to open the configuration manager.
   Select "Release" under "Active solution configuration" and "Any CPU" under "Active solution platform". Choose the projects you want to build in the list and then click "Close" button.



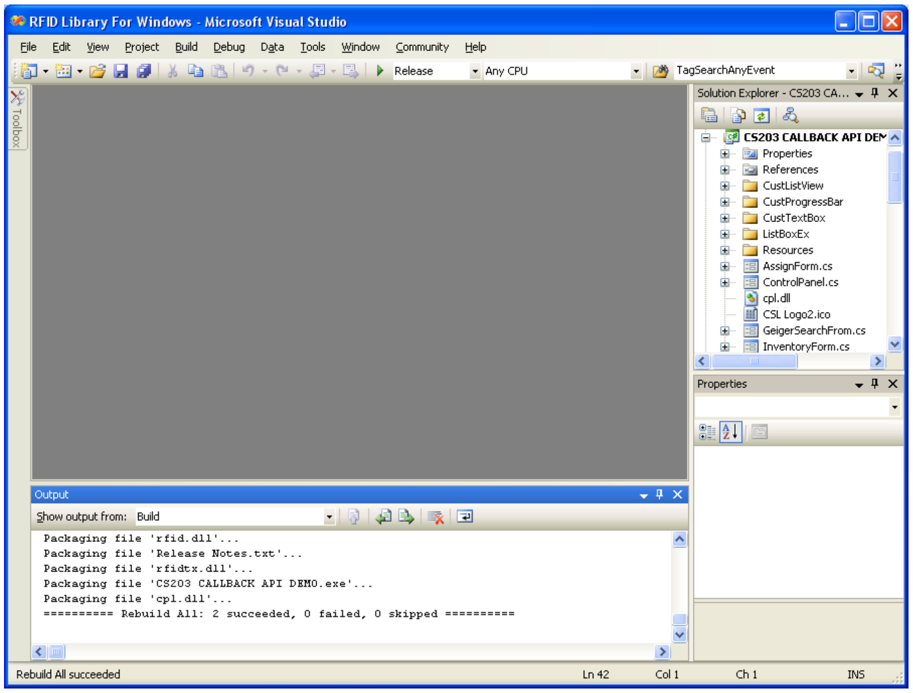
Active solution configuration: Release

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
CS203 CALLBACK API DEMO	Release	Any CPU	✓
CS203 DEMO INSTALLATION	Release		✓

3. Click "Build" → "Rebuild Solution" to rebuild the project on Visual Studio. Wait until the rebuild success.



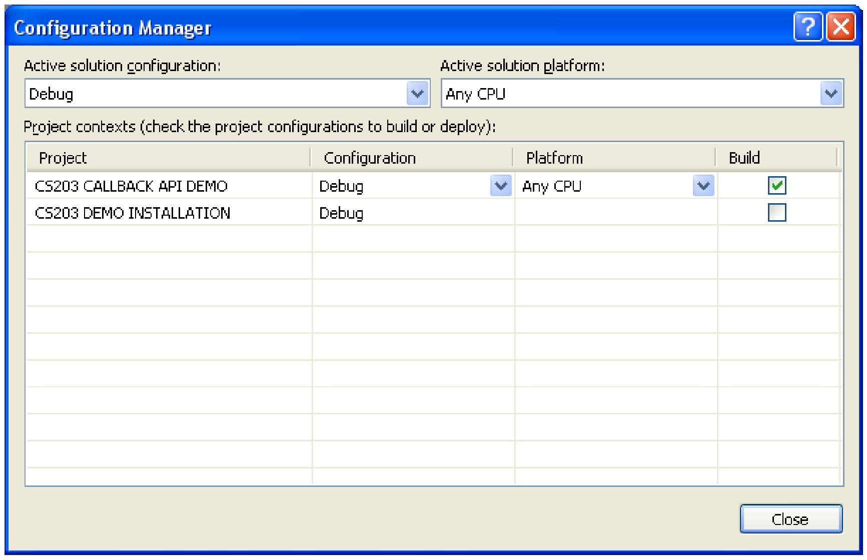
```
Packaging file 'rfid.dll'...
Packaging file 'Release Notes.txt'...
Packaging file 'rfidtx.dll'...
Packaging file 'CS203 CALLBACK API DEMO.exe'...
Packaging file 'cpl.dll'...
========== Rebuild All: 2 succeeded, 0 failed, 0 skipped ==========
```

Rebuild All succeeded

4. After the build success, the executable binary “CS203 CALLBACK API DEMO.exe” is in the folder “CS203 CALLBACK API DEMO\bin\Release\”.
5. Connect the CS203 reader with the development PC on the network (same subnet).
6. Run the DemoApp program by clicking the “CS203 CALLBACK API DEMO.exe” file to connect with the CS203 reader.

8.3 Debugging the DemoApp program

1. In addition to running the built binary file, you could also “debug” the program.
2. After opening the DemoApp program on Visual Studio, click “Build” → “Configuration Manager”.
   Select “Debug” under “Active solution configuration” and “Any CPU” under “Active solution platform”. Choose the project you want to build in the list and then click “Close” button.



Active solution configuration: Debug

Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
CS203 CALLBACK API DEMO	Debug	Any CPU	☑
CS203 DEMO INSTALLATION	Debug		☐

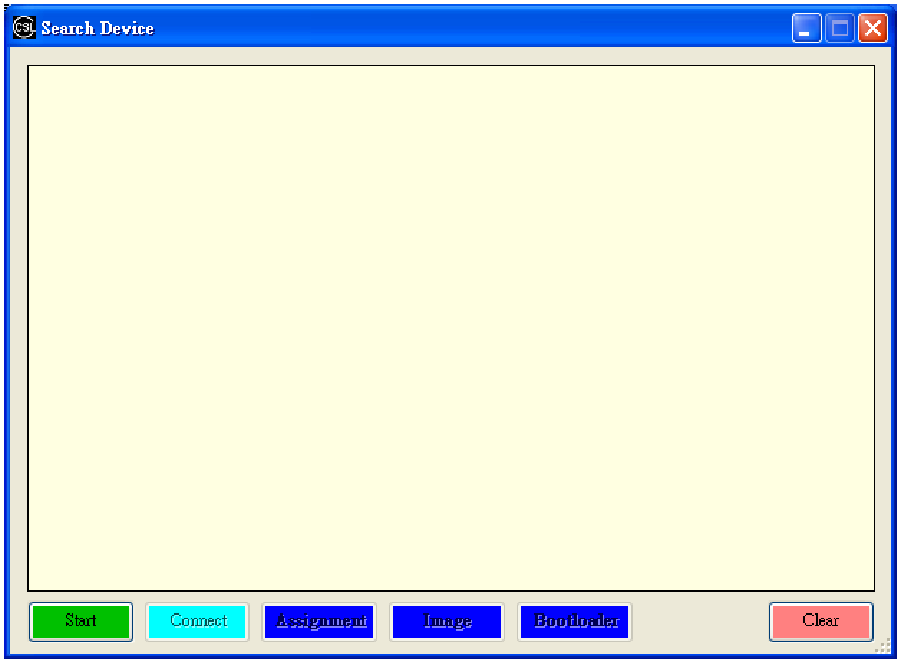
3. Click “Build” → “Rebuild Solution” to rebuild the project on Visual Studio. Wait until the rebuild success.
4. Connect the CS203 with the development PC on the network (same subnet).
5. You can set any break points on the program code if necessary for debugging.
6. Click “Debug” → “Start debugging” on Visual Studio
7. After that, the program is running on the development PC and you can now start debugging the program.

9 Using Callback-based API DemoApp

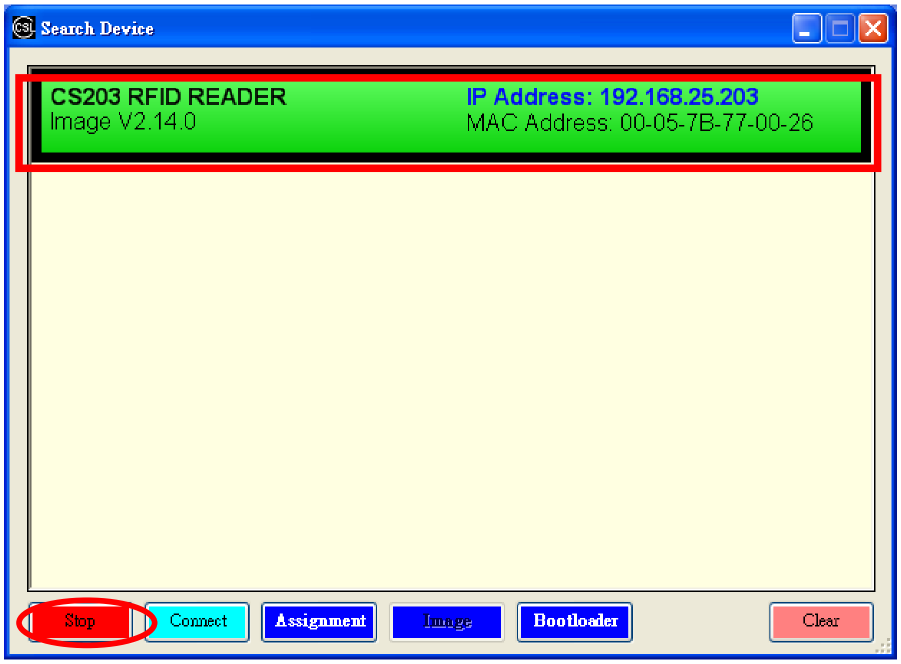
This chapter describes the details about how to use the CS203 demo application program that is based on the Callback-Based API.

9.1 *CS203 Callback API Demo Program Operations*

9.1.1 Searching for CS203 device



For the latest demo application, you can choose specific device to connect.



CS203 RFID READER
Image V2.14.0
IP Address: 192.168.25.203
MAC Address: 00-05-7B-77-00-26

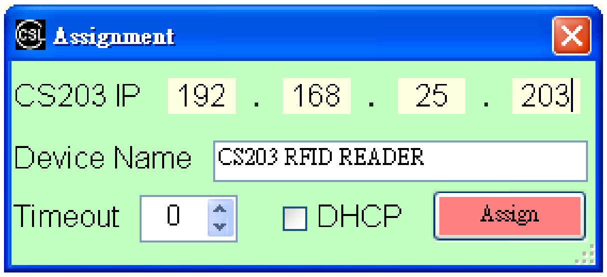
Click "Start" button to search device in the same network.

If you can't find any device, please check the following:

i. make sure the Dot Net Framework 3.5 is installed
ii. make sure the Visual C++ 2005 Redistributable package is installed
iii. Disable the firewall setting on the PC or network (or open the port number 1515 and 1516)
iv. Reboot the CS203 device

9.1.2 Network Configuration of CS203

After you have found a CS203 device, you can configure the target device IP address (DHCP or static IP), Device Name and TCP timeout in "Assignment".

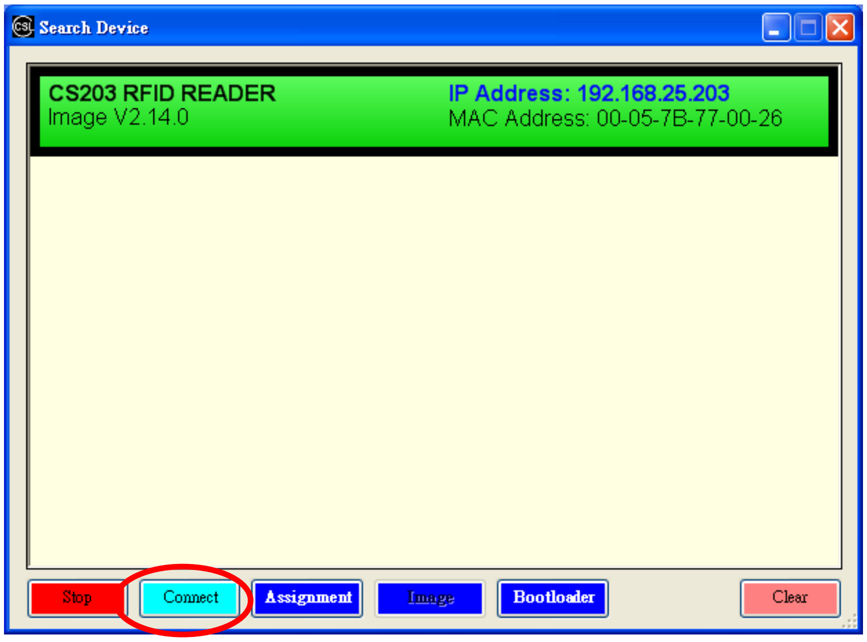


CS203 IP 192 . 168 . 25 . 203
Device Name CS203 RFID READER
Timeout 0
☐ DHCP

Note: Set TCP timeout to be zero at this moment.

9.1.3 Connecting to CS203

In order to connect to a CS203 device, select the device on the list and click the "Connect" button.

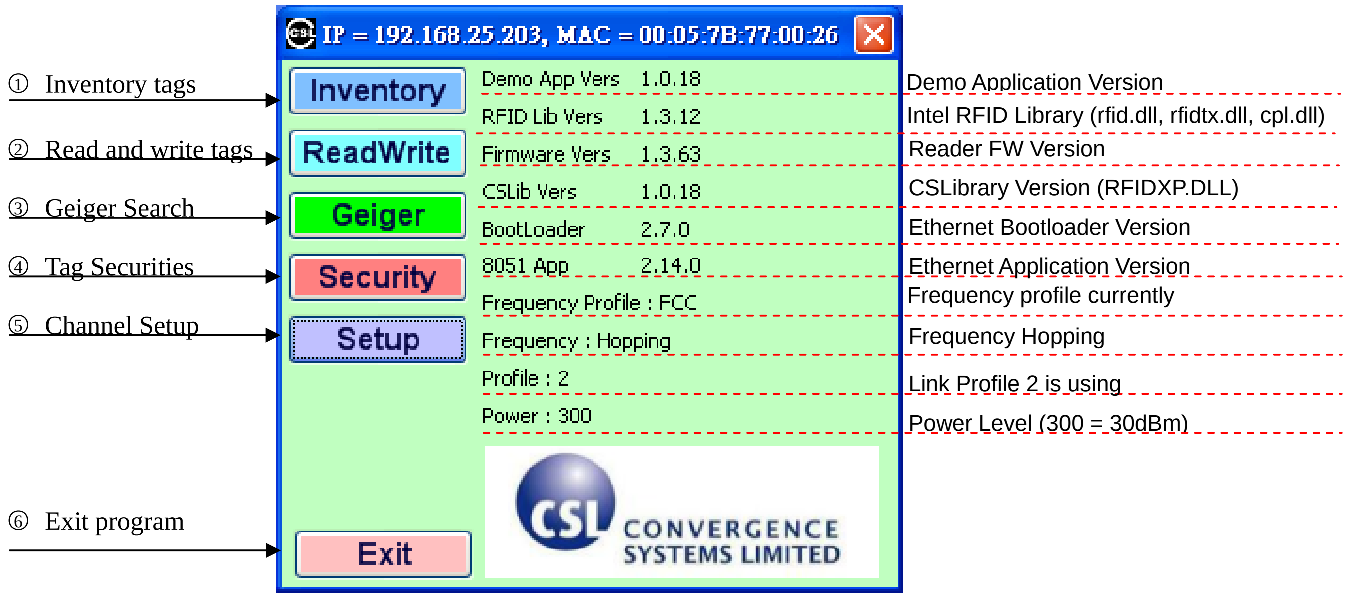


CS203 RFID READER
Image V2.14.0
IP Address: 192.168.25.203
MAC Address: 00-05-7B-77-00-26

Choose a device and click "Connect" button.

9.1.4 Main Menu

In the main menu of the CS203 CALLBACK API DEMO program, the configuration information of the reader is shown and you can select the various functions.

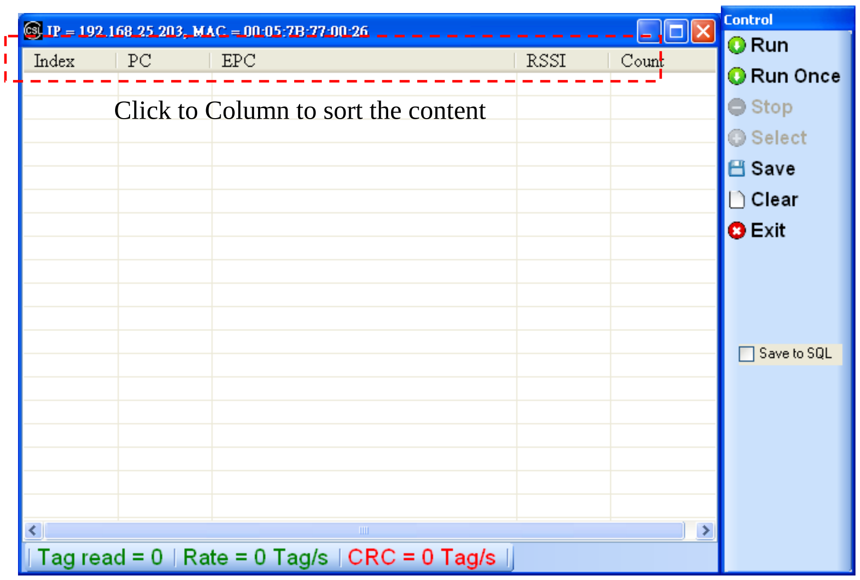


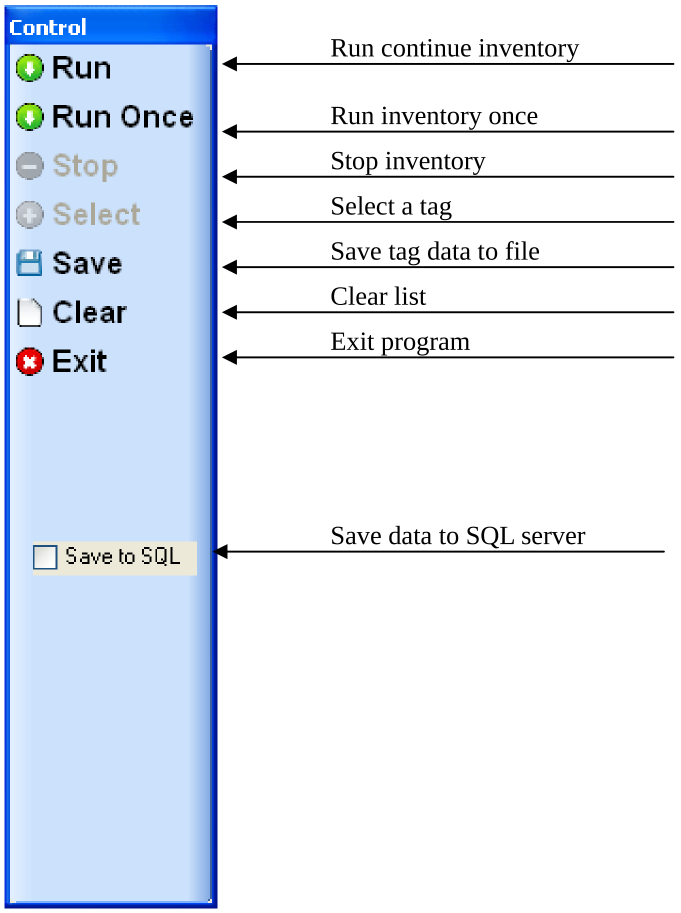
Demo App Vers 1.0.18	Demo Application Version
RFID Lib Vers 1.3.12	Intel RFID Library (rfid.dll, rfidtx.dll, cpl.dll)
Firmware Vers 1.3.63	Reader FW Version
CSLib Vers 1.0.18	CSLibrary Version (RFIDXP.DLL)
BootLoader 2.7.0	Ethernet Bootloader Version
8051 App 2.14.0	Ethernet Application Version
Frequency Profile : FCC	Frequency profile currently
Frequency : Hopping	Frequency Hopping
Profile : 2	Link Profile 2 is using
Power : 300	Power Level (300 = 30dBm)

9.1.5 Inventory

This page demonstrates the tag inventory functions for reading tags continuously with the RSSI value and read count.

Click the "Run" button to start reading tags.

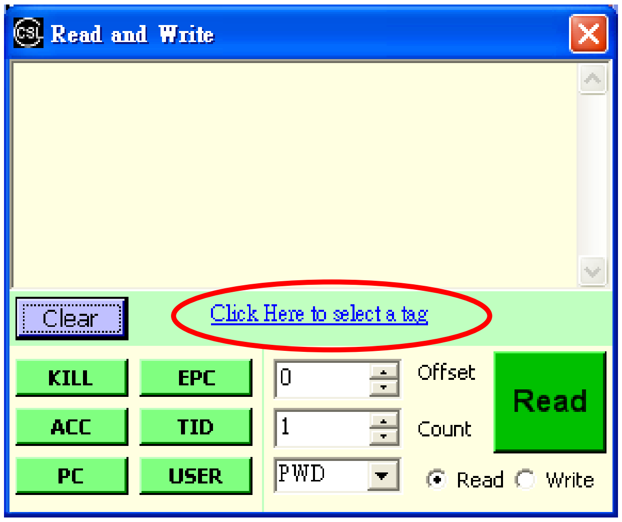



Control
Run
Run Once
Stop
Select
Save
Clear
Exit
Save to SQL
Run continue inventory
Run inventory once
Stop inventory
Select a tag
Save tag data to file
Clear list
Exit program
Save data to SQL server

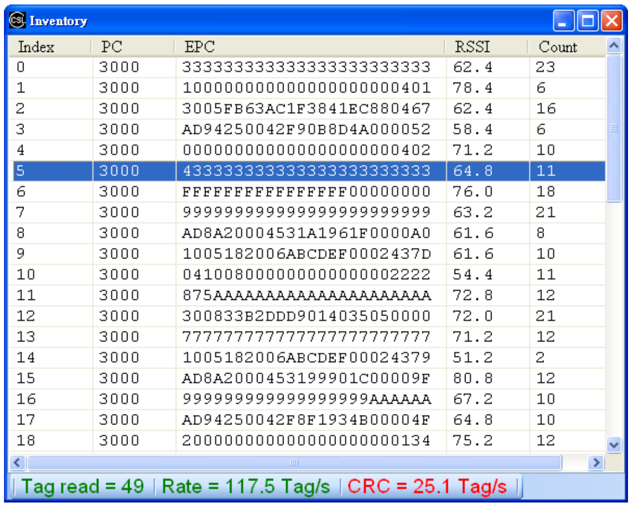
9.1.6 Read/Write

This page demonstrates the function of reading and writing different memory banks of a selected tag.

Click on the "Click Here to select a tag" to scan for and select the tag you want to access.



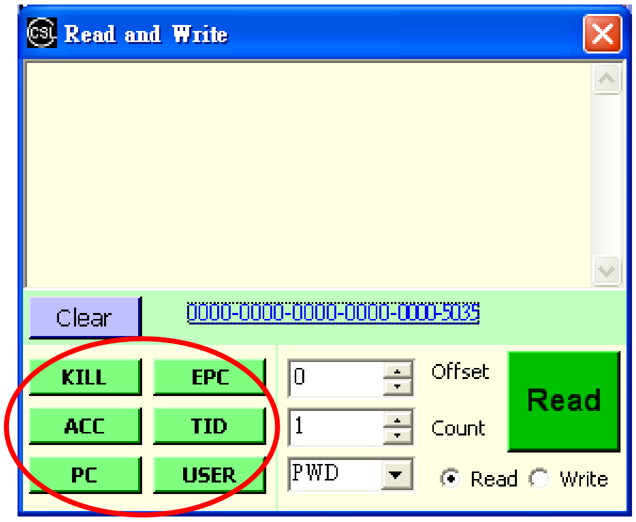
Offset: 0
Count: 1
PWD
[✓ Read] Write



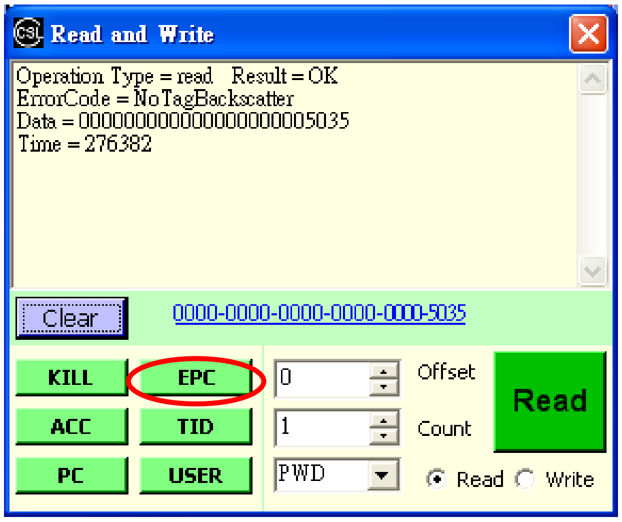
Index	PC	EPC	RSSI	Count
0	3000	333333333333333333333333	62.4	23
1	3000	100000000000000000000401	78.4	6
2	3000	3005FB63AC1F3841EC880467	62.4	16
3	3000	AD94250042F90B8D4A000052	58.4	6
4	3000	000000000000000000000402	71.2	10
5	3000	433333333333333333333333	64.8	11
6	3000	FFFFFFFFFFFFFFFF00000000	76.0	18
7	3000	999999999999999999999999	63.2	21
8	3000	AD8A20004531A1961F0000A0	61.6	8
9	3000	1005182006ABCDEF0002437D	61.6	10
10	3000	041008000000000000002222	54.4	11
11	3000	875AAAAAAAAAAAAAAAAAAAAA	72.8	12
12	3000	300833B2DDD9014035050000	72.0	21
13	3000	777777777777777777777777	71.2	12
14	3000	1005182006ABCDEF00024379	51.2	2
15	3000	AD8A2000453199901C00009F	80.8	12
16	3000	999999999999999999AAAAAA	67.2	10
17	3000	AD94250042F8F1934B00004F	64.8	10
18	3000	200000000000000000000134	75.2	12

Tag read = 49 | Rate = 117.5 Tag/s | CRC = 25.1 Tag/s

After the tag is selected, you can click on the left hand side hotkey buttons or the "Read" button to read the corresponding data of the tag.

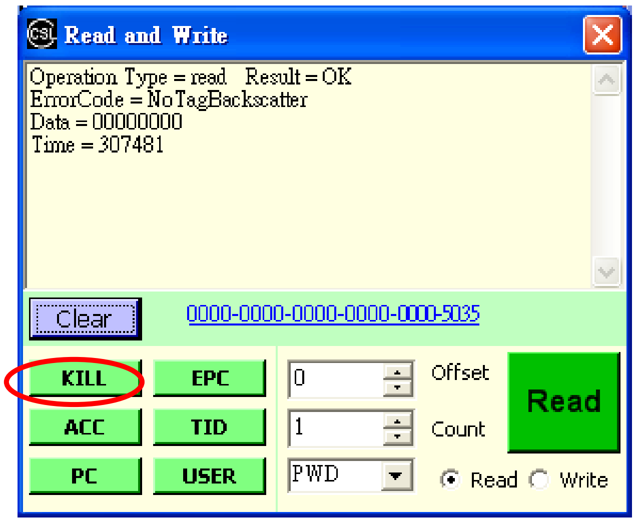


Read EPC: click on the "EPC" button to read the EPC ID



```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = 000000000000000000005035
Time = 276382
```

Read Kill Password: click on the "Kill" button to read the kill password

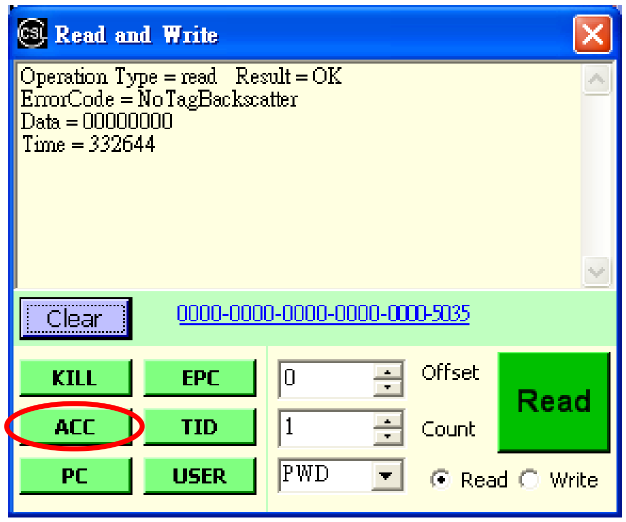


Operation Type = read Result = OK
ErrorCode = NoTagBackscatter
Data = 00000000
Time = 307481

0000-0000-0000-0000-0000-5035

Offset: 0
Count: 1
PWD
[✓ Read] Write

Read Access Password: click on the "ACC" button to read the access password.

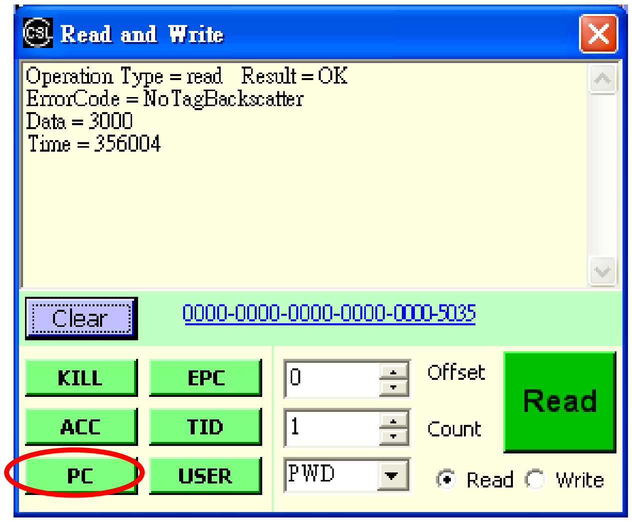


Operation Type = read Result = OK
ErrorCode = NoTagBackscatter
Data = 00000000
Time = 332644

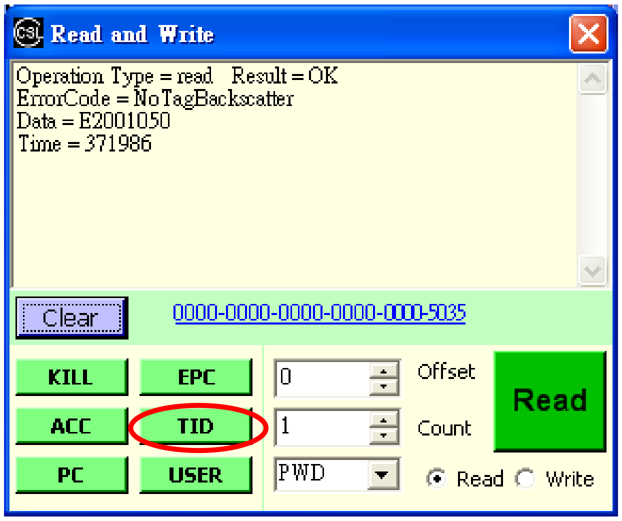
0000-0000-0000-0000-0000-5035

Offset: 0
Count: 1
PWD
[✓ Read] Write

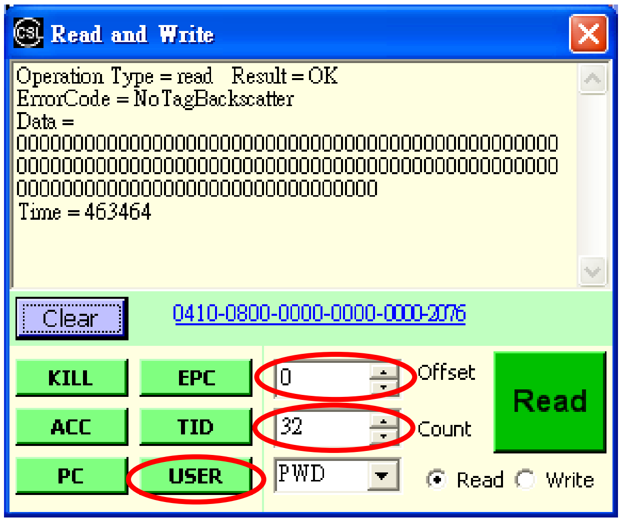
Read Protocol Control (PC): click on the "PC" button to read the PC value.


```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = 3000
Time = 356004
```

Read TID Value: click on the "TID" button to read the TID value.


```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = E2001050
Time = 371986
```

Read User Memory: Set the offset word (starting word) and length of words (Count) you want to read for the user memory bank and click "USER" button to read it.



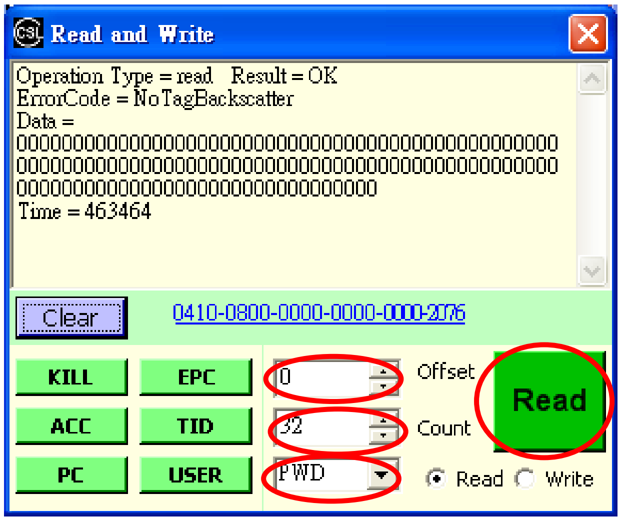
```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data =
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
00000000000000000000000000000000
Time = 463464
```

Offset: 0
Count: 32
PWD
Read
[✓ Read] Write

For example, if you want to read from the 17th bit (after the first word) for a length of 32 bits (2 words), you should set Offset = 1 and Count = 2.

In addition to using the hotkeys on the left hand side, you could also use the "Read" button to read tags as follows:

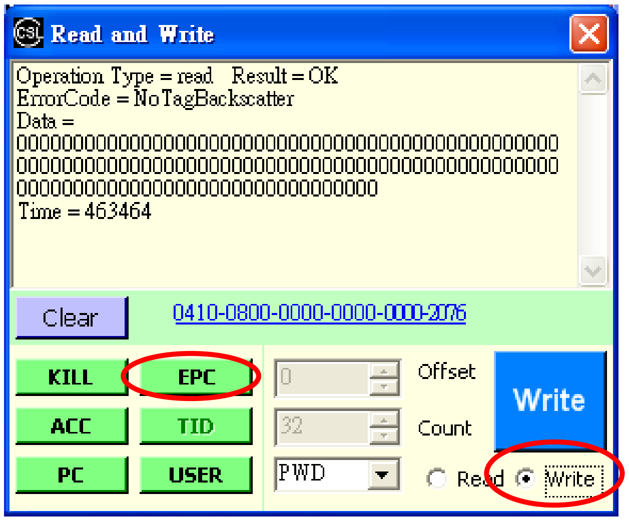
1. Set the starting word you want to read at "Offset"
2. Set length of word you want to read at "Count"
3. Select the memory bank you want to read (PWD, PC-EPC, TID, USER)
4. Click "Read" button to read the data



```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data =
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
00000000000000000000000000000000
Time = 463464
```

Offset: 0
Count: 32
PWD
Read
[✓ Read] Write

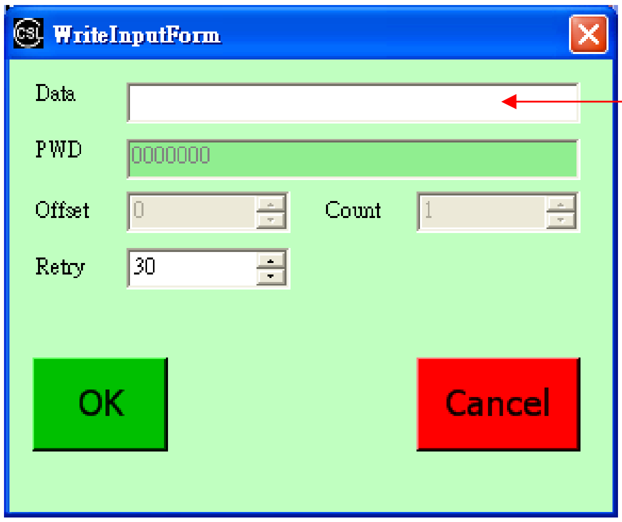
Write EPC: select "Write" and click on "EPC" button to enter the write EPC page



```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data =
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
0000000000000000000000000000000
Time = 463464
```

0410-0800-0000-0000-0000-2076

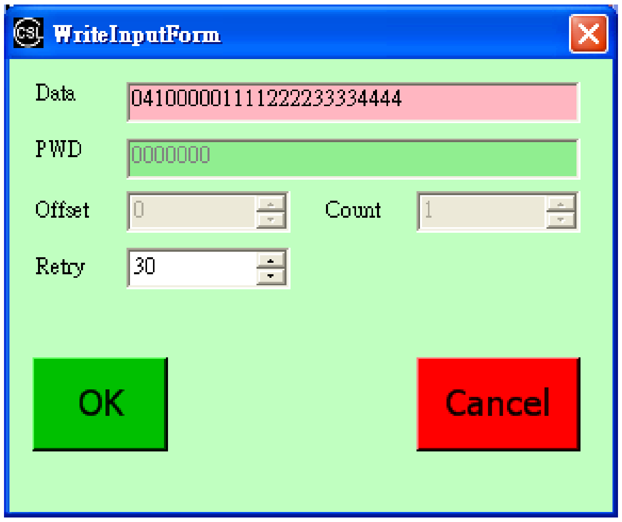
Offset: 0
Count: 32
PWD
Read [✓ Write]



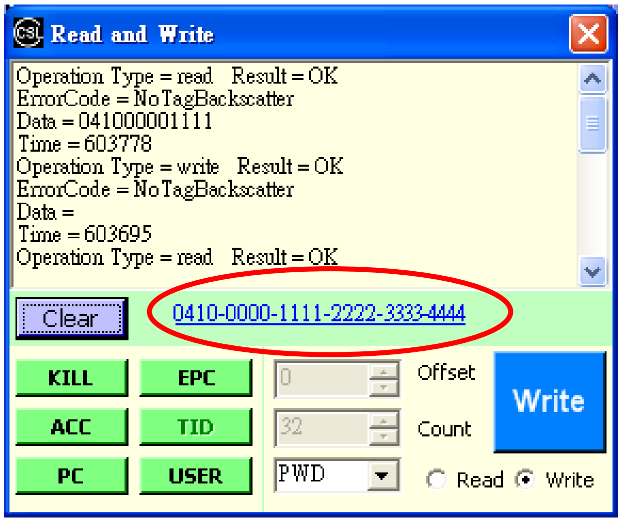
Data:
PWD: 0000000
Offset: 0
Count: 1
Retry: 30

Input new EPC here

Input the new EPC ID in the "Data" field and then click "OK" button



Data: 041000001111222233334444
PWD: 0000000
Offset: 0
Count: 1
Retry: 30



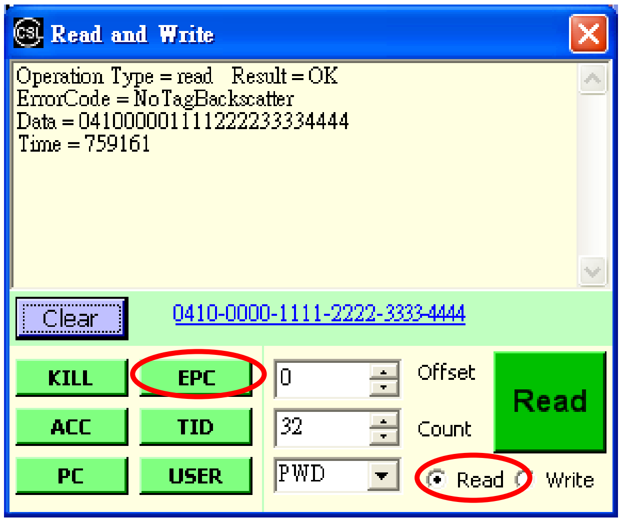
```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = 041000001111
Time = 603778
Operation Type = write   Result = OK
ErrorCode = NoTagBackscatter
Data =
Time = 603695
Operation Type = read   Result = OK
```

0410-0000-1111-2222-3333-4444

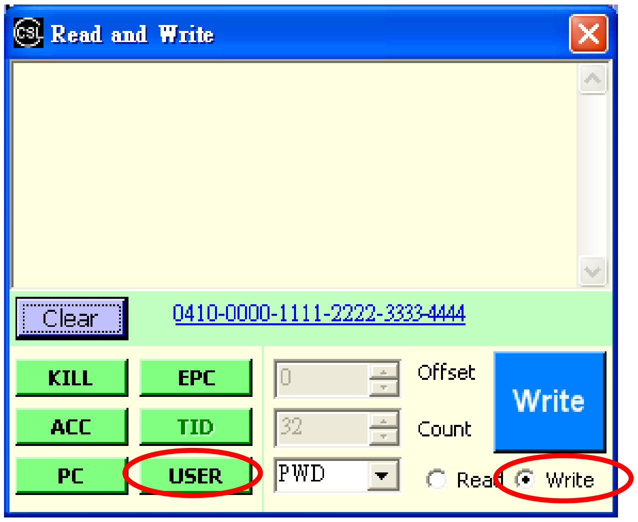
Offset: 0
Count: 32
PWD
Read [✓ Write]

If write success, "Result = OK" will be shown on the screen.

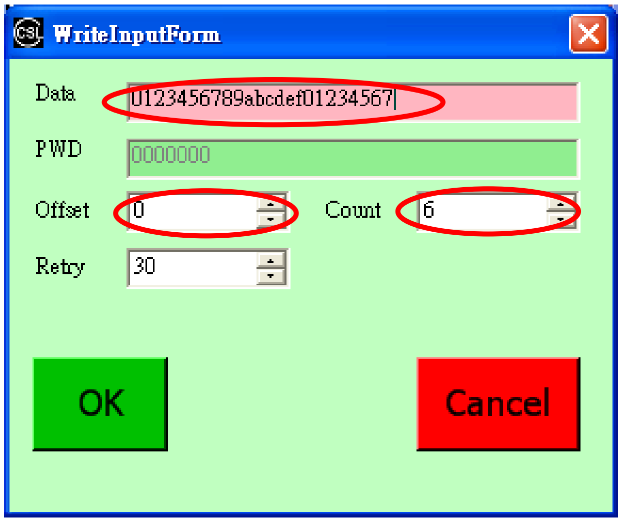
After the EPC is written, you could verify it by reading the EPC ID again


Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = 0410000011112222333344444
Time = 759161

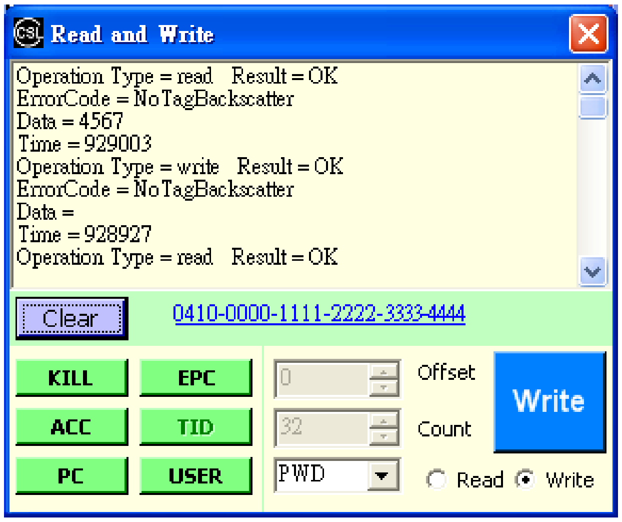
Write User Memory: select "Write" and click on "USER" button to enter the write user memory page.



Select the offset word and length of words you want to write, then input the data into "Data" field and click "OK" button to write the tag.



Data: 0123456789abcdef01234567
PWD: 0000000
Offset: 0
Count: 6
Retry: 30

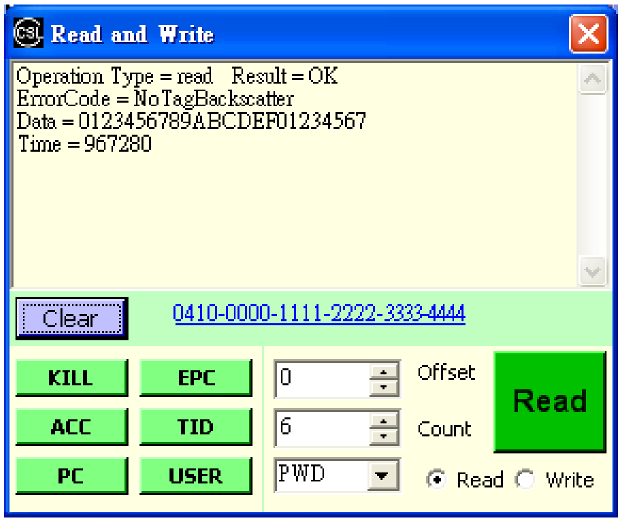


```
Operation Type = read   Result = OK
ErrorCode = NoTagBackscatter
Data = 4567
Time = 929003
Operation Type = write   Result = OK
ErrorCode = NoTagBackscatter
Data =
Time = 928927
Operation Type = read   Result = OK
```

0410-0000-1111-2222-3333-4444

Offset: 0
Count: 32
PWD
Read [✓ Write]

After writing the user memory, you can verify it by reading the user memory again.

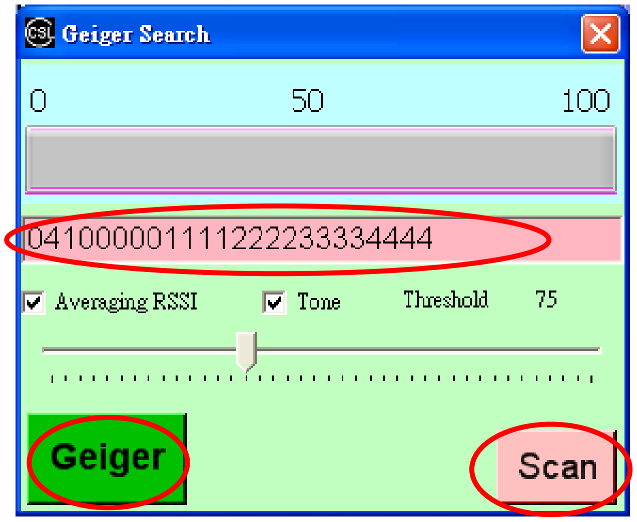


```
Operation Type = read    Result = OK
ErrorCode = NoTagBackscatter
Data = 0123456789ABCDEF01234567
Time = 967280
```

Offset: 0
Count: 6
PWD
[✓ Read] Write

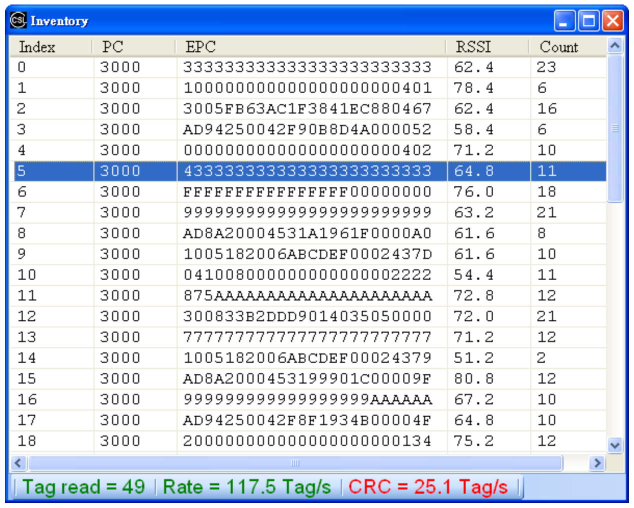
9.1.7 Geiger Counter Search

This page demonstrates the Geiger counter tag search mode. Input the EPC ID of the tag (or scan a tag) you want to search and then click the "Geiger" button.



041000001111222233334444

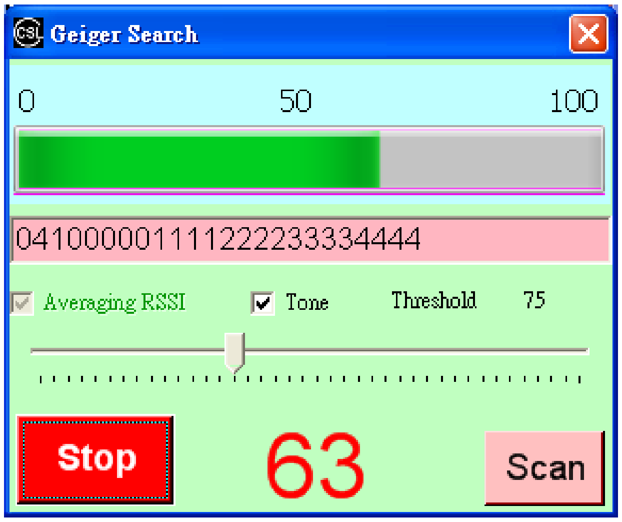
☑ Averaging RSSI ☑ Tone Threshold 75



Index	PC	EPC	RSSI	Count
0	3000	333333333333333333333333	62.4	23
1	3000	100000000000000000000401	78.4	6
2	3000	3005FB63AC1F3841EC880467	62.4	16
3	3000	AD94250042F90B8D4A000052	58.4	6
4	3000	000000000000000000000402	71.2	10
5	3000	433333333333333333333333	64.8	11
6	3000	FFFFFFFFFFFFFFFF00000000	76.0	18
7	3000	999999999999999999999999	63.2	21
8	3000	AD8A20004531A1961F0000A0	61.6	8
9	3000	1005182006ABCDEF0002437D	61.6	10
10	3000	041008000000000000002222	54.4	11
11	3000	875AAAAAAAAAAAAAAAAAAAAA	72.8	12
12	3000	300833B2DDD9014035050000	72.0	21
13	3000	777777777777777777777777	71.2	12
14	3000	1005182006ABCDEF00024379	51.2	2
15	3000	AD8A2000453199901C00009F	80.8	12
16	3000	999999999999999999AAAAAA	67.2	10
17	3000	AD94250042F8F1934B00004F	64.8	10
18	3000	200000000000000000000134	75.2	12

Tag read = 49 | Rate = 117.5 Tag/s | CRC = 25.1 Tag/s

When the tag is seen, it shows the RSSI value.



041000001111222233334444

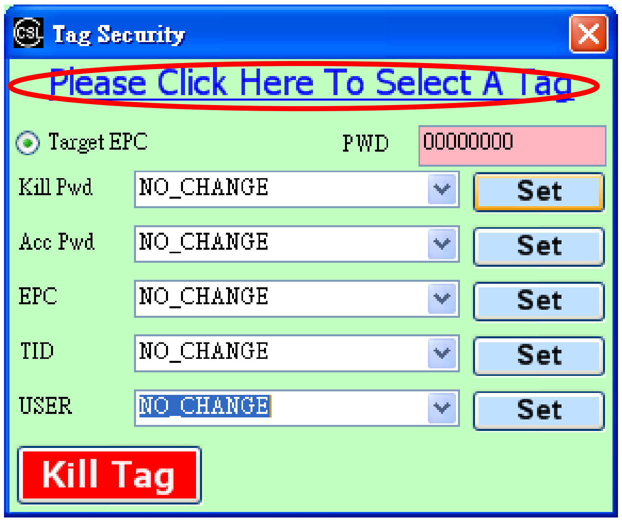
[✓ Averaging RSSI] [✓ Tone] Threshold 75

63

9.1.8 Tag Security

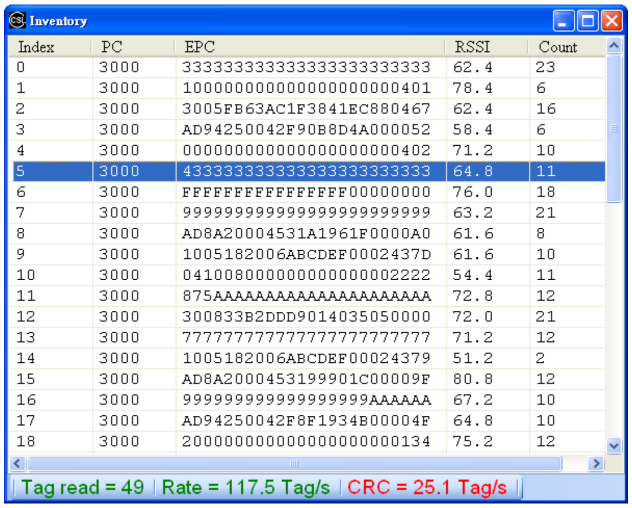
This page demonstrates the tag security operations (lock, unlock and kill)

Click "*Please Click Here To Select A Tag*" to scan for and select the tag you want to access.



[● Target EPC]	PWD	00000000
Kill Pwd	NO_CHANGE	Set
Acc Pwd	NO_CHANGE	Set
EPC	NO_CHANGE	Set
TID	NO_CHANGE	Set
USER	NO_CHANGE	Set

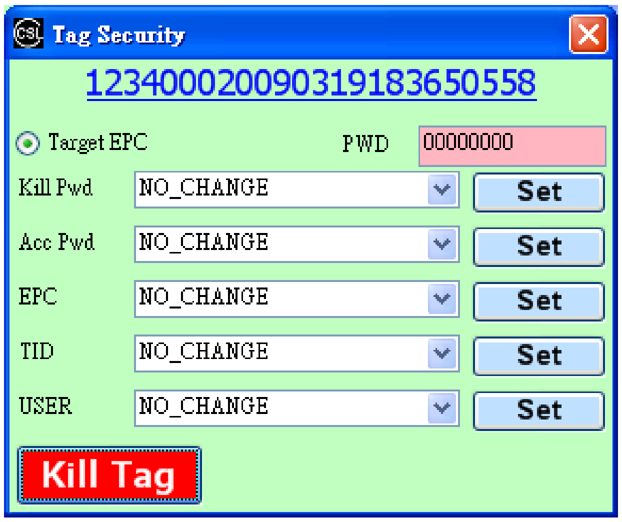
Highlight and select the tag you want to access in the list.



Index	PC	EPC	RSSI	Count
0	3000	333333333333333333333333	62.4	23
1	3000	100000000000000000000401	78.4	6
2	3000	3005FB63AC1F3841EC880467	62.4	16
3	3000	AD94250042F90B8D4A000052	58.4	6
4	3000	000000000000000000000402	71.2	10
5	3000	433333333333333333333333	64.8	11
6	3000	FFFFFFFFFFFFFFFF00000000	76.0	18
7	3000	999999999999999999999999	63.2	21
8	3000	AD8A20004531A1961F0000A0	61.6	8
9	3000	1005182006ABCDEF0002437D	61.6	10
10	3000	041008000000000000002222	54.4	11
11	3000	875AAAAAAAAAAAAAAAAAAAAA	72.8	12
12	3000	300833B2DDD9014035050000	72.0	21
13	3000	777777777777777777777777	71.2	12
14	3000	1005182006ABCDEF00024379	51.2	2
15	3000	AD8A2000453199901C00009F	80.8	12
16	3000	999999999999999999AAAAAA	67.2	10
17	3000	AD94250042F8F1934B00004F	64.8	10
18	3000	200000000000000000000134	75.2	12

Tag read = 49 | Rate = 117.5 Tag/s | CRC = 25.1 Tag/s

The EPC ID of the selected tag is shown.



123400020090319183650558

[✓ Target EPC] PWD 00000000

Kill Pwd	NO_CHANGE
Acc Pwd	NO_CHANGE
EPC	NO_CHANGE
TID	NO_CHANGE
USER	NO_CHANGE

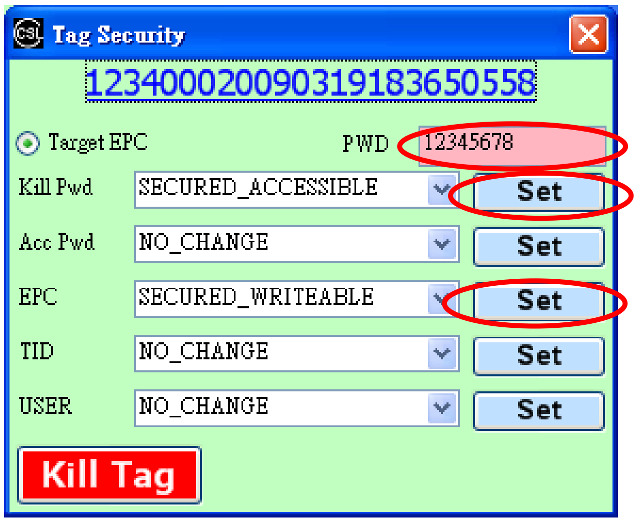
After the tag is selected, select the security you want to apply on each memory bank.
For Kill Password (Kill Pwd) and Access Password (Acc Pwd) banks:

- ACCESSIBLE: Unlock the bank – allow user to read and write it without access password
- ALWAYS_ACCESSIBLE: Permanently unlock the bank – allow user to read and write it forever without access password (cannot lock it again)
- SECURED_ACCESSIBLE: Lock the bank – cannot read and write the tag, need access password to unlock it
- ALWAYS_NOT_ACCESSIBLE: Permanently lock the bank – cannot read and write the tag forever (cannot unlock it again)
- NO_CHANGE: Keep the existing security state

For EPC and User Memory (USER) banks:

- WRITEABLE: Unlock the bank – allow user to write it without access password
- ALWAYS_WRITEABLE: Permanently unlock the bank – allow user to write it forever without access password (cannot lock it again)
- SECURED_WRITEABLE: Lock the bank – cannot write the tag, need access password to unlock it
- ALWAYS_NOT_WRITEABLE: Permanently lock the bank – cannot write the tag forever (cannot unlock it again)
- NO_CHANGE: Keep the existing security state

Enter the access password in "PWD" and click "Set" button to set the security setting.

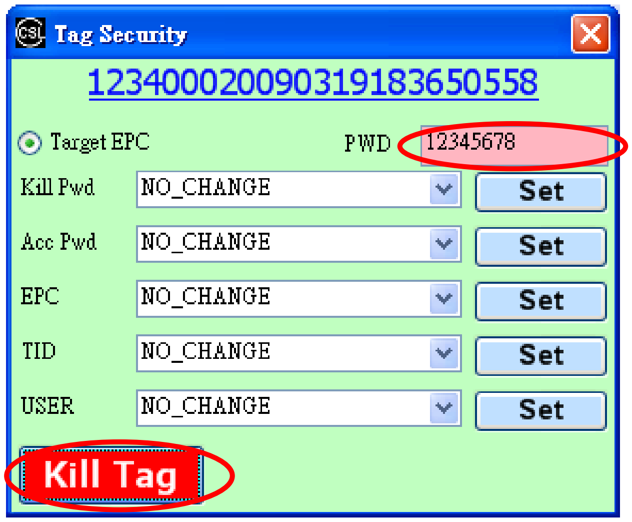


123400020090319183650558

Target EPC PWD 12345678

Kill Pwd	SECURED_ACCESSIBLE	Set
Acc Pwd	NO_CHANGE	Set
EPC	SECURED_WRITEABLE	Set
TID	NO_CHANGE	Set
USER	NO_CHANGE	Set

To kill the tag, enter the access password in "PWD" and click "Kill Tag button.



Tag Security

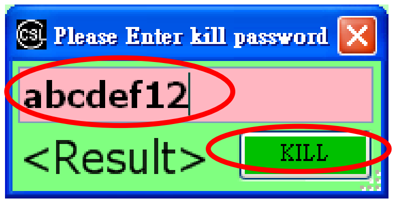
123400020090319183650558

Target EPC PWD 12345678

Kill Pwd	NO_CHANGE	Set
Acc Pwd	NO_CHANGE	Set
EPC	NO_CHANGE	Set
TID	NO_CHANGE	Set
USER	NO_CHANGE	Set

Kill Tag

After that, enter the kill password and click the "KILL" button to kill the tag



Please Enter kill password

abcdef12

<Result> KILL

9.1.9 Setup

The "Setup" page allows the user to configure the country setting, link profile and Gen2 parameter settings.

General Options

In "*General Options*", you could configure the reader's link profile, power and frequency settings.

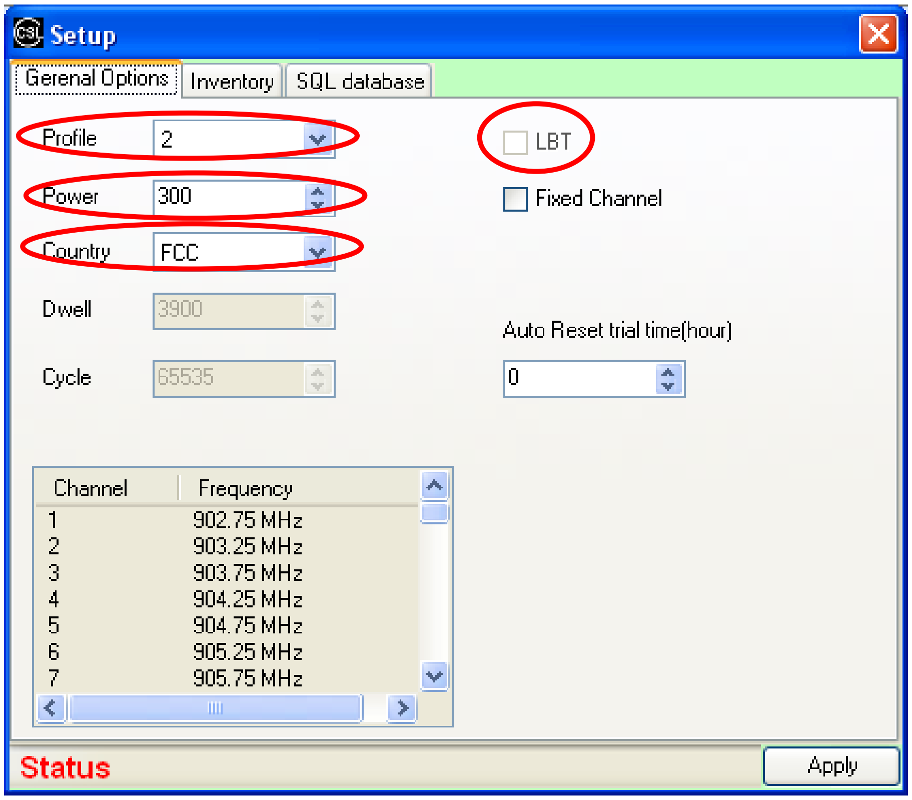
Profile – default setting is profile 2

Power – output power, display value = power x 10 (e.g. 300 = 30dBm)

Country – Select the corresponding country the reader operates in

LBT – Listen-before-talk option, available for CS203-3 (TELEC) readers only

Fixed Channel – Option for selecting fixed frequency channel



Profile 2

Power 300

Country FCC

Dwell 3900

Cycle 65535

LBT

Fixed Channel

Auto Reset trial time(hour) 0

Channel	Frequency
1	902.75 MHz
2	903.25 MHz
3	903.75 MHz
4	904.25 MHz
5	904.75 MHz
6	905.25 MHz
7	905.75 MHz

Inventory

In "*Inventory*", you could configure the reader's Gen2 parameter settings for custom inventory operation.

Operation: operate in continuous or non-continuous reading (CONTINUOUS or NONCONTINUOUS)

Selected: use select flag or not (ALL, ON, OFF, UNKNOWN)

Session: the session number of this reader (S0, S1, S2, S3, UNKNOWN), readers nearby should be configured in different session number

Target: the target flag for this reader (A, B, UNKNOWN)

Algorithm: the inventory algorithm (FixedQ, DynamicQ, DynamicQ_Adjust, DynamicQ_Thresh)

StartQValue: the starting Q-value for Dynamic Q algorithms

Retry: the number of retry for each Q-value inventory

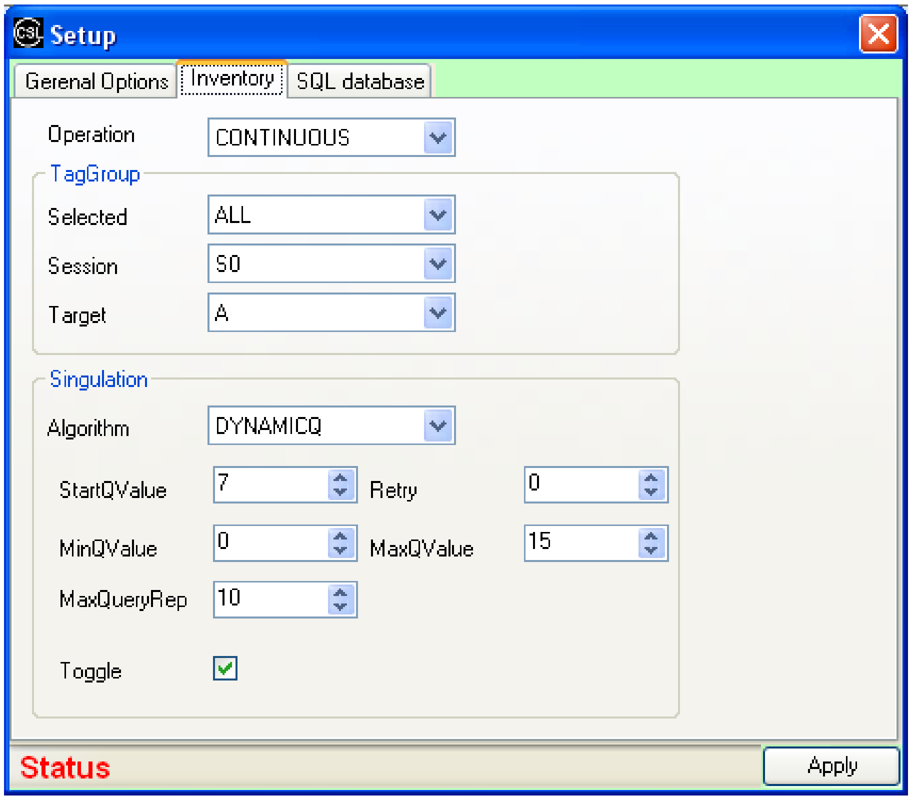
MinQValue: the minimum Q-value allowed for Dynamic Q algorithms

MaxQValue: the maximum Q-value allowed for Dynamic Q algorithms

MaxQueryRep: the maximum number of QueryRep in inventory

Toggle: tick this box for allow toggle of flag during inventory

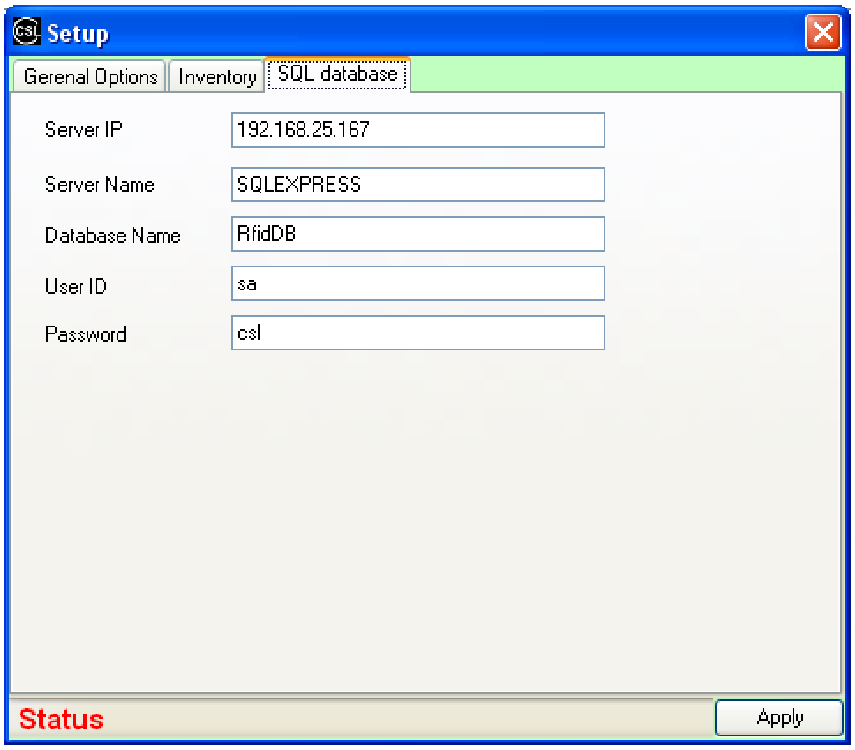
For detail explanation of these Gen2 parameters, please refer to *Chapter 10 Gen2 Parameters Descriptions* or the specification of EPC Class 1 Gen 2 Protocol.



Operation: CONTINUOUS
Selected: ALL
Session: S0
Target: A
Algorithm: DYNAMICQ
StartQValue: 7
Retry: 0
MinQValue: 0
MaxQValue: 15
MaxQueryRep: 10
Toggle: ✓

SQL database

In "SQL database", you could configure the SQL server information for saving the tag data.



Server IP 192.168.25.167

Server Name SQLEXPRESS

Database Name RfidDB

User ID sa

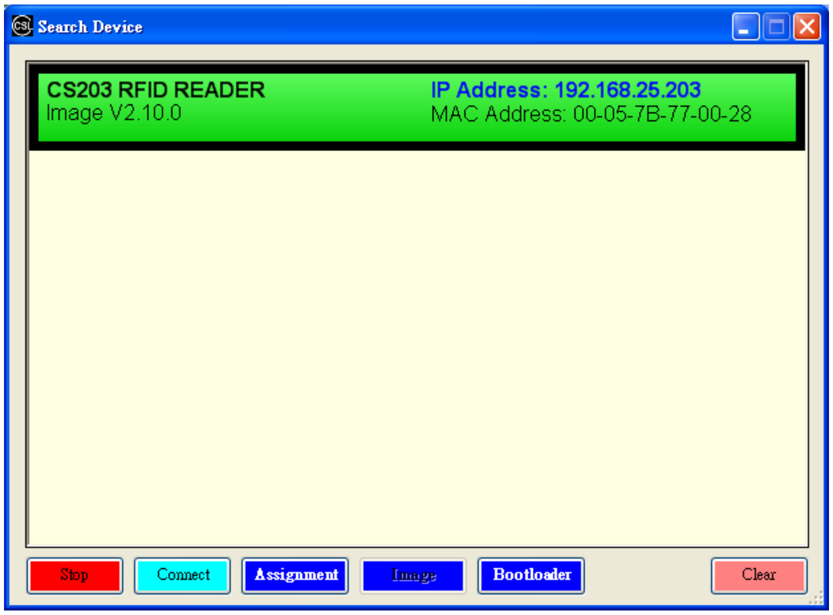
Password csl

9.1.10 Firmware Upgrade

The Callback-based API DEMO program could also be used to perform firmware upgrade on CS203 readers. There are 2 types of firmware: bootloader and application image.

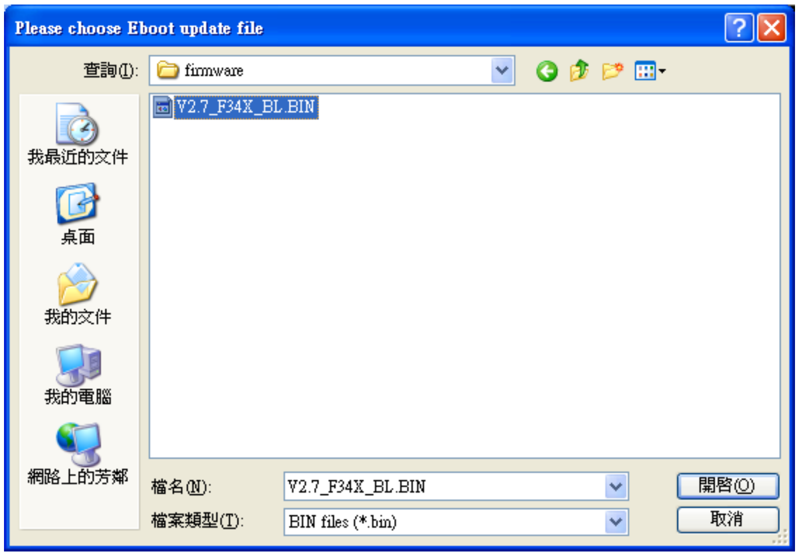
i) Bootloader Upgrade

1. Boot up CS203 reader
2. Run Callback-based API DEMO program and click "Start" to search the reader
3. Select the reader on the list and click "Bootloader" button



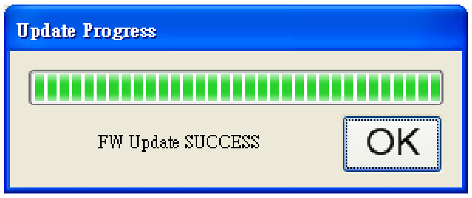
CS203 RFID READER
Image V2.10.0
IP Address: 192.168.25.203
MAC Address: 00-05-7B-77-00-28

4. Select the bootloader image file (*.bin) to update



V2.7_F34X_BL.BIN
檔名(N): V2.7_F34X_BL.BIN
檔案類型(T): BIN files (*.bin)

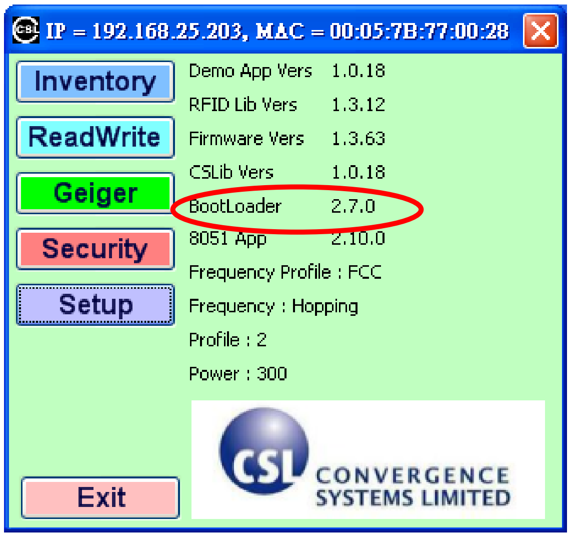
5. Wait until the update finish and then click "OK" button to continue



FW Update SUCCESS

Remark: If "Update successful" doesn't appear after 10 seconds, please restart CS203 and the DemoApp program and repeat the bootloader update process.

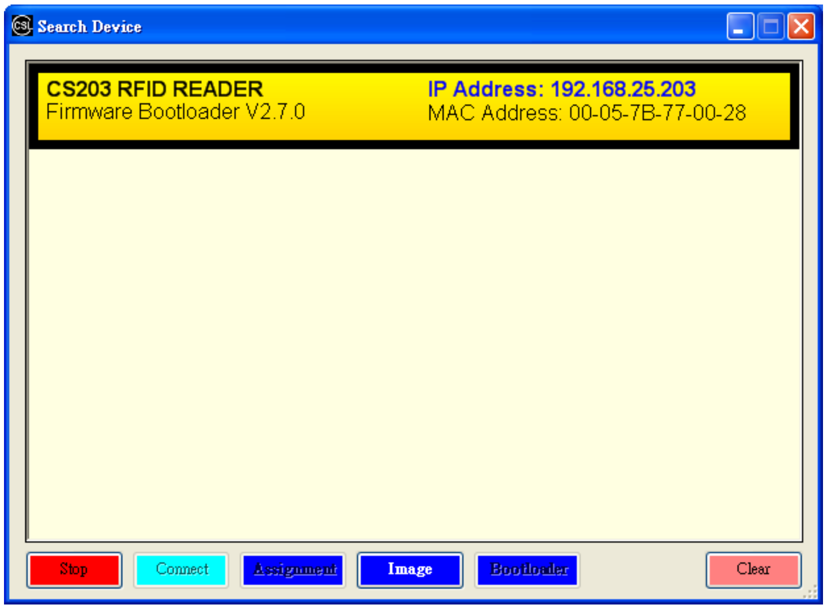
6. After bootloader upgrade process completed, connect to the reader to check if the new bootloader version is updated.


IP = 192.168.25.203, MAC = 00:05:7B:77:00:28
Inventory
ReadWrite
Geiger
Security
Setup
Exit
CSL CONVERGENCE SYSTEMS LIMITED

```
Demo App Vers    1.0.18
RFID Lib Vers    1.3.12
Firmware Vers    1.3.63
CSLib Vers       1.0.18
BootLoader       2.7.0
8051 App         2.10.0
Frequency Profile : FCC
Frequency : Hopping
Profile : 2
Power : 300
```

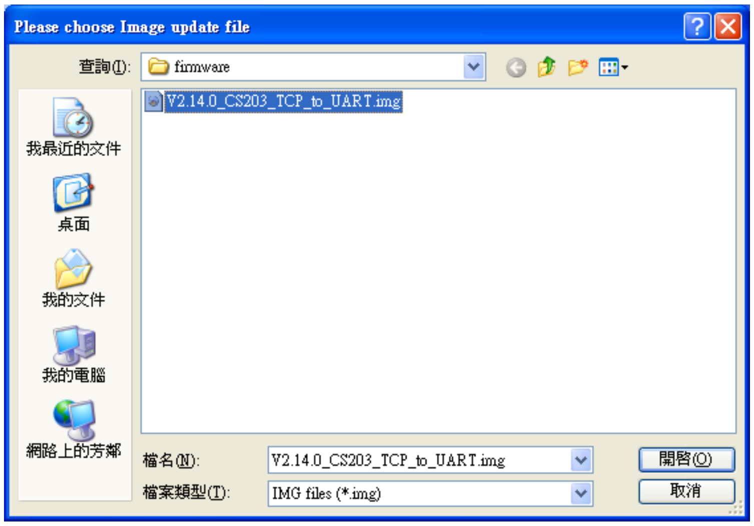
ii) <u>Application Image Upgrade</u>

1. Boot up the reader to "Bootloader Mode" by:
   i. Push the reset button and hold it
   ii. Power up the CS203
   iii. Wait for 5 seconds and then release the reset button
   iv. The "RFID" LED on the back of CS203 should be flashing every second
2. Run the Callback-based API Demo program and click the "Start" button to search the reader. If the reader is successfully boot in Bootloader mode, it will be displayed in the list with yellow background color.



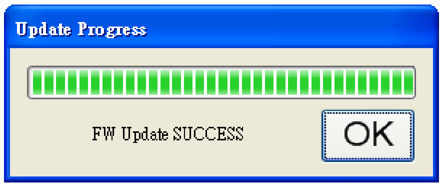
CS203 RFID READER
Firmware Bootloader V2.7.0
IP Address: 192.168.25.203
MAC Address: 00-05-7B-77-00-28

3. Select the reader in the list and click "Image" button.
4. Select the application image file (*.img) to update



firmware
V2.14.0_CS203_TCP_to_UART.img
檔名(N): V2.14.0_CS203_TCP_to_UART.img
檔案類型(T): IMG files (*.img)

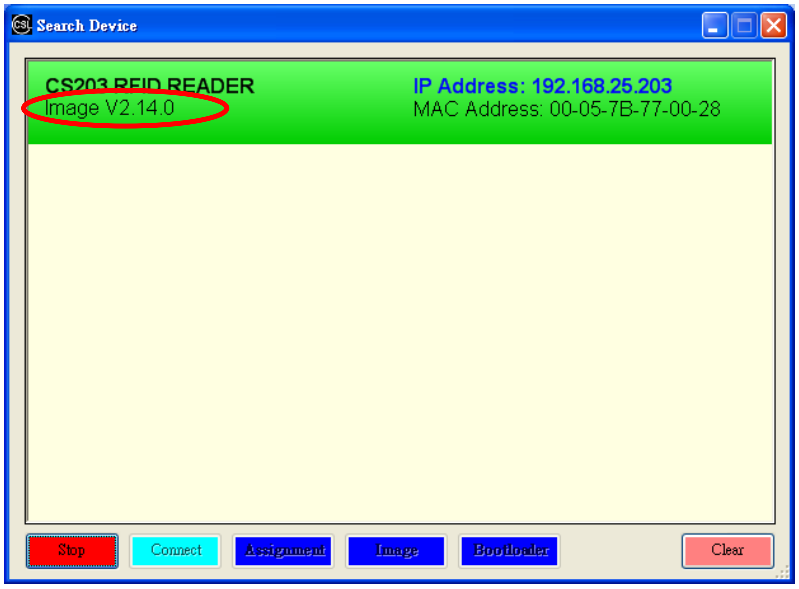
5. Wait until the update finish and then click "OK" button to continue.



FW Update SUCCESS

Remark: If "FW Update SUCCESS:" doesn't appear after 20 seconds, please restart CS203 in Bootloader mode and the DemoApp program and repeat the image update process.

6. After the image upgrade process completed, search for the reader on the Callback-based API Demo program again and check if the new image version is updated.



CS203 RFID READER
Image V2.14.0
IP Address: 192.168.25.203
MAC Address: 00-05-7B-77-00-28

10 Gen 2 Parameters Descriptions

The detail description about the Gen 2 parameters can be referred to the specification of EPC Class 1 Gen 2 Protocol.

10.1 Link Profile

Link Profile: Different modulation profile can be selected by the user for different situation.
In CS203-3 (Japan) and CS203-1 (ETSI) readers, only profiles 0, 2, 3 and 5 are selectable.
The default setting is profile 2. It is also the recommended setting for most of the common applications.

The detail parameter settings of the available profiles are as follows:

Profile	**Reader –to-Tag Modulation**	**Tari (us)**	**Pulse Width (us)**	**Tag Encoding**	**Tag-to-Reader Link Freq. (kHz)**	**Data rate (kbps)**
0	DSB-ASK	25	12.5	FM0	40	40
1	DSB-ASK	12.5	6.25	M=2	160	80
2	**PR-ASK**	**25**	**12.5**	**M=4**	**250**	**62.5**
3	PR-ASK	25	12.5	M=4	300	75
4	DSB-ASK	6.25	3.13	FM0	400	400
5	PR-ASK	25	12.5	M=2	250	125

10.2 Q-Value

Q-value is a parameter that a reader uses to regulate the probability of Tag response. A reader commands Tags in an inventory round to load a Q-bit random (pseudo-random) number into their slot counter; the reader may also command Tags to decrement their slot counter. Tags reply when the value in their slot counter (i.e. their slot) is zero. Q is an integer in the range (0, 15); the corresponding Tag-response probabilities range from $2^0 = 1$ to $2^{-15} = 0.000031$.

It is usually recommended to set a Q-value that have the corresponding number of slots larger than the exact maximum number of Tags to be read by the reader. For example, if there are 40 tags to be read by the reader, the Q-value is configured to 6, in which $2^Q = 2^6 = 64 > 40$.

In CS203, it consists of four main inventory algorithm (one fixed Q and three variable Q). The variable Q algorithms differ in their mechanism for adjusting the Q values at the end of each round. It also provides routines for the four tag access functions (read, write, kill and lock).

1) Fixed Q (Generic) Algorithm

• Fixed Q value
• Basis for all inventory algorithms.
• Optionally executes rounds until no tags are read.
• Optionally retries a rounds "n" times.
• Optionally flips A/B flag at end of round

This algorithm runs all inventory rounds with a single Q value. In this algorithm an inventory cycle consists of one or more rounds, each of which will attempt to read every slot. The number of slots to search is given by 2Q. For example, a Q of 7 will cause the algorithm to search 128 slots on each round. One word of caution, if the time it takes to run the round is greater that the frequency hop time (and the session is 0) or antenna dwell time, the round will never complete.

2) Dynamic Q Algorithm (1)

• Q adjusts up or down at the end of each round.
• Executes repeated rounds until no tags are read when Q = Q minimum.
• Uses Qstart,Qmax and Qmin parameters to control the range of Q.
• MaxReps to limit time spent at each Q value.
• HighThres and LowThres to control how Q will adjust.

In algorithm 1, the value of Q is dynamically adjusted based on the periodic evaluation of the relative frequency of RN16 timeouts vs EPC timeouts.
Each round is comprised of a Query and up to MaxReps queryReps.
The value of Q for the subsequent round is determined by the results of the current round. If the number of RN16 timeouts is greater than the number of EPC timeouts multiplied by thresHi, Q is decremented (presumed empty slots outnumber presumed collisions). If the number of RN16 timeouts is less than the number of EPC time outs times thresLo, Q is incremented (presumed collisions outnumber presumed empty slots). If the number of RN16 time outs falls between those two values, Q remains unchanged.
An inventory cycle is comprised of one or more inventory rounds, and is terminated when a round is executed with Q = 0 and no tags read.

3) Dynamic Q Algorithm (2) – Dynamic Q Adjust

• Almost identical to algorithm 1.
• Same control interface as algorithm 1.
• Uses QueryAdjust command to modify Q value

Algorithm 2 is identical to algorithm 1 with the sole exception that a queryAdjust command is used to adjust the value of Q rather than a query command.
Read rate performance is increased relative to algorithm 1 because a) the query Adjust command is shorter, and b) new rounds are not initiated each time the value of Q changes, reducing the frequency of duplicate tag reads in the course of an inventory cycle.

Note though that algorithm 1 may deliver superior performance when reading small fast moving (or changing) tag populations due to the increased frequency with which query Commands are issued.

4) Dynamic Q Algorithm (3) – Dynamic Q Thresh

• New Q adjustment algorithm.
• Uses Qstart,Qmax and Qmin parameters to control the range of Q.
• QueryReps are not limited to a maximum number on a round.
• Single threshold multiplier used to control Q adjustment.
• QueryAdjust command used to modify Q value as in Algorithm 2.

In algorithm 3, the value of Q is adjusted based on the continuous evaluation of the relative frequency of RN16 timeouts vs EPC timeouts.
An inventory cycle consists of a single round initiated by a Query command.
Following the query command, up to ((2^^Q)-1) queryRep commands are issued.
If in the course of operation the number RN16 timeouts exceeds the adjusted number of EPC timeouts by a calculated threshold, the value of Q is decremented (presumed empty slots outnumber presumed collisions). If the adjusted number of EPC timeouts exceeds the number of RN16 timeouts by a calculated threshold, the value of Q is incremented (presumed collisions outnumber presumed empty slots). While the relative number of RN16 time outs vs the adjusted number of EPC time outs falls within the threshold, Q is unchanged.
When the value of Q changes, or if all slots under the current Q value have been inventoried, the slot counters of the participating tag population is refreshed using a queryAdjust command.
The calculated threshold equals the current value of Q times a multiplier (set by default to 1).
The EPC timeout count is adjusted by Rtot, the ratio of (EPC timeout / RN16 timeout).
An inventory cycle is terminated when all slots have been checked with Q = Qmin and no tags have been read.
Read rate performance is increased relative to algorithm 2 because a) Q remains unchanged while well matched to the population, b) Q value is changed more quickly when it is not well matched, and c) on the average, fewer queryAdjust commands are issued.

10.3 Session

An inventory process comprising a reader and an associated Tag population. A reader chooses one of four sessions and inventories Tags within that session. The reader and associated Tag population operate in one and only one session for the duration of an inventory round. For each session, Tags maintain a corresponding inventoried flag. Sessions allow Tags to keep track of their inventoried status separately for each of four possible time-interleaved inventory processes, using an independent inventoried flag for each process.

10.4 Inventories flag (Flag)

A flag that indicates whether a Tag may respond to a reader. Tags maintain a separate inventoried flag for each of four sessions; each flag has symmetric A and B values. Within any given session, reader typically inventory Tags from A to B followed by a re-inventory of Tags from B back to A (or vice versa).

11 Appendix: Frequency Channels List

Region	Frequency Range (MHz)	# of channels	Channels (MHz)
Australia (AU)	920 – 926	10	920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25, 923. 75, 924. 25, 924. 75, 925. 25,
Brazil 1 (BR1)	915 – 928	24	915. 75, 916. 25, 916. 75, 917. 25, 917. 75, 918. 25, 918. 75, 919. 25, 919. 75, 920. 25, 920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25, 923. 75, 924. 25, 924. 75, 925. 25, 925. 75, 926. 25, 926. 75, 927. 25,
Brazil 2 (BR2)	902 – 907 915 – 928	33	902. 75, 903. 25, 903. 75, 904. 25, 904. 75, 905. 25, 905. 75, 906. 25, 906. 75, 915. 75, 916. 25, 916. 75, 917. 25, 917. 75, 918. 25, 918. 75, 919. 25, 919. 75, 920. 25, 920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25, 923. 75, 924. 25, 924. 75, 925. 25, 925. 75, 926. 25, 926. 75, 927. 25,
China (CN)	920 – 925	10	920. 625, 920. 875, 921. 125, 921. 375, 921. 625, 921. 875, 922. 125, 922. 375, 922. 625, 922. 875, 923. 125, 923. 375, 923. 625, 923. 875, 924. 125, 924. 375,
ETSI, G800	865 – 868	4	865. 70, 866. 30, 866. 90, 867. 50,
Hong Kong (HK) Singapore (SG)	920 – 925	8	920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25, 923. 75, 924. 25,
India (IN)	865 – 868	3	865. 70, 866. 30, 866. 90,
Japan (JP)	952 – 954	7	952. 40, 952. 60, 952. 80, 953. 00, 953. 20, 953. 40,

			953. 60,
Korea (KR)	910 – 914	19	910. 20, 910. 40, 910. 60, 910. 80, 911. 00, 911. 20, 911. 40, 911. 60, 911. 80, 912. 00, 912. 20, 912. 40, 912. 60, 912. 80, 913. 00, 913. 20, 913. 40, 913. 60, 913. 80,
Malaysia (MY)	919 – 924	8	919. 75, 920. 25, 920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25,
South Africa (ZA)	915 – 919	16	915. 7, 915. 9, 916. 1, 916. 3, 916. 5, 916. 7, 916. 9, 917. 1, 917. 3, 917. 5, 917. 7, 917. 9, 918. 1, 918. 3, 918. 5, 918. 7,
Taiwan (TW)	922 – 928	12	922. 25, 922. 75, 923. 25, 923. 75, 924. 25, 924. 75, 925. 25, 925. 75, 926. 25, 926. 75, 927. 25, 927. 75,
USA (FCC)	902 – 928	50	902. 75, 903. 25, 903. 75, 904. 25, 904. 75, 905. 25, 905. 75, 906. 25, 906. 75, 907. 25, 907. 75, 908. 25, 908. 75, 909. 25, 909. 75, 910. 25, 910. 75, 911. 25, 911. 75, 912. 25, 912. 75, 913. 25, 913. 75, 914. 25, 914. 75, 915. 25, 915. 75, 916. 25, 916. 75, 917. 25, 917. 75, 918. 25, 918. 75, 919. 25, 919. 75, 920. 25, 920. 75, 921. 25, 921. 75, 922. 25, 922. 75, 923. 25, 923. 75, 924. 25, 924. 75, 925. 25, 925. 75, 926. 25, 926. 75, 927. 25