Report No.14017634 001

Appendix 9:

User Manual

FCCID: UB4CS101C1GEN2

(Total: 213 pages, include this page)

CSL CS101-2 EPC Class 1 Gen 2 RFID Handheld Reader User's Manual

Version 1.0

CSL: The One-Stop-Shop for RFID Solutions

1 Content

2 FCC Statement

FCC NOTICE: To comply with FCC part 15 rules in the United States, the system must be professionally installed to ensure compliance with the Part 15 certification. It is the responsibility of the operator and professional installer to ensure that only certified systems are deployed in the United States. The use of the system in any other combination (such as co-located antennas transmitting the same information) is expressly forbidden.

3 Introduction

3.1 CS101-2 Handheld RFID Reader

The CS101-2 handheld RFID reader is a ruggedized reader designed from the drawing board to have **extremely long read range and high read rate** – in that it is designed to replace fixed reader in many applications where fixed reader is a non-portable and therefore non-viable option. In fact it is nicknamed “Fixed Reader in Your Hand”. CS101-2 is a product that arises out of popular requests for applications such as:

1. Dock Door applications where the handheld reader is used to complement fixed reader when tags are not 100% read by the fixed reader.
2. Loading Bay applications where the fixed reader is not allowed because there is no place to put a permanent reader stand.
3. Warehouse applications where the handheld is used to do long read range inventory of all the shelves – apparently not a good idea to use a fixed reader and move it around up and down.
4. Special applications where long read range is a MUST because the operator does not want to go near the tagged item, example police inspecting the electronic license plate of a suspect vehicle with a suspicious driver inside.
5. Retail shop inventory applications where high read rate is most useful – workers can go home earlier!!

3.2 How to Use this Manual

This manual provides a comprehensive introduction to the CSL CS101-2 EPC Class1 Gen 2 handheld RFID reader (chapter 2), Installation Guide (chapter 3), Quick Start Guide (chapter 4), Applications Interface (chapter 5), CSL Demo Programs (chapter 6), Software Development Environment (chapter 7), PC Side Demo Program (chapter 8) and Usage Tips for CS101-2 (chapter 9).
Some other information such as RFID Cook Book (chapter 10), RFID Best Practices (chapter 11) and RFID Use Cases (chapter 12) are also provided for reference.

3.3 Product Package

3.3.1 Basic Package Content

The reader package contains:

- Handheld reader
- Charger with power adapter and country specific power cord
- Batteries – 2 pieces
- Wrist strap
- Shoulder strap
- Sample tags
- User Manual (in CD format)

3.3.2 Unpacking Instructions

Unpacking of the reader is very simple. Just open up the box and take out the content to a table. **The charger should be connected and the 2 batteries charged for a minimum of 10 hours before first use.**

3.4 Product Specification

Figure 3-1 CS101-2 Reader

Features:

- ISO 18000-6C and EPCglobal Class 1 Gen 2 UHF RFID protocol compliant including dense reader mode
- Ultra long read range – peak at 5 to 7 meters for Banjo tag
- Ultra high read rate – peak at 200 tags per second
- Sophisticated data handling for efficient management of large streams of tag data.
- Highly configurable buffering and tag filtering modes to eliminate the redundant tag data so as to reduce wireless LAN traffic and server loading
- 400 kbps tag-to-reader data rate profile
- Robust performance in dense-reader environments
- Excellent in transmit and receive mode – generates a different combination of unique reader-to-tag command rate, tag-to-reader backscatter rate, modulation format, and backscatter type
- Configurable parameters offer maximum throughput and optimal performance
- Supports all Gen 2 commands, including write, lock and kill

Specifications:

Physical Characteristics:	Length: 20 cm; Width: 12.5 cm; Height: 22.5 cm; Weight: 1.2 Kg
Environment:	Operating Temp: 0^0C to 50^0C Storage Temp: -40^0C to 85^0C Humidity: 5% to 95% non-condensing Enclosure: IP-63
Antenna:	Linear with excellent polarization diversity
Power:	14.8 Volt 1400 mAh Lithium Polymer battery
RFID Frequency Ranges:	902-928 MHz band
Interfaces	Wi Fi 802.11b/g with WPA Configurable to use fixed IP address or DHCP USB RS-232 Maximum 2GB SD card storage
Operating System:	WinCE Profession 5.0
Maximum Tag Read Rate:	200 tag/sec.
Maximum Speed of Tag:	660 ft/min
Accessories:	Charger, batteries, wrist strap, shoulder strap
Order Code:	CS101-2
Restrictions on Use:	Approvals, features and parameters may vary depending on country legislation and may change without notice

4 Installation

4.1 Devices

4.1.1 Reader

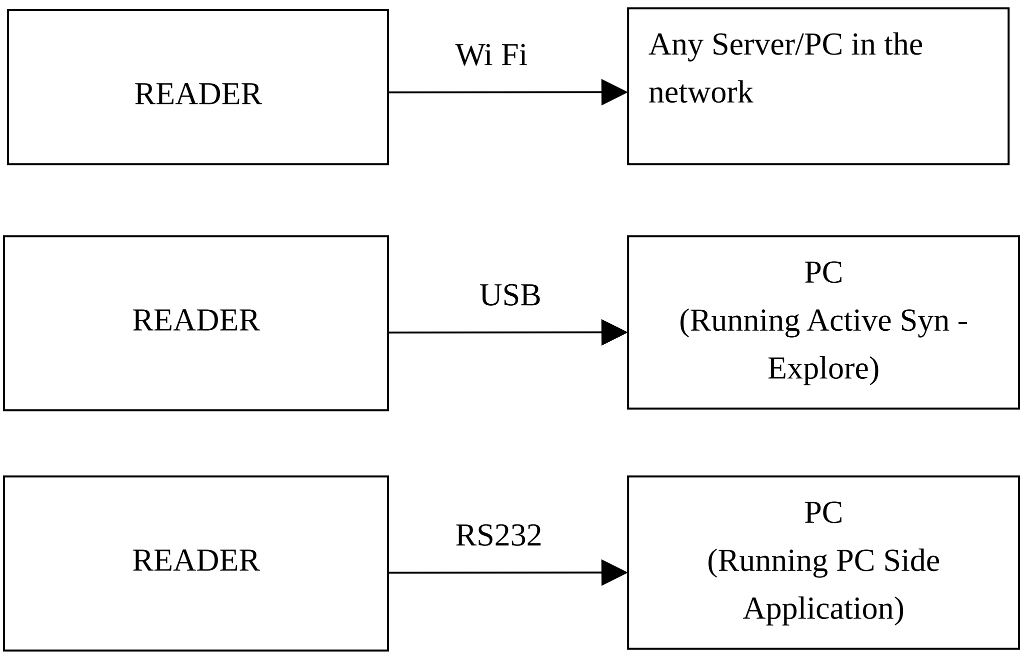
The CSL CS101-2 handheld RFID Reader is an EPCglobal Class 1 Gen 2 handheld reader product.

Figure 4-1 CS101-2 Reader Front View

Figure 4-2 CS101-2 Reader Side View

Figure 4-3 CS101-2 Reader Plan View

The reader is connected to the network via Wi Fi. The reader can have a static IP address or can obtain an IP address using DHCP. Normally, a static IP address is more convenient to use because it does not change when the reader reboots, but the user has to make sure there is no collision with other network devices in the network. If the reader is configured to be DHCP, then a separate discovery program that runs on the PC side can help the user find all readers in the same local area network.



4.1.2 Charger

Figure 4-4 CS101-2 Charger with AC Adaptor

4.2 Power Up Sequence

The reader can be turned on to run RFID operation in a most simple manner:

1. Insert battery into the handle of the handheld reader with the metal contact inward. Also, make sure it is in the correct direction in terms of front and back. If the front-back direction is reversed, the battery cannot go in – in that case do not force it in, just reverse the battery and it should slide in effortlessly. Then use the cap to hold the battery firmly.
2. Press the power button on the upper right corner of the keypad continuously until LCD screen display appears.
3. Wait till WinCE screen shows up.
4. On the WinCE screen, there is an application called CS101. Double click it to start the application.
5. A screen will show up asking for ID and password. For ID, input **root**, for password, input **root.** (You can change that later, either setting it to NO ID/PASSWORD mode so that the software will not ask for ID and password, or change to ID and password to whatever name you want)
6. After that, the screen will enter the application selection page and you can start reading and writing tags, inventory of tags, search of tags, etc.

4.3 Usage Recommendation

4.3.1 Strap: Wrist Strap and Shoulder Strap

The wrist strap and shoulder strap should be attached to the handheld reader to allow additional weight support during use.

4.3.2 IO Connection

The IO connector consists of one USB connector (mini-USB) and a RS232 Serial connector (Firewire) with dedicated cable that come with the reader.

Figure 4-5 IO Interface

4.4 Verification and Validation

The reader comes with standard demo application, double click the icon “CS101 Demo App” to start.

Figure 4-6 WinCE Screen

The main menu will show on screen, it includes two pages, click “More…” to see the remaining items on next page.

Figure 4-7 Main Menu page 1

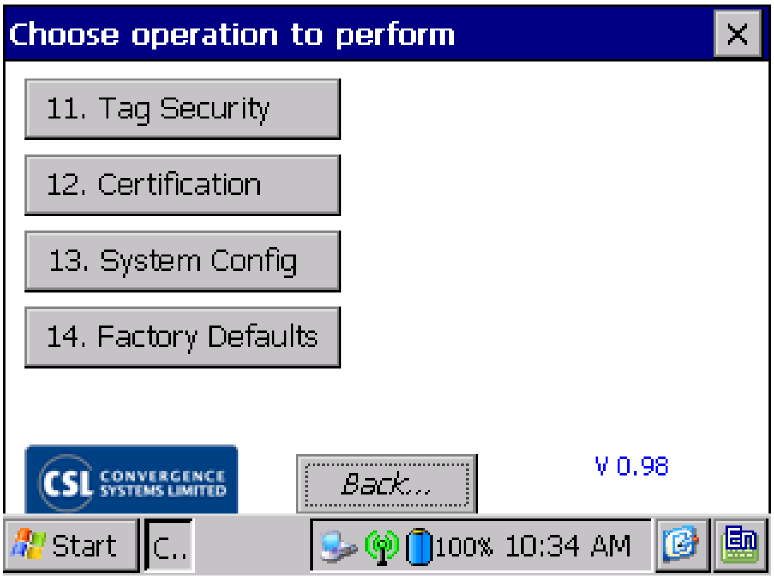


Figure 4-8 Main Menu page 2

Please double check the current reader configuration in the "Reader Configuration Brief" as shown in figure 4-10. Whenever the reader is stayed in the idle state (not during reading or writing the tags), you can press the function key "F1" to display the reader configuration information, it is a fast and convenient way to review of the reader setting e.g. current IP address, country code and antenna power output.

Figure 4-9 Keyboard

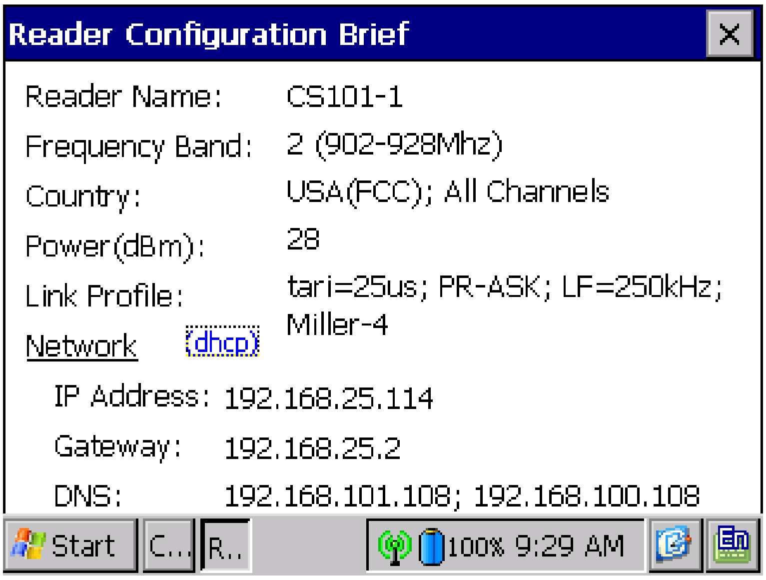


Figure 4-10 Reader Configuration Brief

To read tags, click the “Tag Read” button and then click “Scan First”:

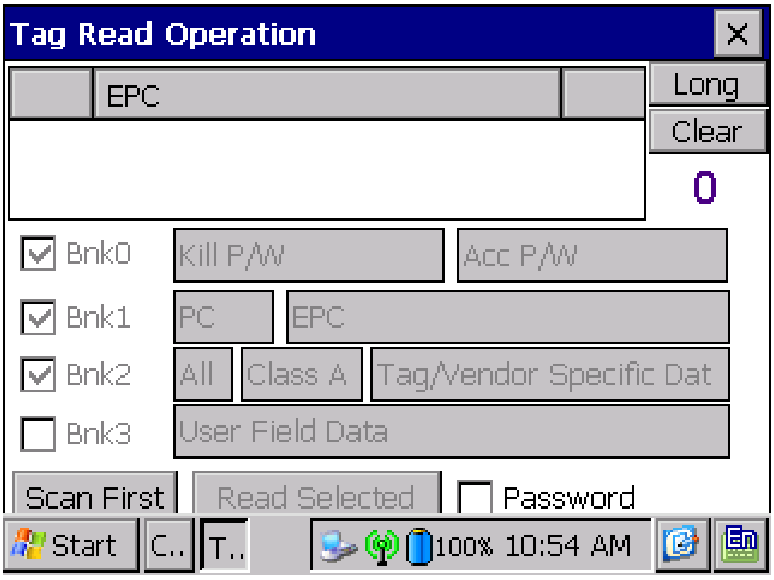


Figure 4-11 Tag Read Operation

Take the sample tags and put them in front of the handheld reader, all the tag's EPC within the read range of the reader will be read by handheld reader:

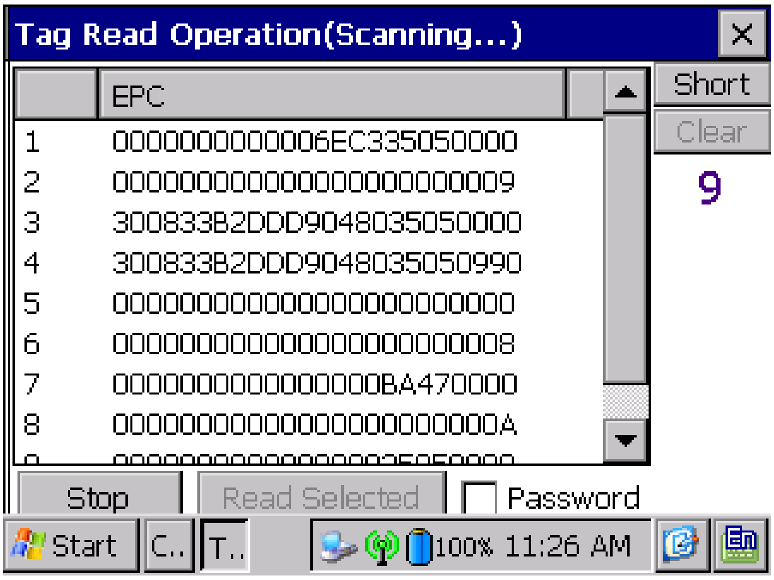


Figure 4-12 Tag Read – Listing

4.5 Cautions

The default IP address of handheld reader is printed on the reader label. To change this IP address, please go to System Configuration page of the demo application to do that.

5 Quick Start

5.1 Login

- Press the power button to power up the reader.
- To login, input the “User Name” and “Password”, then click the “Login” button. The default administrator login name and password are as follows:

Login: root
Password: root

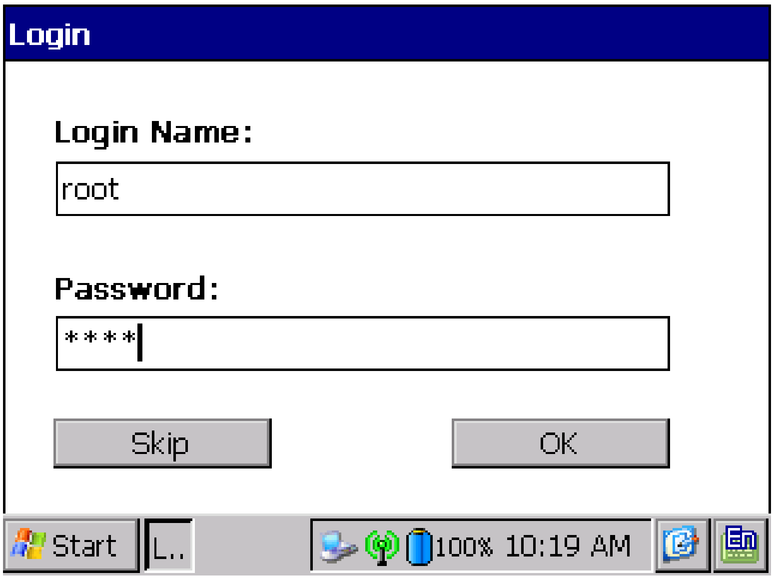


Figure 5-1 User Login Screen

5.2 System Configuration

System configuration allows user to set basic properties of the handheld reader, such as identity of the reader (reader name), authentication requirement of the application (ID and password), etc.

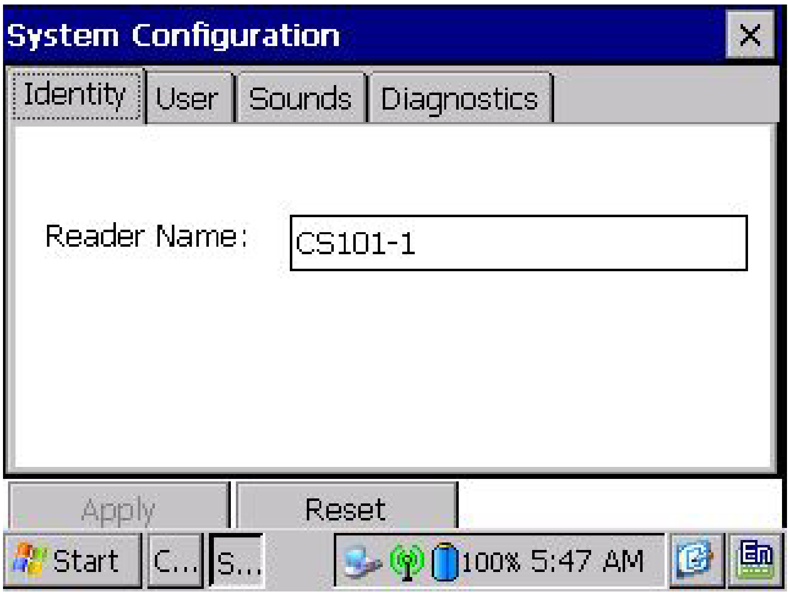


Figure 5-2 System Configuration Screen

5.3 Setup RFID Configuration

User can set up operation profile of the RFID reading and writing operation by going to the RFID Configuration screen.

Select operation profile:

- Please open page “Link Profile” as shown in Figure 5-3. You can reach the page by clicking “RFID Config -> Link Profile”.
- Select correct values and then click “Apply” button.

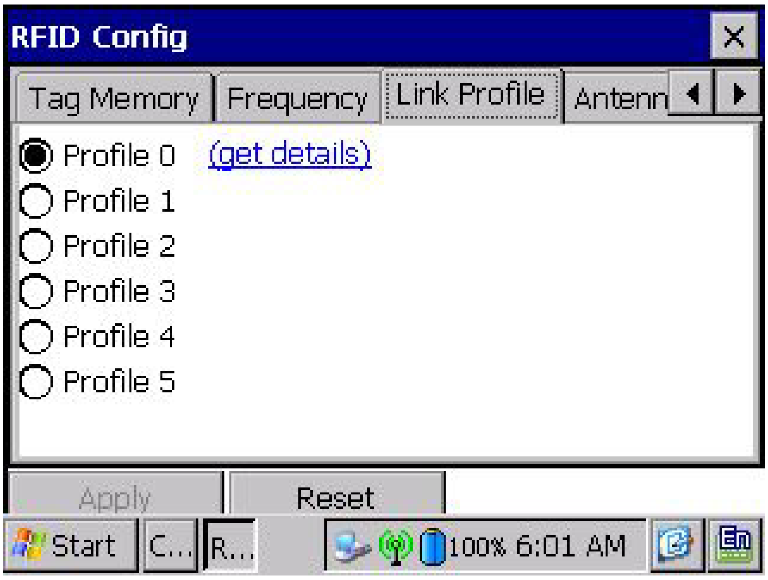


Figure 5-3 Reader Configuration Screen

5.4 Reader Configuration Brief

You can check the current reader configuration in the "Reader Configuration Brief". Whenever the reader is stayed in the idle state (not during reading or writing the tags), you can press the function key "F1" to display the reader configuration information, it is convenient to review the reader settings e.g. current IP address, country code and antenna output power.

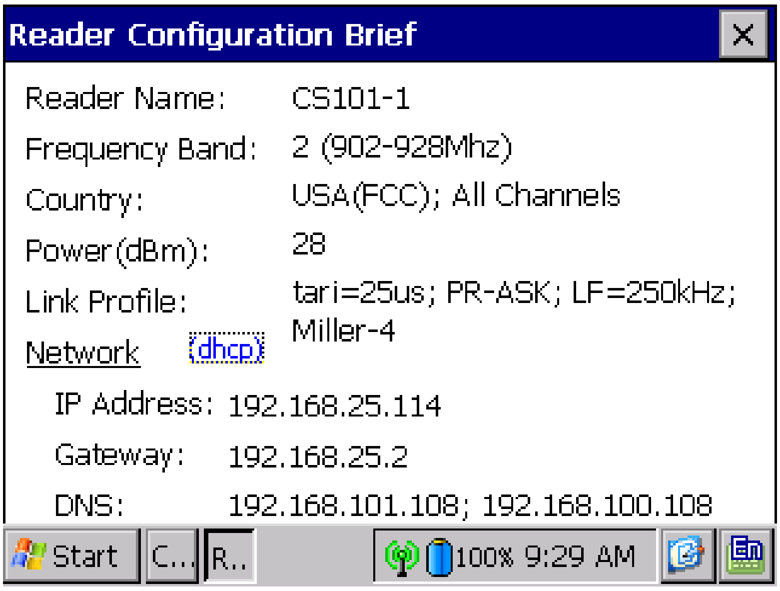


Figure 5-4 Reader Configuration Brief

If you want to check further information of the DHCP server such as IP address or the lease time, your can click the word "dhcp" on the screen.

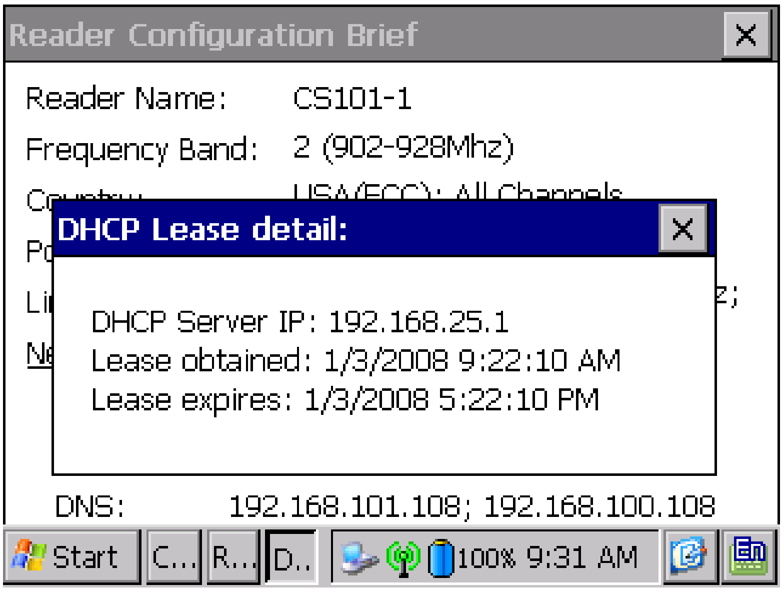


Figure 5-5 Reader Configuration Brief - DHCP Lease Details

5.5 Tag Inventory

Tag inventory provides an efficient way to read and count all tags at a time. In tag inventory, a duplicate filter is implemented to filter out the tags with same EPC ID, hence the tags with same EPC ID will not be displayed on the screen more than once. The function is most useful for warehouse inventory or whatever kind of inventory where you want to know the unique tags in the environment.

Besides the EPC ID, you can also read the ambient temperature and internal temperature of the handheld reader.

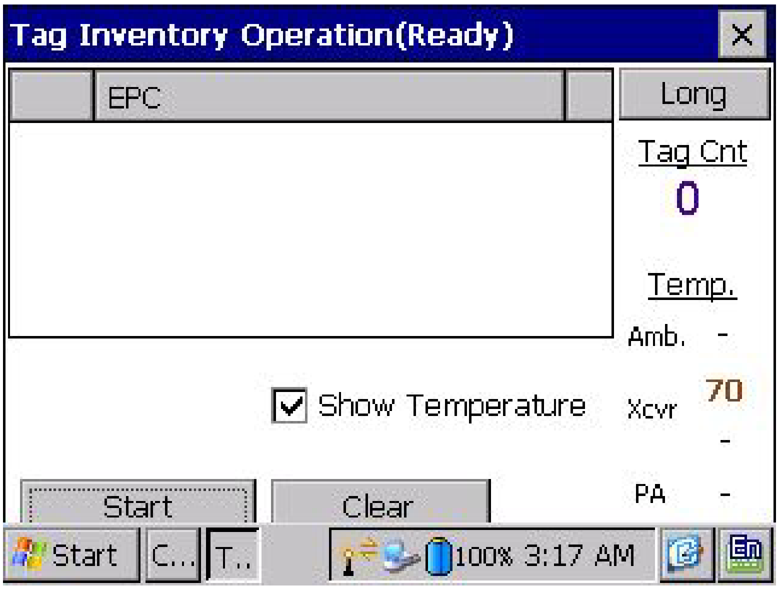


Figure 5-6 Tag Inventory Operation

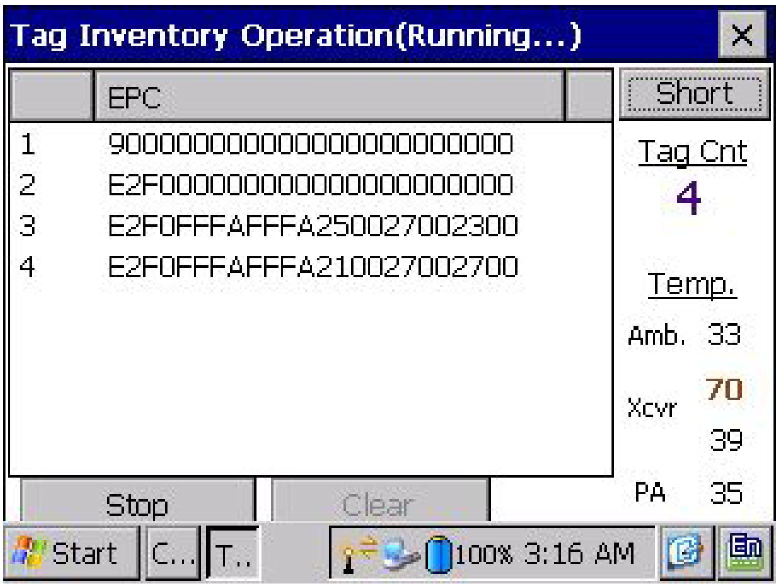


Figure 5-7 Tag Inventory Operation – Reading

5.6 Tag Ranging

The Tag Ranging lets user read tags with RSSI (RF Signal Strength Indicator), the RSSI value will keep changing when the handheld reader moves to and from the tags (RSSI will increase when the handheld reader is moved close to the tags). It is useful for the user to identify a tag when it starts falling into the read range of the reader. This function is not really for normal operation use but is meant mainly for system integrator to select appropriate tag designs in the actual application environment. In other words, for each customer he works with, the asset to be tagged may be different, and the customer business process may be different. To satisfy that the exact model of tag that can fulfill the requirements may be different. In that case the system integrator needs to put different types of tags onto the asset in the real environment and use this program to check if indeed the tag can be read at the position of the operator.

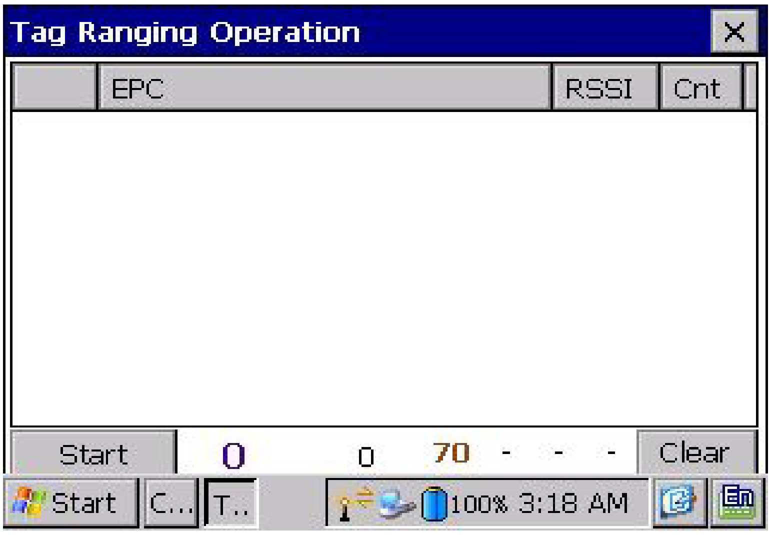


Figure 5-8: Tag Ranging Main

Figure 5-9 Tag Ranging Result

6 Demo Application

6.1 Introduction

The WinCE screen contains a short cut called CSL 101 Demo App, as shown in Figure 6-1. Please double click that short cut to start the application. This application is kind of like the unit test program that will allow the user to quickly experiment with the various functionalities of the reader (physical tag read and write and higher layer applications) and, because source code is provided, allows the user to quickly develop their own programs.

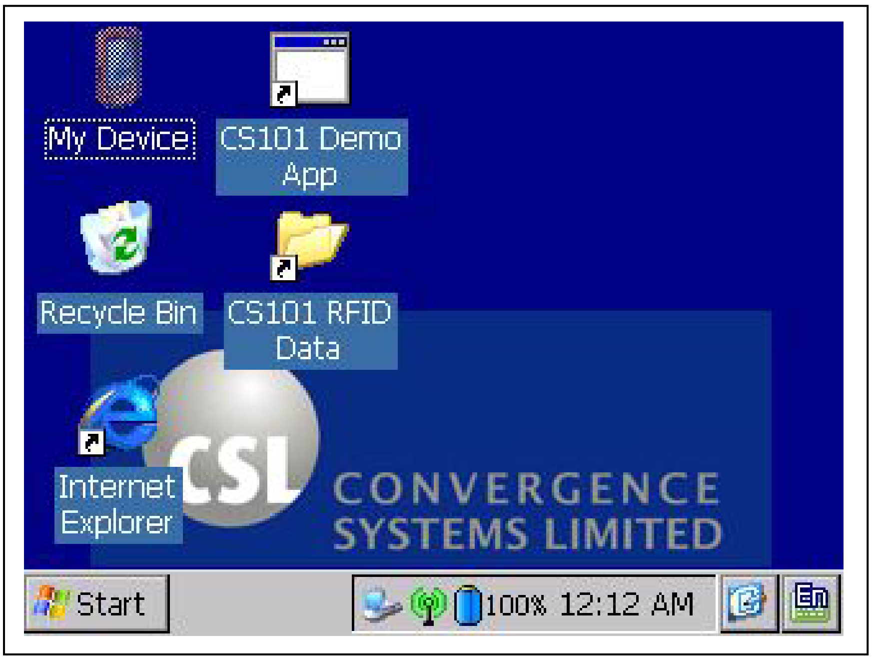


Figure 6-1 WinCE Screen

6.2 Splash Screen

The splash screen will display, wait until the application start up.

Figure 6-2 Splash Screen

6.3 ID and Password Page

The ID and Password page, as shown in Figure 6-3, allows controlled access to this application. Note: This page is configurable to be displayed, it is set to be turned off by default.

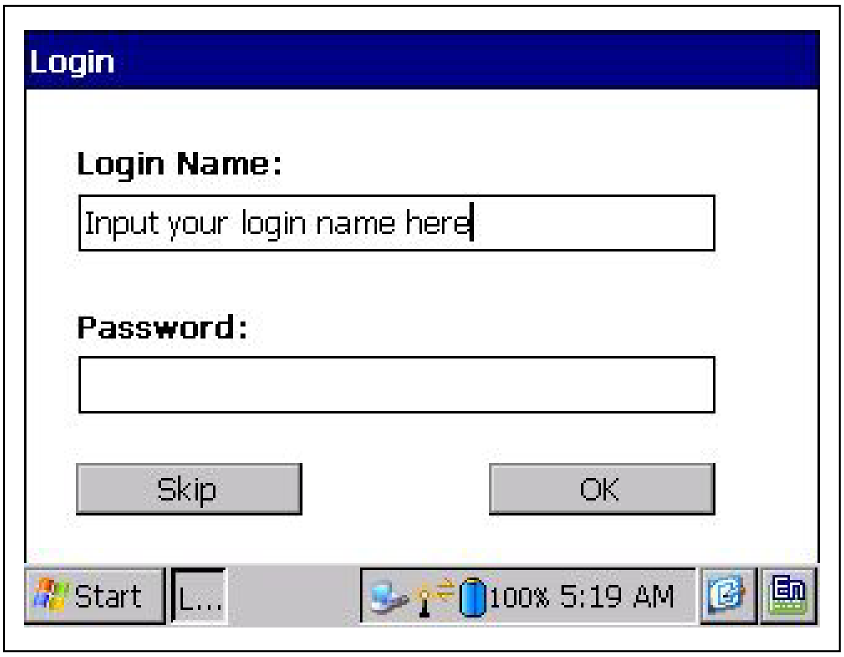


Figure 6-3 ID and Password Page

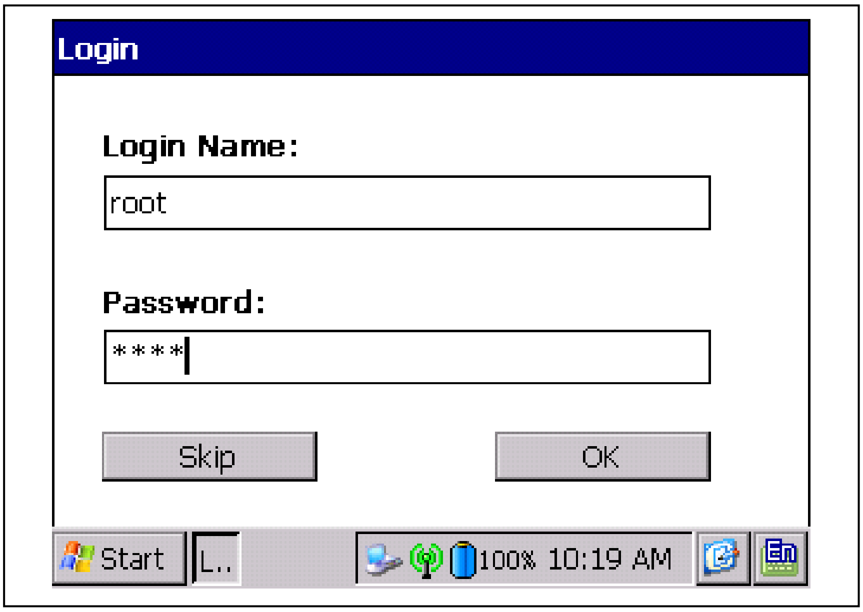


Figure 6-4 ID and Password Page

6.4 Applications Selector Screen

The Applications Selector Screen contains buttons that carry out different CS101-2 functions. This is a multiple screen interface, where user can navigate to the next screen using the "More…" button.

Figure 6-5 Applications Selector Screen 1

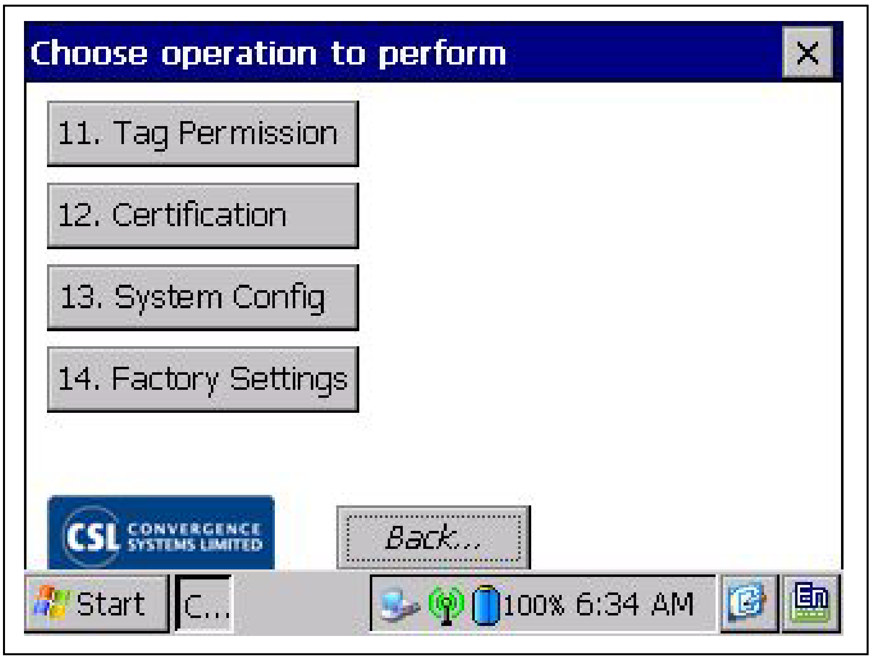


Figure 6-6 Applications Selector Screen 2

6.4.1 Tag Read

To read tags, one can use the Tag Read demo application. Firstly, select “Tag Read” in the main menu, then select “Scan First”, all tag’s EPC ID within the readable range will be read into the handheld reader.

If one want to read further information from the desirable tag, such as access password or kill tag password, select the tag EPC ID from the EPC list, then press “Read Selected”. If the tag memory banks are locked, access password is required to provide.

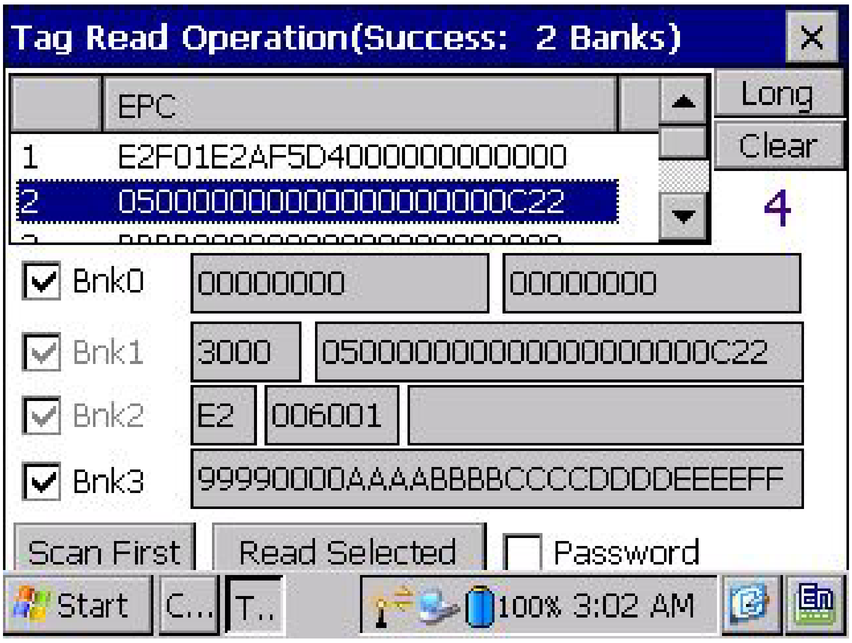


Figure 6-7 Tag Read

6.4.2 Tag Write

To write tag by using the Tag Write demo application, firstly, select "Tag Write" in the main menu, then select "Scan First" to read all tags EPC IDs into the reader, now, you can read further tag information by selecting any tag EPC ID from the tag list and then clicking the button of "Read Selected". Different memory band can be selected to be written by highlighting the memory bank and then keying in a new value, click "Write Selected" to start write tag at once.

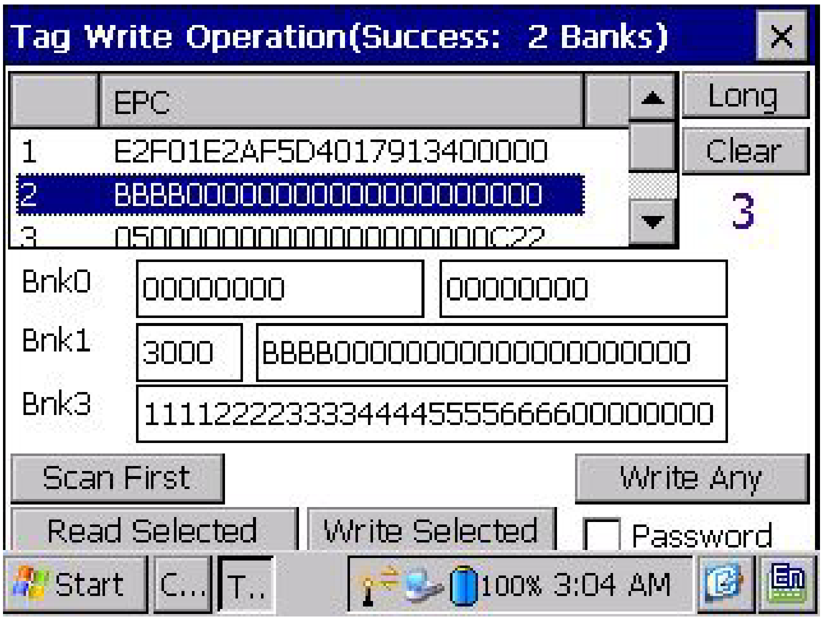


Figure 6-8 Tag Write

You can also rewrite any tag's EPC ID regardless of its original EPC ID. Configure the reader settings by entering the new ID in "EPC Value" and then click "Write Any" button. If you just want to write one tag only, tick the option of "Stop at first tag", or tick the "Auto-increment" box to write more than one tag with the EPC value is automatically increment.

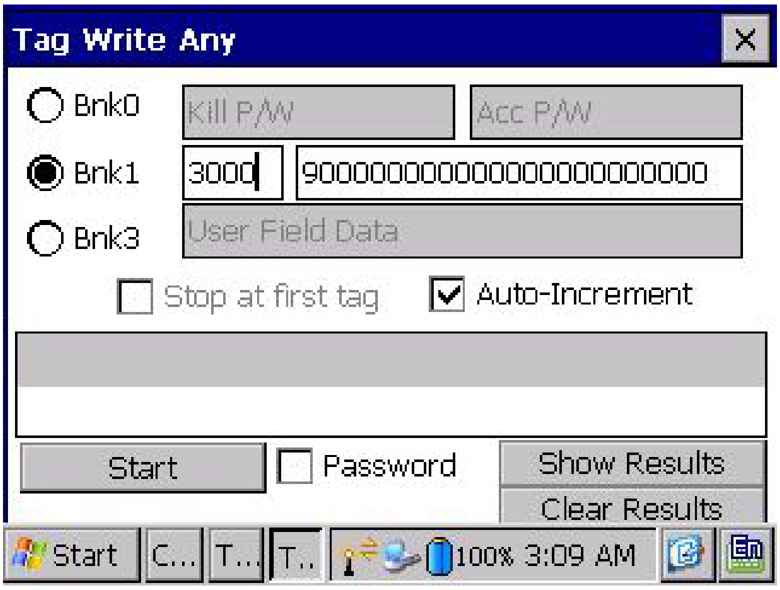


Figure 6-9 Tag Write – Auto-Increment

We can check the original ID on the left column (Accessed Tag ID) and the corresponding written ID on the right column (New ID).

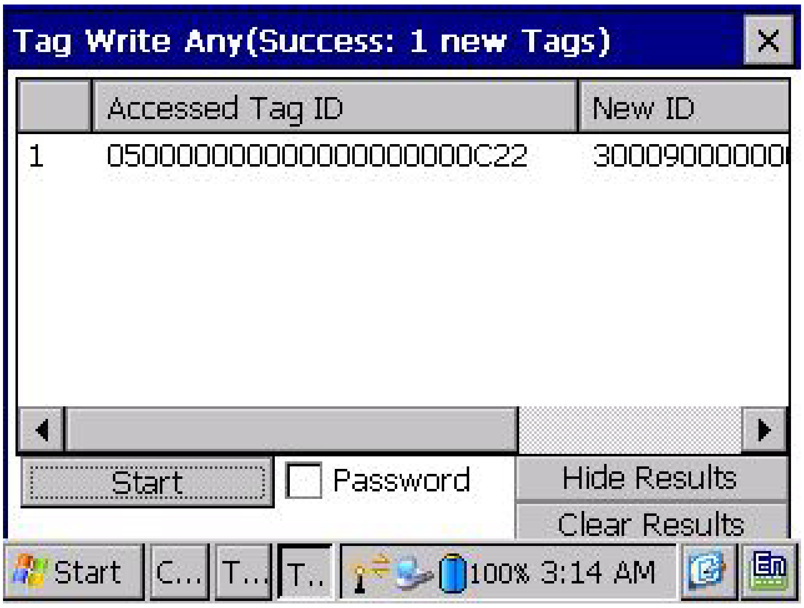


Figure 6-10 Tag Write – Result

Enter the tag mask in the "Tag Group Mask" so that the tag EPC ID will not be written when the prefix of tag EPC ID is same as the masking value.

Figure 6-11 Tag Write – Masking

Press "Start Write" to write tag, the handheld reader will write the tag non-stop until you press the "Stop" button.

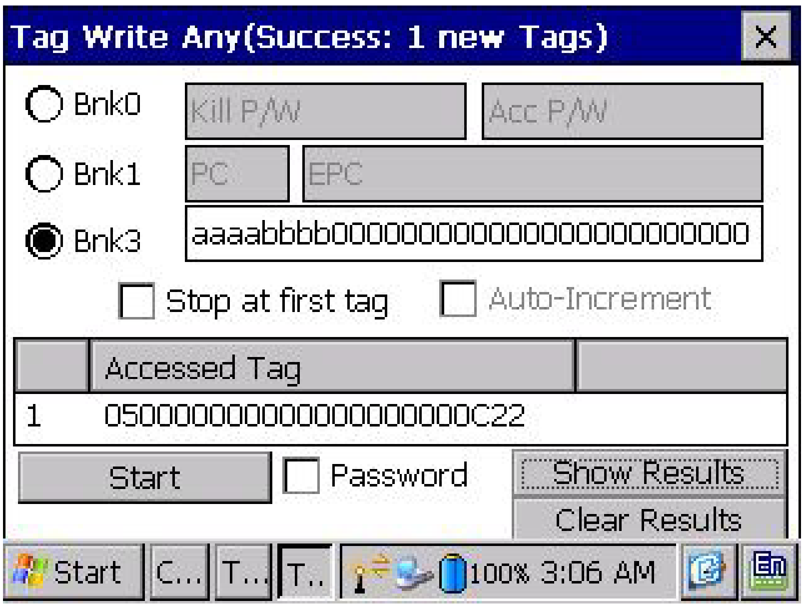


Figure 6-12 Tag Write – Result

6.4.3 Tag Inventory

Tag inventory provides an efficient way to read and count all tags at a time. In tag inventory, a duplicate filter is implemented to filter out the tags with same EPC ID, hence the tags with same EPC ID will not be displayed on the screen more than once. The function is most useful for warehouse inventory or whatever kind of inventory where you want to know the unique tags in the environment.

Besides the EPC ID, you can also read the ambient temperature and internal temperature of the handheld reader.

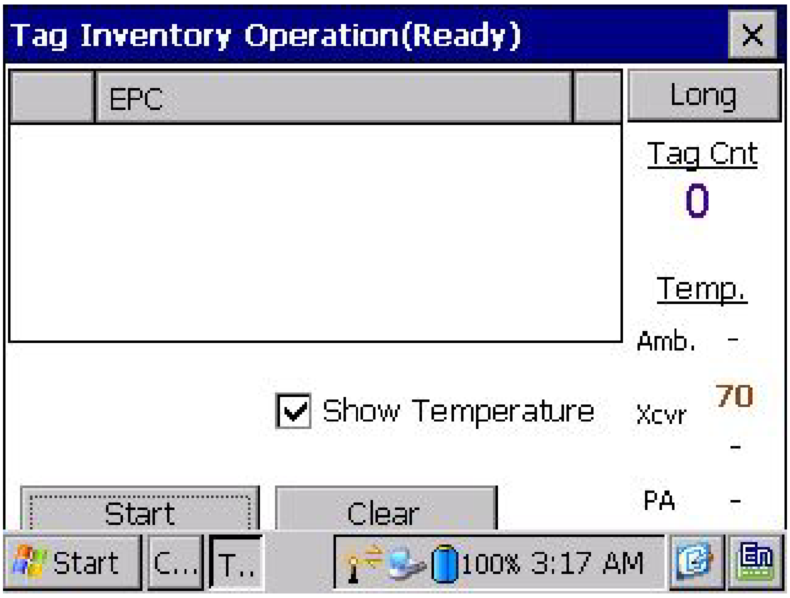


Figure 6-13 Inventory Main

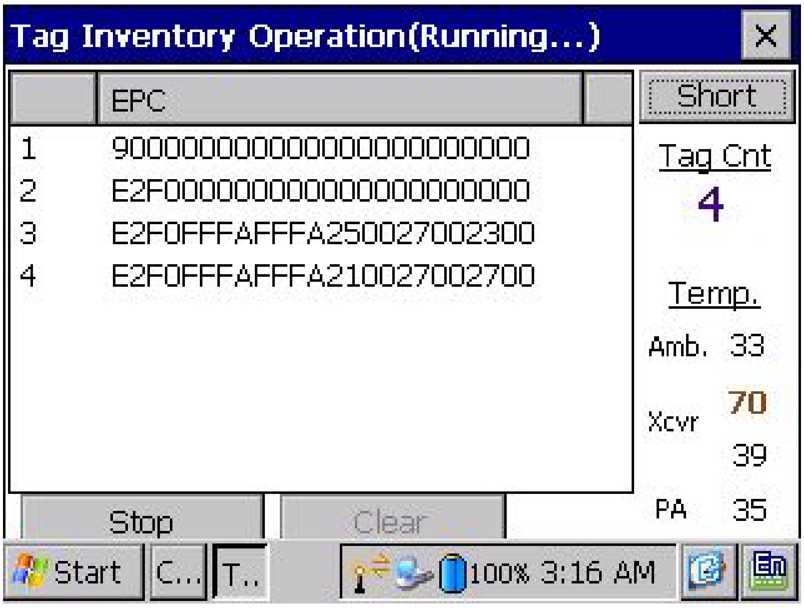


Figure 6-14 Inventory Result

6.4.4 Tag Ranging

The Tag Ranging lets user read tags with RSSI (RF Signal Strength Indicator), the RSSI value will keep changing when the handheld reader moves to and from the tags (RSSI will increase when the handheld reader is moved close to the tags). It is useful for the user to identify a tag when it starts falling into the read range of the reader. This function is not really for normal operation use but is meant mainly for system integrator to select appropriate tag designs in the actual application environment. In other words, for each customer he works with, the asset to be tagged may be different, and the customer business process may be different. To satisfy that the exact model of tag that can fulfill the requirements may be different. In that case the system integrator needs to put different types of tags onto the asset in the real environment and use this program to check if indeed the tag can be read at the position of the operator.

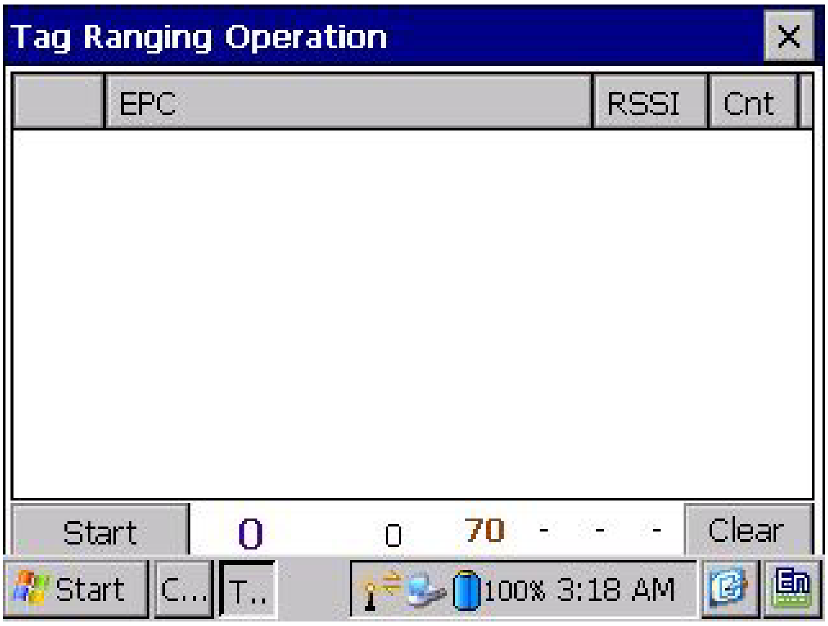


Figure 6-15 Tag Ranging Main

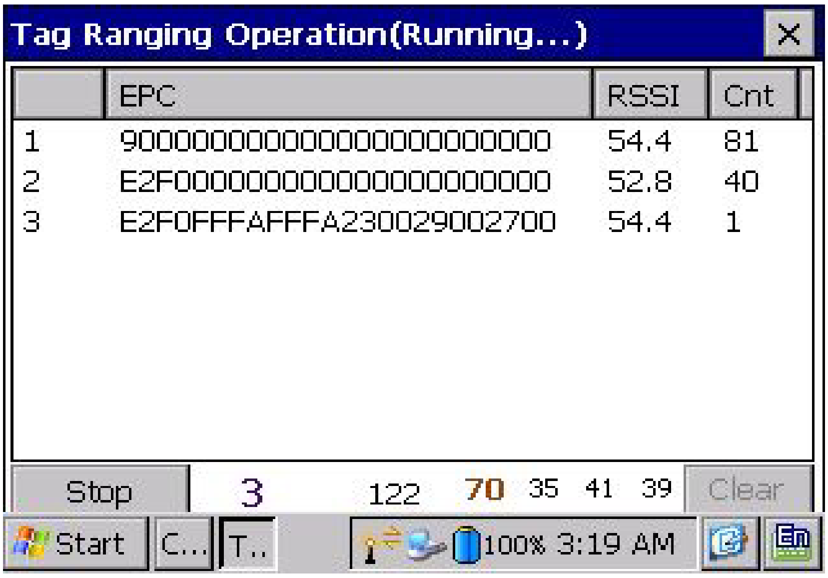


Figure 6-16 Tag Ranging Result

6.4.5 Tag Search

The Tag Search application allows user to zero in onto tag using a Geiger like buzzer pattern. It is useful for the application of item searching, since the item location can be found out by keeping track with the RSSI or tag rate.

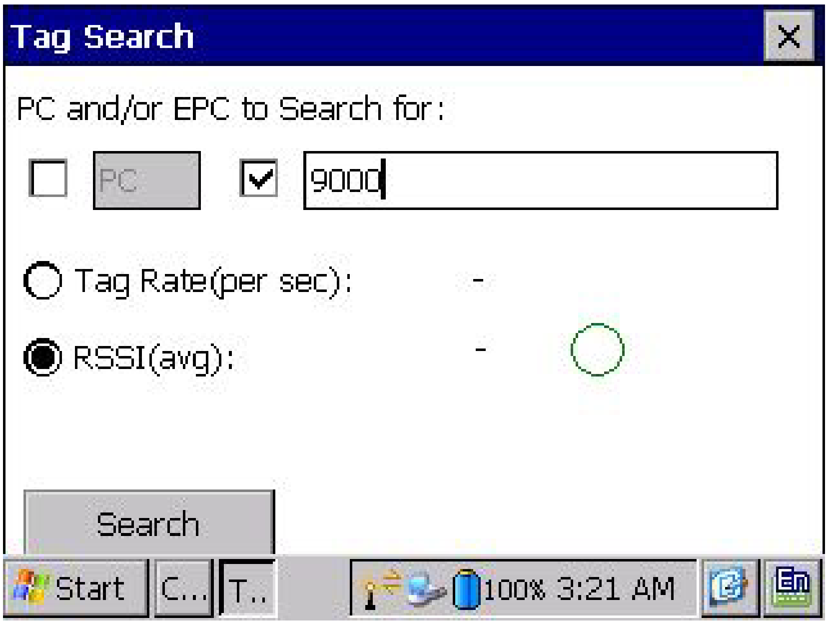


Figure 6-17 Tag Search

Figure 6-18 Tag Search

6.4.6 Tag Commissioning

The Tag Commissioning allows the user to associate the Bar Code ID and Tag ID and then save it into a file in CSV format

Figure 6-19 Tag Commissioning – tag read

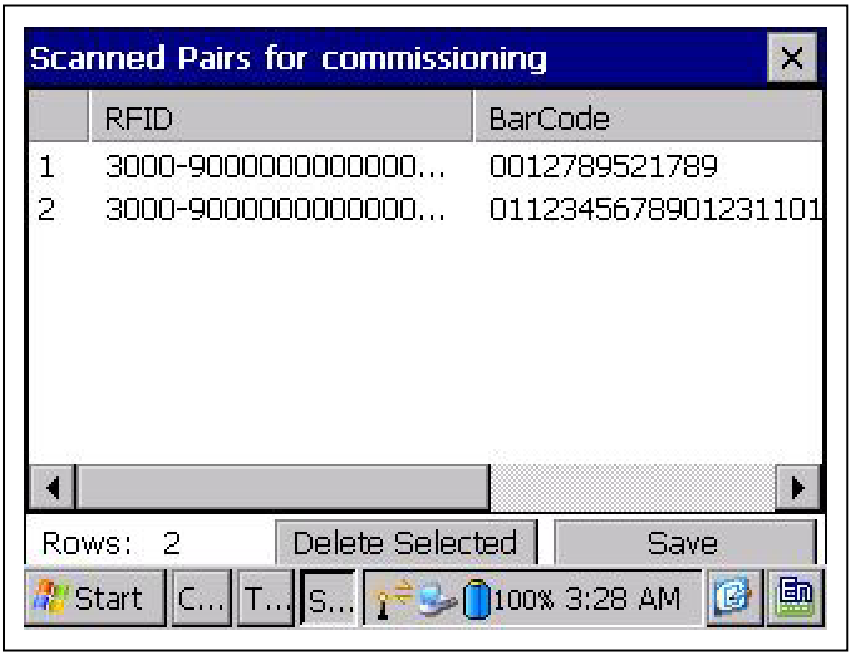


Figure 6-20 Tag Commissioning – association

6.4.7 Tag Authentication

The Tag Authentication allows the user to compare between the Barcode/EPC ID based on a CSV file saved in the handheld reader and the Barcode/EPC ID that can be read currently.

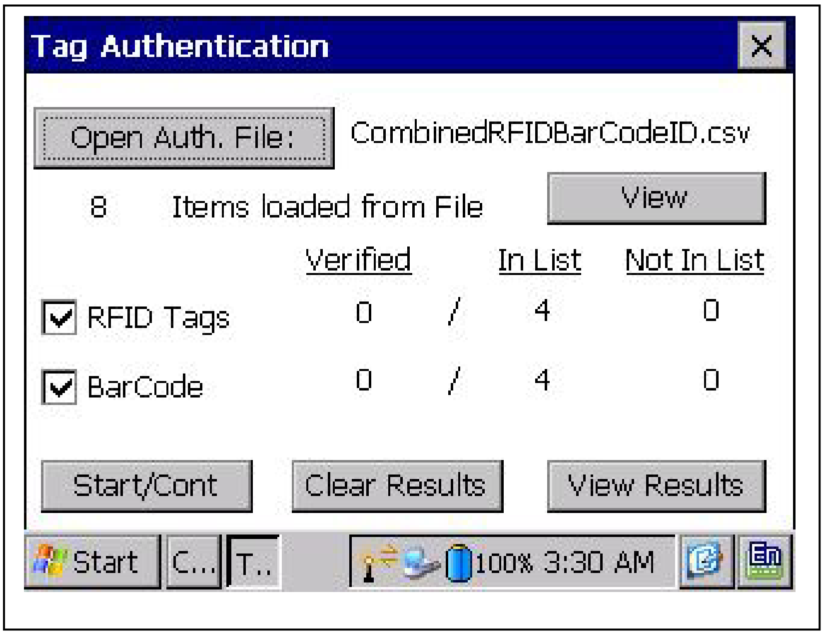


Figure 6-21 Tag Authentication – Main

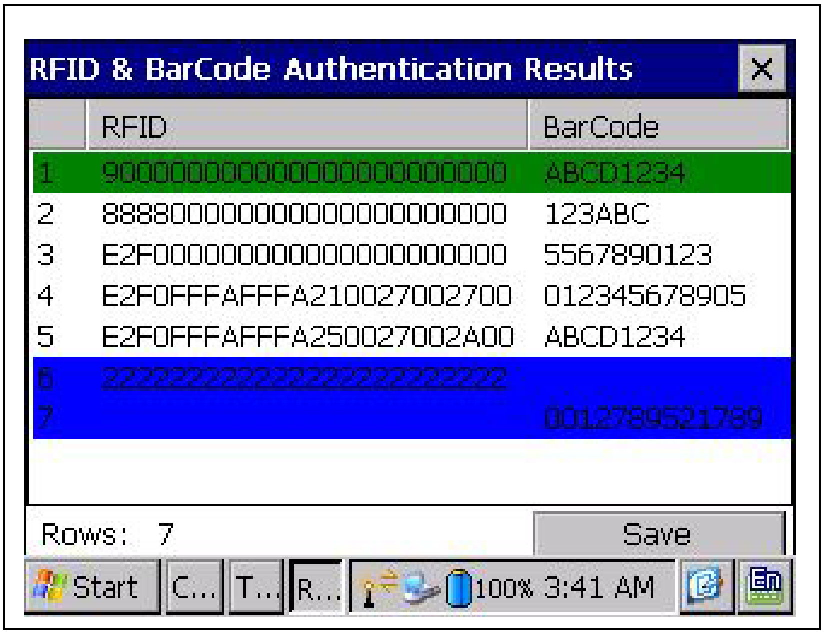


Figure 6-22 Tag Authentication – Result

6.4.8 Database Management

Inventory: Tag Inventory including EPC ID, RSSI and processing date/time can be stored into a file and then upload or download to/from remote server.

Data file format: file format to be processing
Remote File: define the location and file name for the remote server so that the handheld reader can upload or download the file through this location
Local File: define the local file location in the handheld reader
View Data: list the tag records stored in the local file
Consolidate: combine the tag record read from the handheld reader and the local file

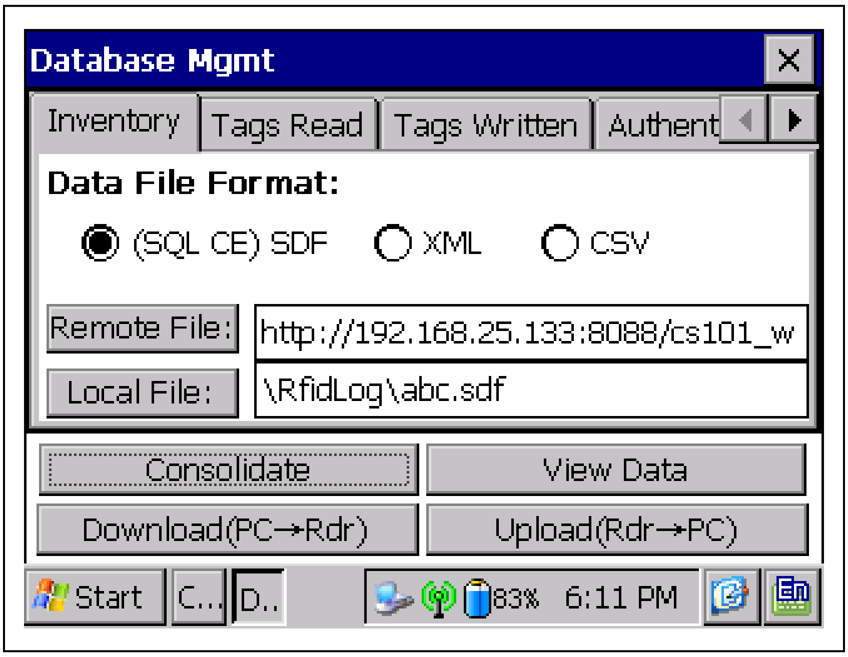


Figure 6-23 Database Management

6.4.9 RFID Configuration

The RFID Configuration allows the user to set parameters for the Inventory, Tag Memory, Frequency, Link Profile, Antenna and Overheat Protection.

Inventory Setup:

Session: Session number must be different from reader to reader if they are pointing into the same zone.

Est. Tag Population Size: it is the estimated population of tags to be read at a time.

Tag Filter Mask: Set the filter to select the tags that you want to read/write in the tag inventory submenu.

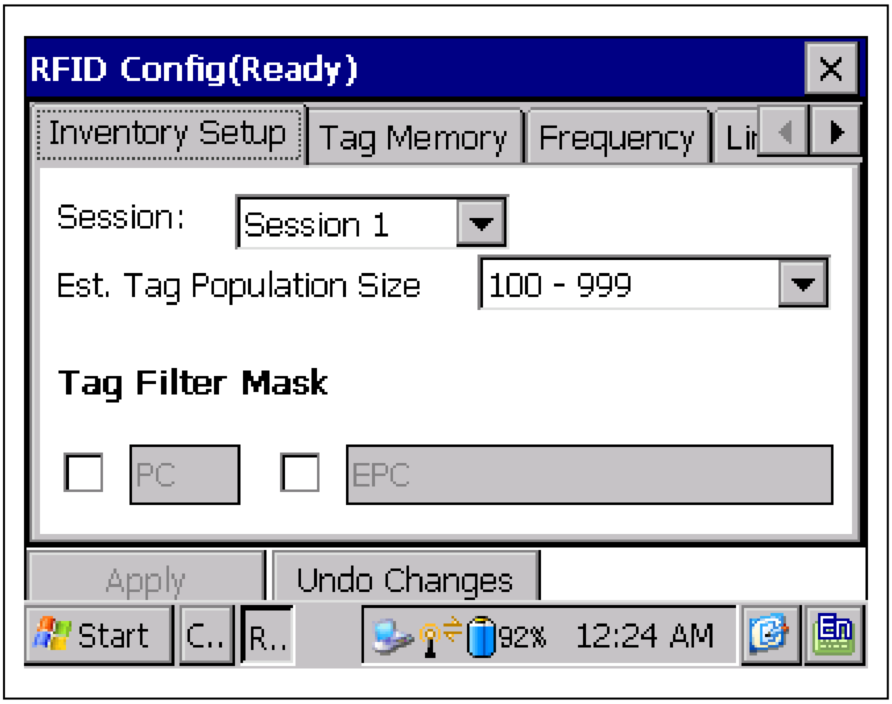


Figure 6-24 RFID Configuration – Inventory Setup

Tag Memory:

Vendors: Select the vendor type to determine the size of the memory bank

Tag Bank Sizes: Besides the predefine memory size from different tag vendors, you can also change the size depending on the type of the tags.

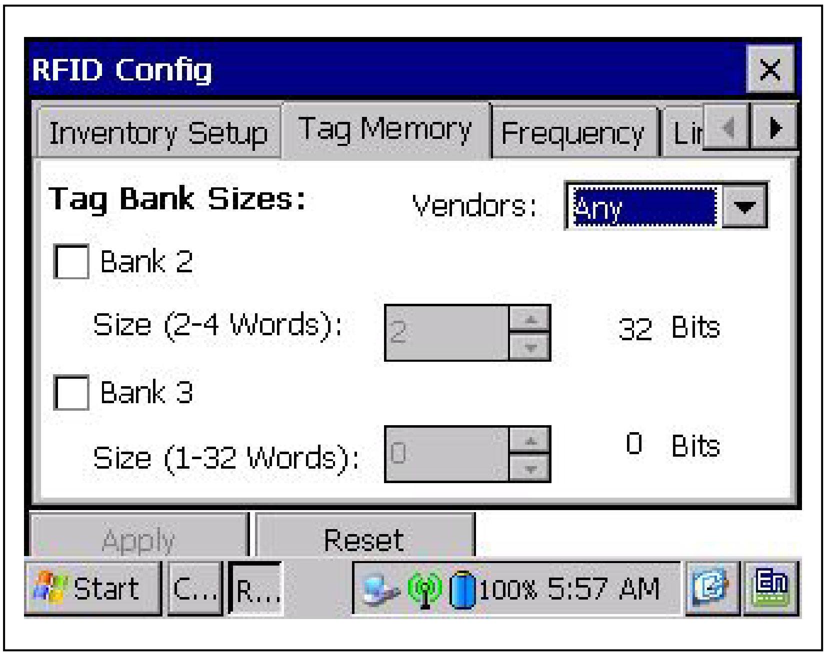


Figure 6-25 RFID Configuration – Tag Memory

Link Profile: Different modulation profile can be selected by the user for different situation

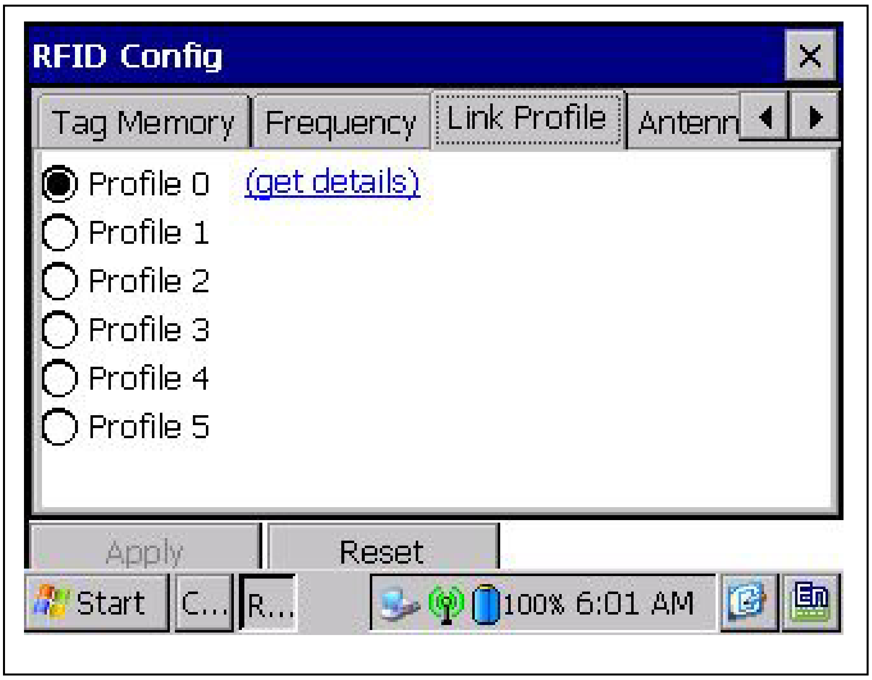


Figure 6-26 RFID Configuration – Link Profile

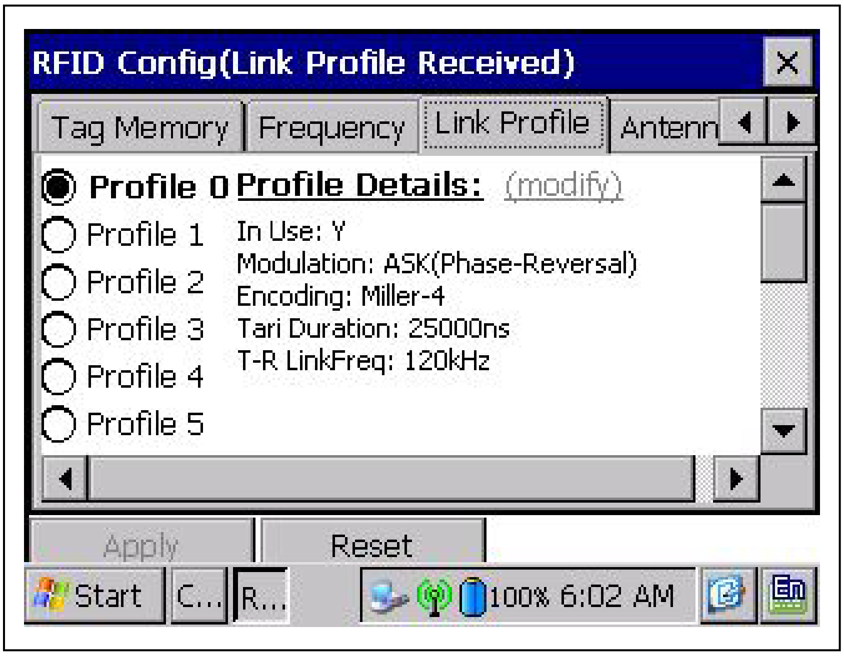


Figure 6-27 RFID Configuration – Link Profile Details

Antenna: Depending on the read range and the tag type, you can adjust the antenna power range from 0 to 30dBm

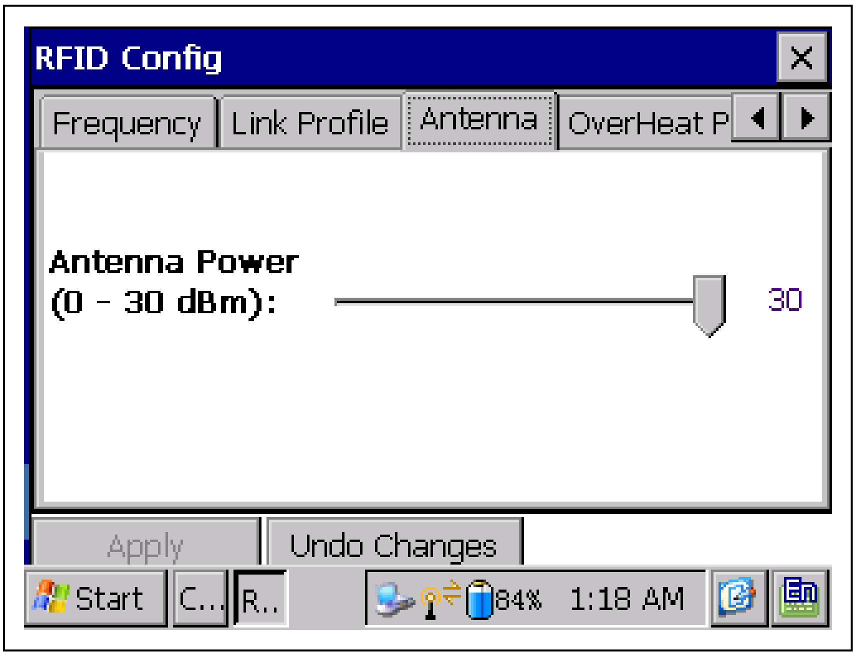


Figure 6-28 RFID Configuration – Antenna

OverHeat Protection: The function in this page lets the user to set the Antenna on/off duty cycle and transceiver temperature to protect the handheld reader to avoid overheat

Duty Cycle: The function of duty cycle prevents the user to read/write for a long time. When tag read/write is working over the predefined period, it will stop to do the tag read/write and then start it again for another predefined period.
Overheat Protection: Set a temperature value here so that the reader will show an overheat warning when the handheld internal temperature is reached to this value.

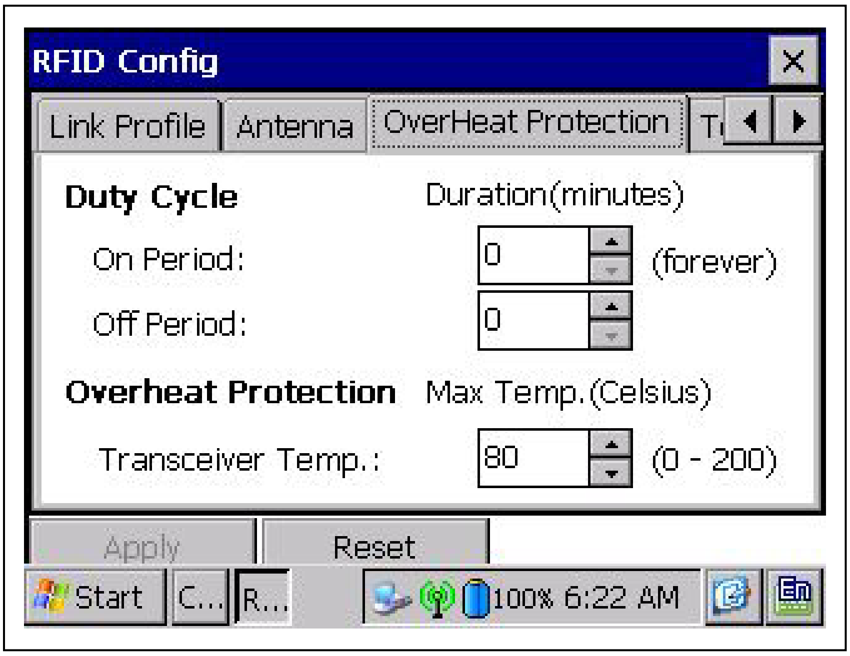


Figure 6-29 RFID Configuration – OverHeat Protection

Temperature: when the handheld reader temperature reaches the predefine temperature value on this page, it will display the warning message and stop tag read/write function.

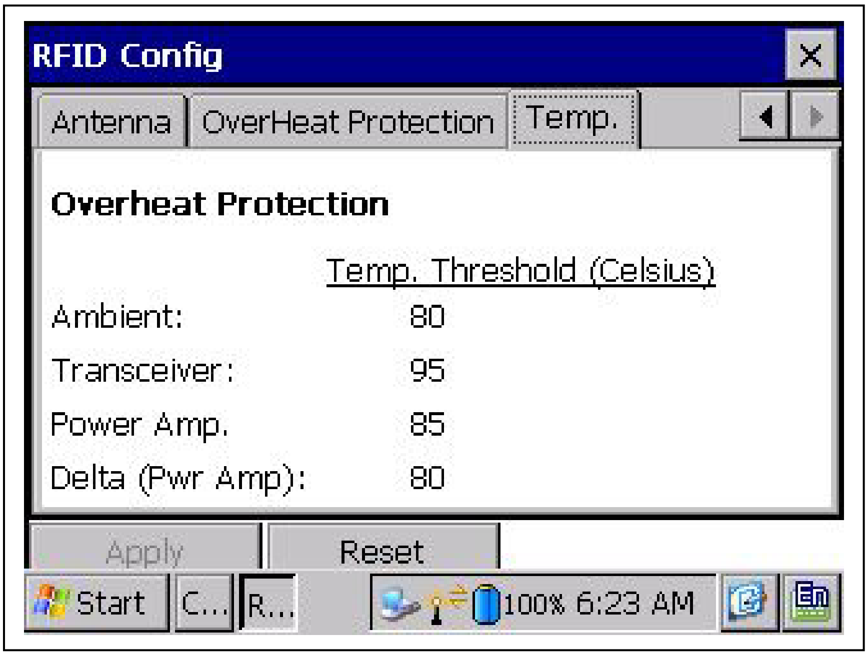


Figure 6-30 RFID Configuration – Temperature

6.4.10 Scan Barcode

The Scan Barcode allows the user to scan barcode and then save it into a file in CSV format

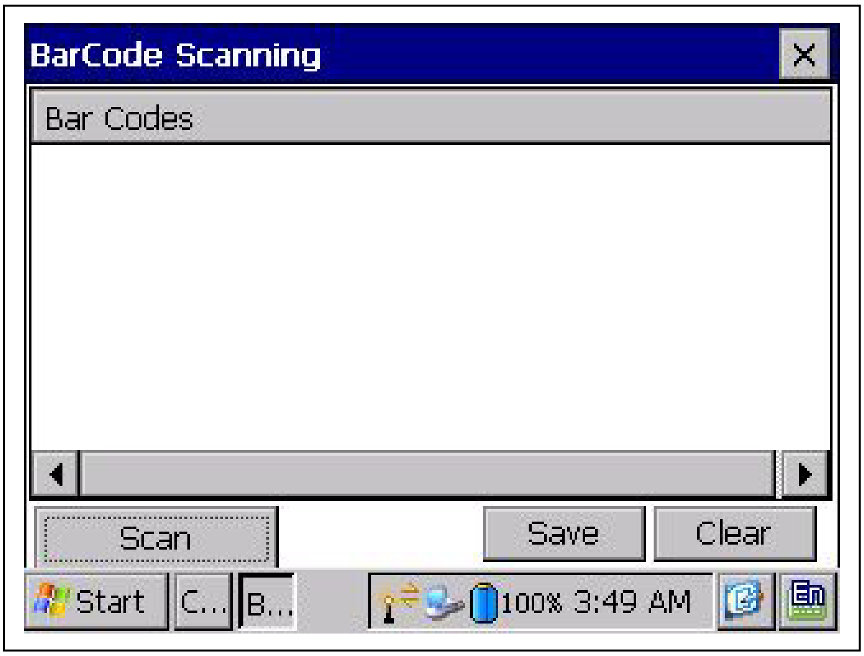


Figure 6-31 Scan Barcode - Main

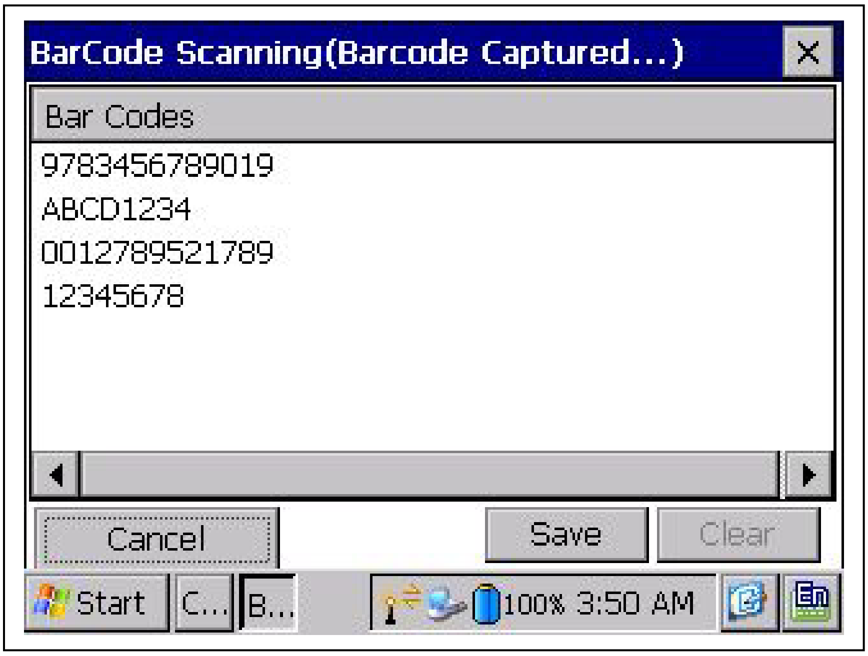


Figure 6-32 Scan Barcode - Scanning

6.4.11 Tag Security

Tag Security: You can use the tag security to set the protection feature of the tag.
Firstly, click “Choose another Tag” to scan the available tag that is placed within the coverage of the handheld reader, and then select the tag from the tag list.
The handheld reader can let the user to set the protection of kill password, access password, EPC ID, TID and user bank depending on the tag type.

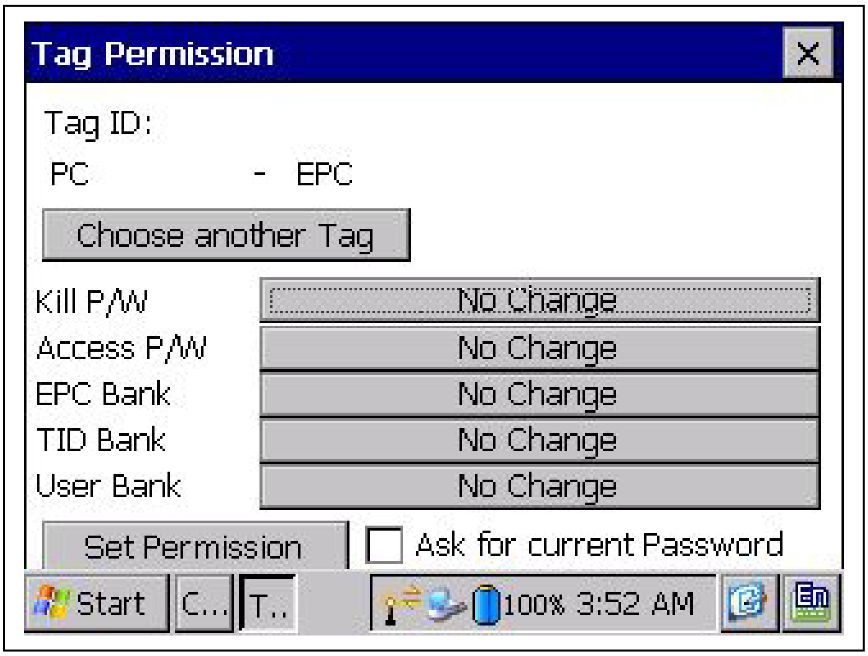


Figure 6-33 Tag Permission

Allow: allow read/write the memory bank
Always Allow: Tag can never be locked
Password Protect: need password when access the tag memory bank
Always Deny: tag cannot be read even correct password is provided
No Change: keep previous status

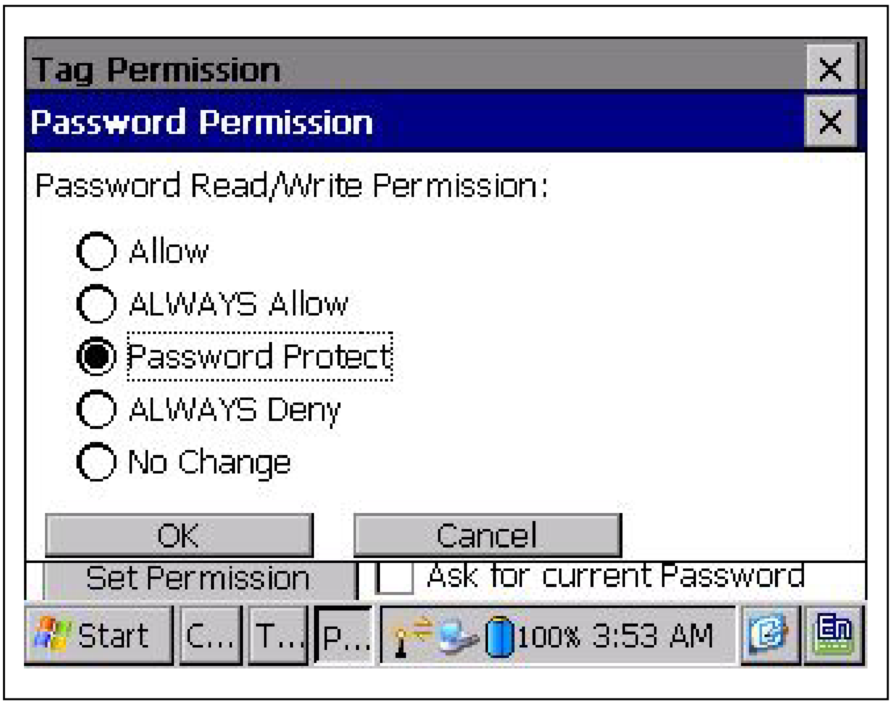


Figure 6-34 Tag Permission

6.4.12 System Configuration

The System Configuration contains the submenu for Identity the reader, user login in/out, sound melody for different user function.

Identity: Set the unique name/ID for the reader

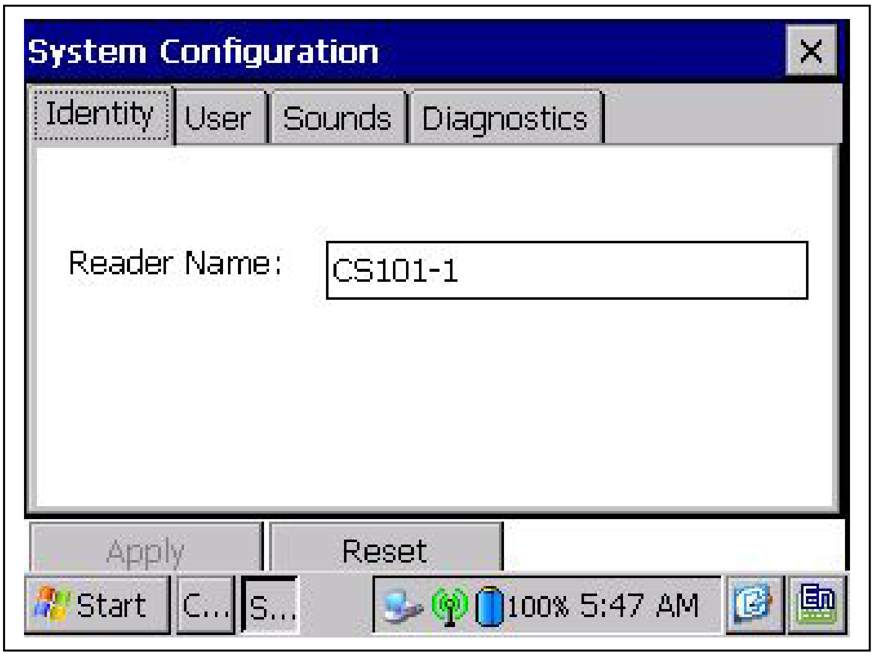


Figure 6-35 System Configuration - Identity

User: Set login name and password can restrict the unauthorized user to run the demo program in this handheld reader.

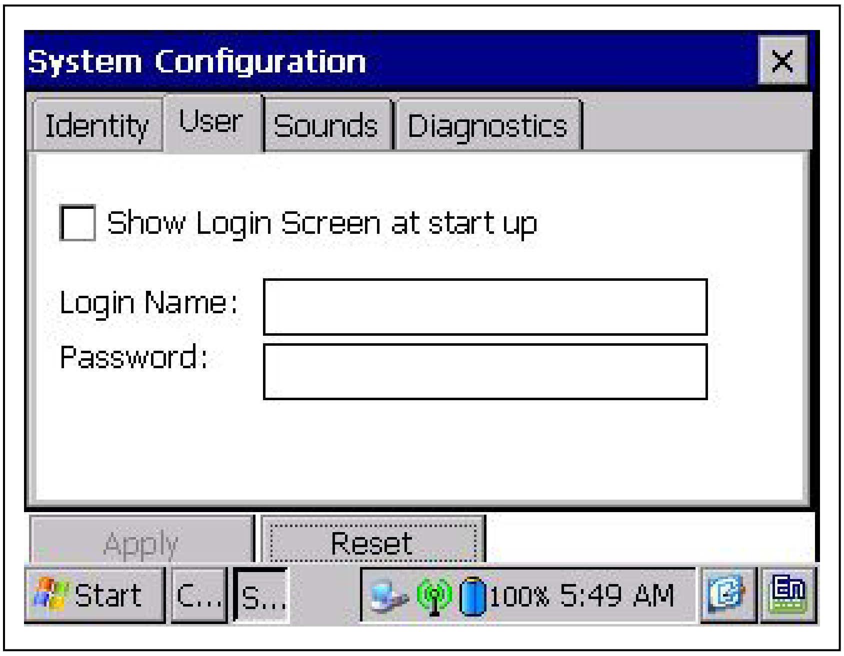


Figure 6-36 System Configuration - User

Sounds: Sound melodies and volume can be chosen in this page. One can also enable or disable the sound melody for each type of operation.

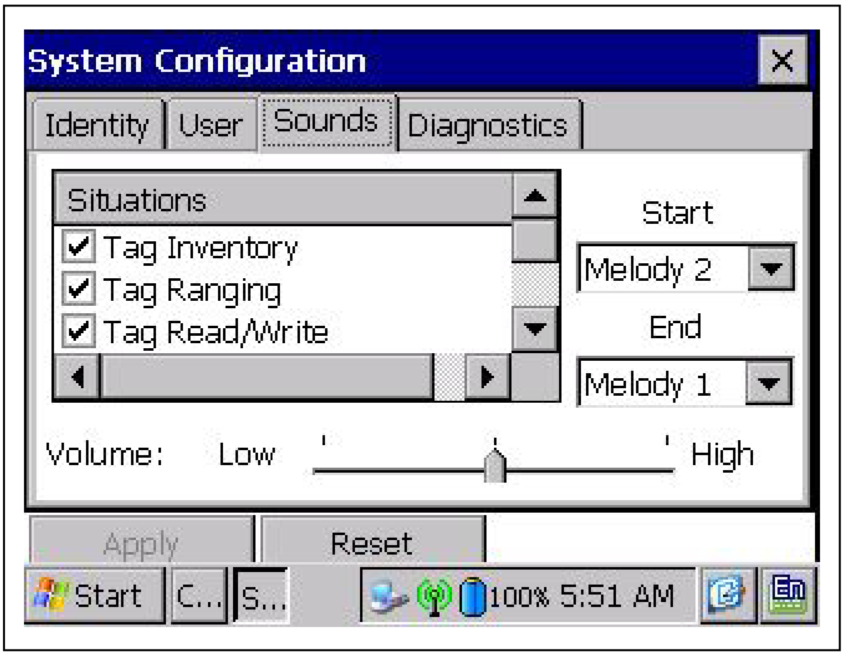


Figure 6-37 System Configuration - Sounds

Diagnostics: The Diagnostics submenu allows the user to check the version of current RFID driver and MAC

User can also check the current and record high temperature of the transceiver and power amplifier. Trace log can be enabled by check the box of trace log

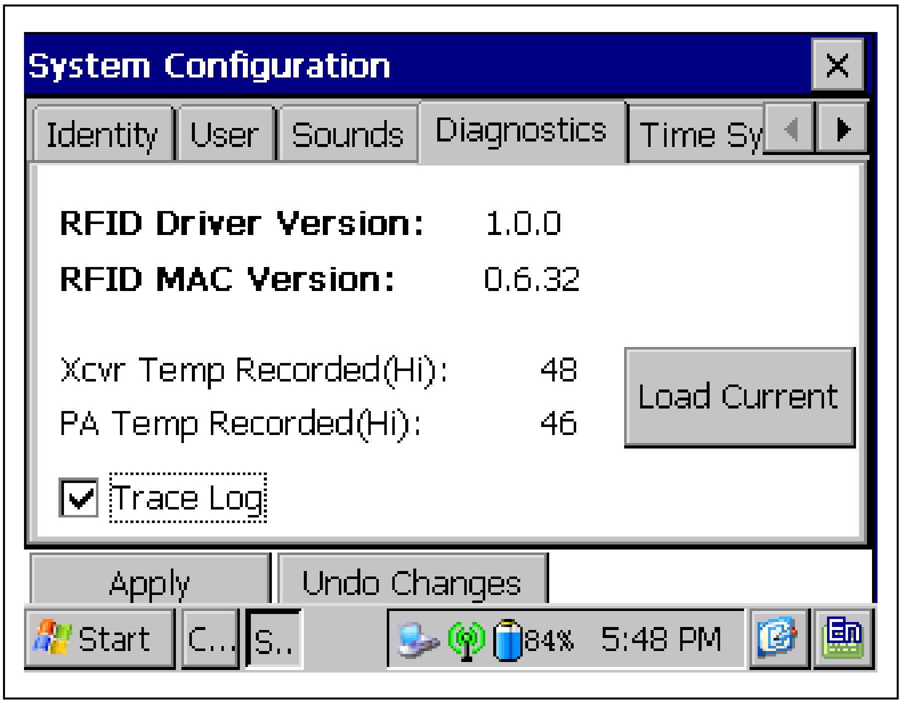


Figure 6-38 System Configuration - Diagnostics

Time Synchronization: This page allows you to set the NTP server so that the system time can synchronize with NTP server.

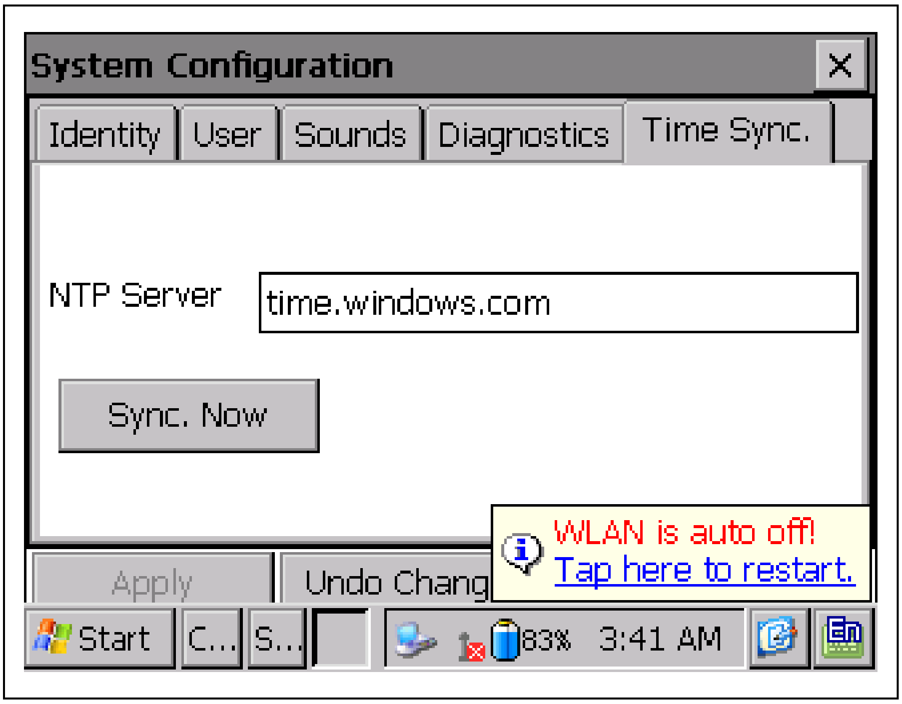


Figure 6-39 System Configuration – Time Synchronization

6.4.13 Factory Defaults

Factory Defaults: User can use the factory defaults to reset the RFID Config, System Config and data Folders into the default settings.

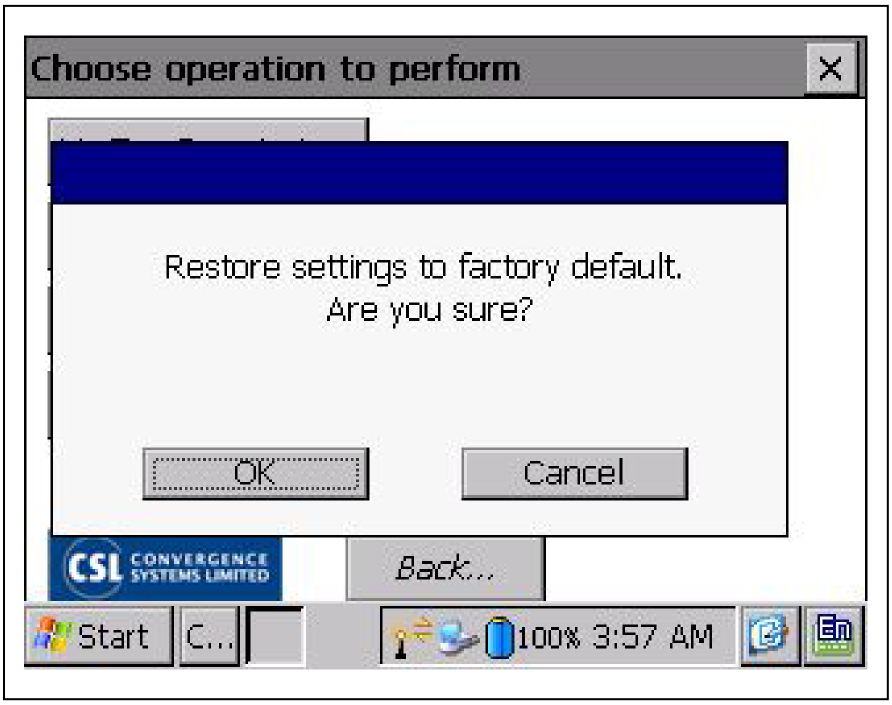


Figure 6-40 Factory Defaults

7 Software Development Kit

The CSL CS101-2 handheld reader software development kit provides the following components for quick and easy application development:

1. Software specifications
2. Block diagrams
3. Application Programming Interface (API) definitions
4. Application scenarios with program source codes
5. Unit test plan and results
6. Build environment
7. Debug methods

7.1 Software Specifications

The overall software architecture consists of CS101-2 RFID Libraries on the WinCE OS inside the handheld reader, CS101-2 Demonstration Application (which consists of a whole series of applications, such as tag read, tag write, tag inventory, tag search, tag authentication, tag commissioning, barcode scanning, RFID configuration, system configuration, database file manipulation, network database file transfer, etc.), CS101-2 Keep Alive Monitor, all of the above inside the handheld device; and then also CS101-2 Server Side Database Administration Application, which resides on the WinXP server side.

7.1.1 CS101-2 RFID Libraries

The CS101-2 RFID Libraries consists of 3 parts:

1. RfidSp
2. PosSp
3. ClsSys Util

These calls are designed to be called by C# applications with the PInvoke (Platform Invoke) method.

7.1.2 CS101-2 Demonstration Application

The CS101-2 Demonstration Application is a comprehensive C# demonstration program that demonstrates how to write an application on the CS101-2 platform. It offers all possible RFID related and barcode related functionalities. The purpose is to give a good Out-Of-Box OOB experience to the system integrator. The functions include:

1. Tag Read
2. Tag Write
3. Tag Inventory
4. Tag Ranging
5. Tag Search
6. Tag Commissioning
7. Tag Authentication

8. Database Management
9. RFID Configuration
10. Scan Barcode
11. Tag Security
12. System Configuration
13. Factory Defaults

7.1.3 CS101-2 Keep Alive Monitor

CS101-2 Keep Alive Monitor is an independent application that is turned on during WinCE boot up to monitor health situations, including:

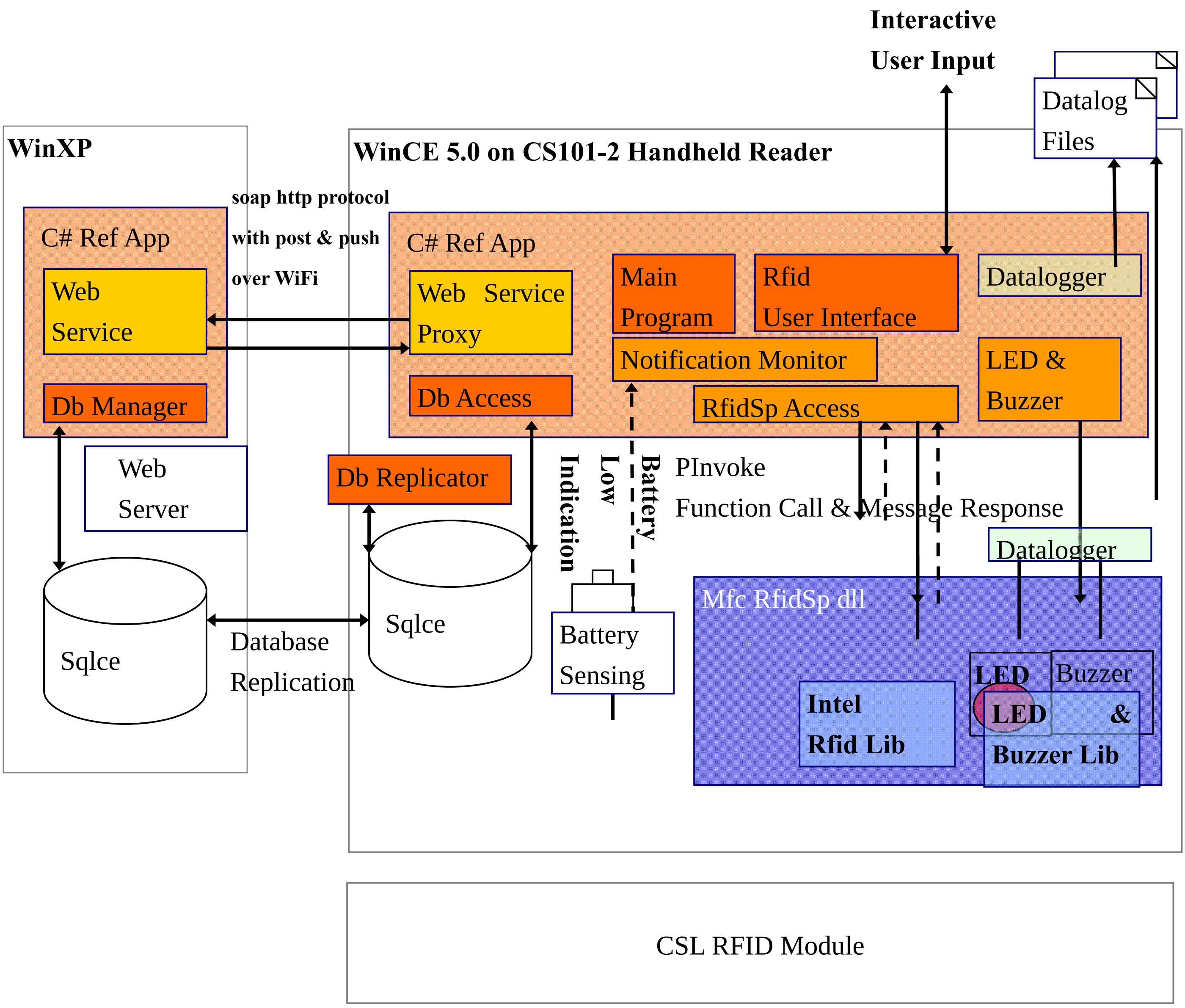
1. Battery Monitoring and Alert
2. Memory (RAM) Monitoring and Alert
3. Disk Space (Internal Flash) Monitoring and Alert
4. Disk Space (SD Card) Monitoring and Alert
5. SD Card Physical Action Monitoring and Alert (insertion and ejection)
6. Network Condition Monitoring and Alert
7. Automatic Files Backup

7.1.4 CS101-2 Server Side Application

The CS101-2 Server Side Application handles collection of tag data and converting them to typical formats.

7.2 Block Diagrams

The software architecture is illustrated by the following block diagram:



On the WinCE machine:

The PDA is connected to the intranet through the WiFi Access Point. The PDA has a DHCP IP-address.
It has a local SqlCe database storing all the known information (e.g. Known Inventory, TagGroup to Lock). It should never goto the suspend state.

A) The **Db Replicator** is a standalone program that replicates the database data between the WinCE & WinXP machine.
B)**C# RefApp on WinCE** is a reference application. It provides 2 user interfaces, 1 database interface, 3 sub-system interfaces:
1) The **Web Service**:
This provides the network communication services to the Web Service on the WinXP.
2) The **Rfid User Interface**:
This provides the GUI (Window-Forms) on the LCD
3) The **Db Access**:
This connects to the local SqlCe Database (Microsoft SqlCe3.1). It has access to the data using sql commands.
4) The **RfidSp Access**:
This setup the RfidSp.dll. This controls the Rfid Reader & get back raw data from the reader &/ the post-processed data from the Rfid Middleware.
5) The **LED & Buzzer Control**:
This controls the 7-color LED & the buzzer (volume & frequency).
6) The **Notification Monitor**:
This alert the main program that the “battery-low” notification is signaled, & the main program should alert the user to exit the Rfid application immediately (in order to terminate the connections gracefully & has the latest data stored locally).
7) The **Datalogger**:
This, when enabled, writes the datalog text to the logfiles.
8) The **Main Program**:
This is the central unit of the application.
It controls the calling sequence to all the modules described above.

On the WinXP machine:
The PC has a fixed IP & it is in the intranet.
C) **C# RefApp on WinXP** provides 2 features, namely the web services & the database

management:

1) The **Web Services**:

This provides Soap (xml-text & binary) data over the HTTP GET, HTTP POST, or SOAP protocol to the client upon request or web-push.

2) The **Database Manager**:

This allows the user to edit/import/export/review the Master Database, & setup the Sql Data (by stored procedures) for each WinCE Rfid Reader to get.

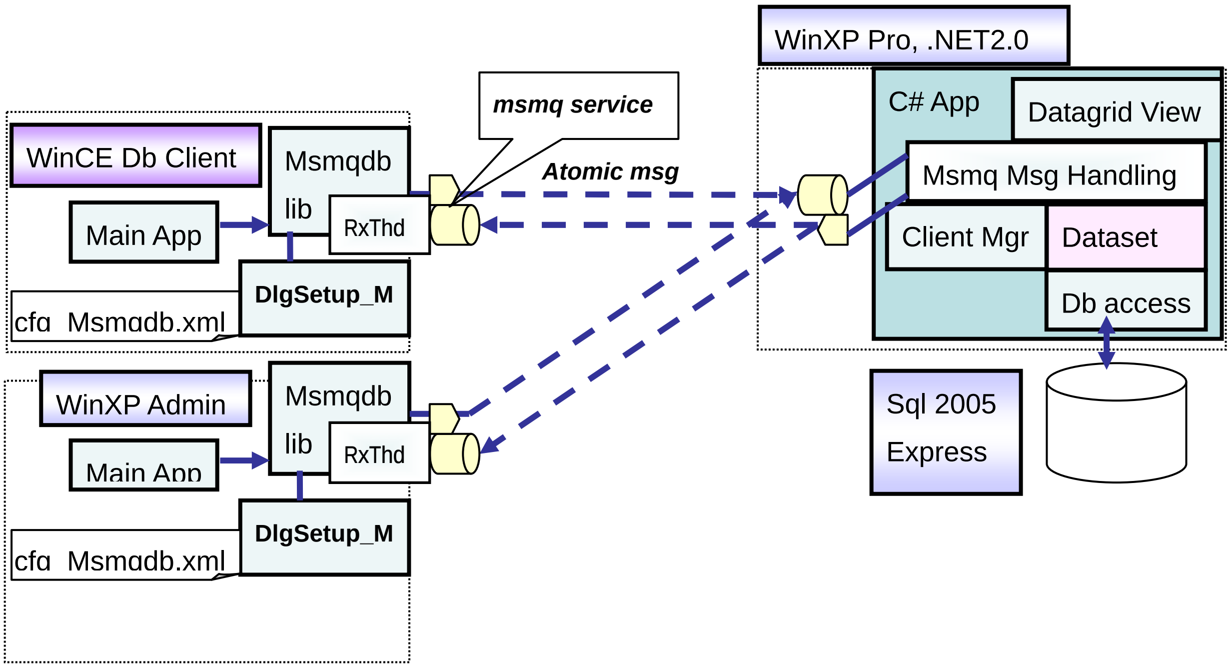
To Probe Further:

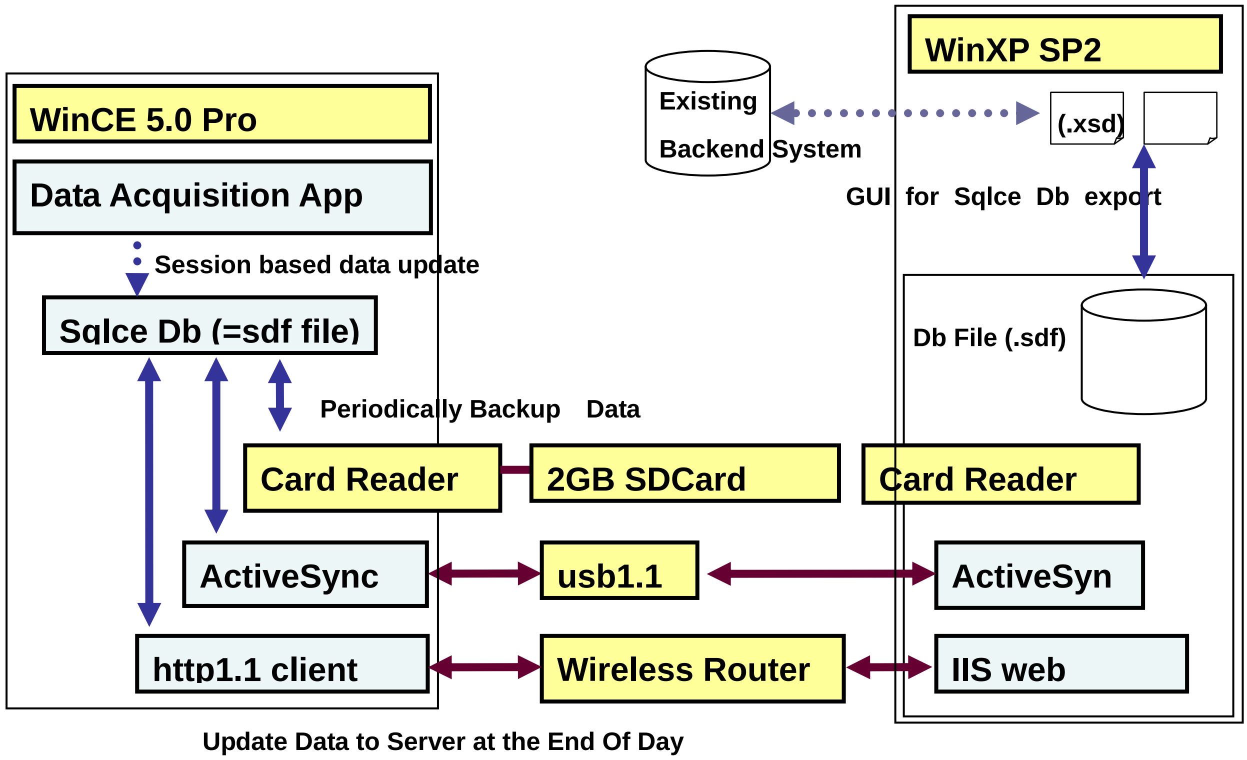
1) If there is only 1 WinCE & WinXP machine, the database file (\Program Files\Rfid\Db.sdf) can also be copied between WinCE & WinXP through ActiveSync or ftp.

In general, the database Replication between the SqlCe server on WinCE & WinXP is done by the RDASync (the Remote Data Access Synchronization) technique from Microsoft. Synchronization between Sql2005 & SqlCe on WinXP is not included in the reference solution.

2) The required 802.11 a/b/g WiFi Access Point provides intranet connection & assign DHCP IP address for the WinCE devices. WEP/ WPA/WPA2 Encryption is recommended.

3) **Encryption** (using Microsoft Windows CE Enhanced Cryptographic Provider) can be added to the C# programs for the WinXP-to-WinCE Soap data stream, if the additional loading is acceptable.


WinXP Pro, .NET2.0
msmq service
Atomic msg
C# App
Datagrid View
Msmq Msg Handling
Client Mgr
Dataset
Db access
Sql 2005
Express
WinCE Db Client
Main App
Msmqdb
lib
RxThd
DlgSetup_M
cfg_Msmqdb.xml
WinXP Admin
Main App
Msmqdb
lib
RxThd
DlgSetup_M
cfg_Msmqdb.xml



WinCE 5.0 Pro
Data Acquisition App
Session based data update
Sqlce Db (=sdf file)
Periodically Backup Data
Card Reader
2GB SDCard
Card Reader
ActiveSync
usb1.1
ActiveSyn
http1.1 client
Wireless Router
IIS web
Update Data to Server at the End Of Day
Existing Backend System
WinXP SP2
(.xsd)
GUI for Sqlce Db export
Db File (.sdf)

7.3 Application Programming Interface (API) Definitions

This chapter describes the interface definition for the three CSL C# libraries, namely, the **RfidSp** library, the **PosSp** library and the **ClsSysUtil** library.

RfidSp:

Overview:
RfidSp is a C# class in the that provide a C# managed interface of the Rfid Reader Threads. RfidSp dll is designed to be used by our reference applications, which provides a wrapper class for function calls & a Message Window class to receive messages.
C# namespace:
ClslibRfidSp .
Dependencies:
Program Files\W_RfidSp.dll;

1. Type Definitions:

HRESULT_RFID_STATUS
Prototype:
using HRESULT_RFID_STATUS = ClslibRfidSp.HRESULT_RFID; //= System.Int32;
Description:
This enumerates the status in the response messages.

RFID_RADIO_HANDLE
Prototype:
using RFID_RADIO_HANDLE = System.UInt32;
Description:
This is the handle to the RFID radio object. A zero or negative is an invalid value.
e.g. A valid value is 0x00010000.

2. Constants:

(member variables in class RfidSp)
RFID_PACKET_18K6C_TAG_ACCESS__DATA_MAXSIZ

Prototype:

public const int RFID_PACKET_18K6C_TAG_ACCESS__DATA_MAXSIZ = 32;.

Description:

This is the maximum number of UINT32 in tag_access a message defined in the Rfid library.

RFID_PACKET_18K6C_INVENTORY__DATA_MAXSIZ

Prototype:

public const int RFID_PACKET_18K6C_INVENTORY__DATA_MAXSIZ = 24;

Description:

This is the maximum number of UINT32 in a tag_inventory message defined in the Rfid library.

WM_USER

Prototype:

public const int WM_USER = 0x0400;

Description:

This is the starting index for user-defined message on WinCE.

RFID_INVALID_RADIO_HANDLE

Prototype:

public const RFID_RADIO_HANDLE RFID_INVALID_RADIO_HANDLE = ((RFID_RADIO_HANDLE)0);.

Description:

This is the invalid radio handle.

SELECTCRITERIA_COUNT

Prototype:

public const int SELECTCRITERIA_COUNT = 4;.

Description:

This is the number of selectcriteria to set.

POSTMATCHCRITERIA_COUNT

Prototype:

public const int POSTMATCHCRITERIA_COUNT = 4;

Description:

This is the number of postmatchcriteria to set.

RFID_18K6C_SELECT_MASK_BYTE_LEN

Prototype:

public const int RFID_18K6C_SELECT_MASK_BYTE_LEN = 32;

Description:

This is the size(in byte) of the select mask for partitioning a tag population.

RFID_18K6C_SINGULATION_MASK_BYTE_LEN

Prototype:

public const int RFID_18K6C_SINGULATION_MASK_BYTE_LEN = 62;

Description:

This is the size(in byte) of the single post-singulation match mask.

USHORTSEQNUMINVALID

Prototype:

public const int USHORTSEQNUMINVALID = 0xffff;

Description:

This is the value of the invalid RfidMw sequence number.

3. Enumerations:

HRESULT_RFID

Prototype:

```
public enum HRESULT_RFID : uint {
    S_OK                              = 0x00000000, // Success
    E_ABORT                           = 0x80004004, // Operation aborted
    E_ACCESSDENIED                    = 0x80070005, // General access denied error
    E_FAIL                            = 0x80004005, // Unspecified failure
    E_HANDLE                          = 0x80070006, // Handle that is not valid
    E_INVALIDARG                      = 0x80070057, // One or more arguments are not valid
    E_NOINTERFACE                     = 0x80004002, // No such interface supported
    E_NOTIMPL                         = 0x80004001, // Not implemented
    E_OUTOFMEMORY                     = 0x8007000E, // Failed to allocate necessary memory
    E_POINTER                         = 0x80004003, // Pointer that is not valid
    E_UNEXPECTED                      = 0x8000FFFF, //   Unexpected failure
    S_RFID_STATUS_OK                  = 0x00040000, // RFID Success
    E_RFID_ERROR_ALREADY_OPEN         = 0x8004D8F1, // Attempted to open a radio that is already open
    E_RFID_ERROR_BUFFER_TOO_SMALL = 0x8004d8f2, //Buffer supplied is too small
    E_RFID_ERROR_FAILURE              = 0x8004d8f3, //General failure
    E_RFID_ERROR_DRIVER_LOAD          = 0x8004d8f4, //Failed to load radio bus driver
    E_RFID_ERROR_DRIVER_MISMATCH      = 0x8004d8f5, //Library cannot use version of radio bus driver
```

E_RFID_ERROR_EMULATION_MODE = 0x8004d8f6, //Operation cannot be performed in emulation mode
E_RFID_ERROR_INVALID_ANTENNA = 0x8004d8f7, //Antenna number is invalid
E_RFID_ERROR_INVALID_HANDLE = 0x8004d8f8, //Radio handle provided is invalid
E_RFID_ERROR_INVALID_PARAMETER = 0x8004d8f9, //One of the parameters is invalid
E_RFID_ERROR_NO_SUCH_RADIO = 0x8004d8fa, //Attempted to open a non-existent radio
E_RFID_ERROR_NOT_INITIALIZED = 0x8004d8fb, //Library has not been successfully initialized
E_RFID_ERROR_NOT_SUPPORTED = 0x8004d8fc, //Function not supported
E_RFID_ERROR_OPERATION_CANCELLED = 0x8004d8fd, //Operation was cancelled by call to cancel operation, close radio, or shut down the library
E_RFID_ERROR_OUT_OF_MEMORY = 0x8004d8fe, //Library encountered an error allocating memory
E_RFID_ERROR_RADIO_BUSY = 0x8004d8ff, //The operation cannot be performed, radio is busy
E_RFID_ERROR_RADIO_FAILURE = 0x8004d900, //The underlying radio module encountered an error
E_RFID_ERROR_RADIO_NOT_PRESENT = 0x8004d901, //The radio has been detached from the system
E_RFID_ERROR_CURRENTLY_NOT_ALLOWED = 0x8004d902, //library function is not allowed at this time.
E_RFID_ERROR_RADIO_NOT_RESPONDING = 0x8004d903 //The radio module's MAC firmware is not responding to requests.
}; .

Description:

This enumerates the Success / Error status.

RFID_PACKET_TYPE

Prototype:

```
public enum RFID_PACKET_TYPE:uint{
    RFID_PACKET_TYPE_COMMAND_BEGIN = 0x0000,
    RFID_PACKET_TYPE_COMMAND_END ,
    RFID_PACKET_TYPE_ANTENNA_CYCLE_BEGIN ,
    RFID_PACKET_TYPE_ANTENNA_BEGIN ,
    RFID_PACKET_TYPE_18K6C_INVENTORY_ROUND_BEGIN,
    RFID_PACKET_TYPE_18K6C_INVENTORY,
    RFID_PACKET_TYPE_18K6C_TAG_ACCESS,
    RFID_PACKET_TYPE_ANTENNA_CYCLE_END,
    RFID_PACKET_TYPE_ANTENNA_END,
    RFID_PACKET_TYPE_18K6C_INVENTORY_ROUND_END,
    RFID_PACKET_TYPE_INVENTORY_CYCLE_BEGIN,
    RFID_PACKET_TYPE_INVENTORY_CYCLE_END,
    RFID_PACKET_TYPE_CARRIER_INFO,
    RFID_PACKET_TYPE_NONCRITICAL_FAULT = 0x2000
  };
```

Description:
These are message types for the Rfid Packets.

RFID_MSGID

Prototype:

```
    public enum RFID_MSGID : uint {
    RFID_REQUEST_TYPE_MSGID_Startup = RfidSp.WM_USER +0x0040,
//=RFID_REQUEST_TYPE_MSGID_START,
    RFID_REQUEST_TYPE_MSGID_Shutdown,
    RFID_REQUEST_TYPE_MSGID_RetrieveAttachedRadiosList,
    RFID_REQUEST_TYPE_MSGID_RadioOpen,
    RFID_REQUEST_TYPE_MSGID_RadioClose,
    RFID_REQUEST_TYPE_MSGID_RadioSetConfigurationParameter,
    RFID_REQUEST_TYPE_MSGID_RadioGetConfigurationParameter,
    RFID_REQUEST_TYPE_MSGID_RadioSetOperationMode,
    RFID_REQUEST_TYPE_MSGID_RadioGetOperationMode,
    RFID_REQUEST_TYPE_MSGID_RadioSetPowerState,
    RFID_REQUEST_TYPE_MSGID_RadioGetPowerState,
    RFID_REQUEST_TYPE_MSGID_RadioSetCurrentLinkProfile,
    RFID_REQUEST_TYPE_MSGID_RadioGetCurrentLinkProfile,
    RFID_REQUEST_TYPE_MSGID_RadioGetLinkProfile,
    RFID_REQUEST_TYPE_MSGID_RadioWriteLinkProfileRegister,
    RFID_REQUEST_TYPE_MSGID_RadioReadLinkProfileRegister,
    RFID_REQUEST_TYPE_MSGID_AntennaPortGetStatus,
    RFID_REQUEST_TYPE_MSGID_AntennaPortSetState,
    RFID_REQUEST_TYPE_MSGID_AntennaPortSetConfiguration,
    RFID_REQUEST_TYPE_MSGID_AntennaPortGetConfiguration,
    RFID_REQUEST_TYPE_MSGID_18K6CSetSelectCriteria,
    RFID_REQUEST_TYPE_MSGID_18K6CGetSelectCriteria,
    RFID_REQUEST_TYPE_MSGID_18K6CSetPostMatchCriteria,
    RFID_REQUEST_TYPE_MSGID_18K6CGetPostMatchCriteria,
    RFID_REQUEST_TYPE_MSGID_18K6CSetQueryTagGroup,
    RFID_REQUEST_TYPE_MSGID_18K6CGetQueryTagGroup,
    RFID_REQUEST_TYPE_MSGID_18K6CSetCurrentSingulationAlgorithm,
    RFID_REQUEST_TYPE_MSGID_18K6CGetCurrentSingulationAlgorithm,
    RFID_REQUEST_TYPE_MSGID_18K6CSetQueryParameters,
    RFID_REQUEST_TYPE_MSGID_18K6CGetQueryParameters,
    RFID_REQUEST_TYPE_MSGID_18K6CTagInventory,
```



```
RFID_REQUEST_TYPE_MSGID_18K6CTagRead,
RFID_REQUEST_TYPE_MSGID_18K6CTagWrite,
RFID_REQUEST_TYPE_MSGID_18K6CTagKill,
RFID_REQUEST_TYPE_MSGID_18K6CTagLock,
    RFID_REQUEST_TYPE_MSGID_RadioCancelOperation,
    RFID_REQUEST_TYPE_MSGID_RadioAbortOperation,
RFID_REQUEST_TYPE_MSGID_RadioSetResponseDataMode,
RFID_REQUEST_TYPE_MSGID_RadioGetResponseDataMode,
RFID_REQUEST_TYPE_MSGID_MacUpdateFirmware,
RFID_REQUEST_TYPE_MSGID_MacGetVersion,
 RFID_REQUEST_TYPE_MSGID_MacReadOemData,
 RFID_REQUEST_TYPE_MSGID_MacWriteOemData,
RFID_REQUEST_TYPE_MSGID_MacReset,
RFID_REQUEST_TYPE_MSGID_MacClearError,
 RFID_REQUEST_TYPE_MSGID_MacBypassWriteRegister,
 RFID_REQUEST_TYPE_MSGID_MacBypassReadRegister,
 RFID_REQUEST_TYPE_MSGID_MacSetRegion,
 RFID_REQUEST_TYPE_MSGID_MacGetRegion,
RFID_REQUEST_TYPE_MSGID_RadioSetGpioPinsConfiguration,
RFID_REQUEST_TYPE_MSGID_RadioGetGpioPinsConfiguration,
RFID_REQUEST_TYPE_MSGID_RadioReadGpioPins,
RFID_REQUEST_TYPE_MSGID_RadioWriteGpioPins,
RFID_REQUEST_TYPE_MSGID_END = RFID_REQUEST_TYPE_MSGID_RadioWriteGpioPins,
//////// 43 Request ACK MsgId
RFID_REQEND_TYPE_MSGID_START = RFID_REQUEST_TYPE_MSGID_END + 0x01,
RFID_REQEND_TYPE_MSGID_Startup = RFID_REQEND_TYPE_MSGID_START,
RFID_REQEND_TYPE_MSGID_Shutdown,
RFID_REQEND_TYPE_MSGID_RetrieveAttachedRadiosList,
RFID_REQEND_TYPE_MSGID_RadioOpen,
RFID_REQEND_TYPE_MSGID_RadioClose,
RFID_REQEND_TYPE_MSGID_RadioSetConfigurationParameter,
RFID_REQEND_TYPE_MSGID_RadioGetConfigurationParameter,
RFID_REQEND_TYPE_MSGID_RadioSetOperationMode,
RFID_REQEND_TYPE_MSGID_RadioGetOperationMode,
RFID_REQEND_TYPE_MSGID_RadioSetPowerState,
RFID_REQEND_TYPE_MSGID_RadioGetPowerState,
RFID_REQEND_TYPE_MSGID_RadioSetCurrentLinkProfile,
RFID_REQEND_TYPE_MSGID_RadioGetCurrentLinkProfile,
```

RFID_REQEND_TYPE_MSGID_RadioGetLinkProfile,
RFID_REQEND_TYPE_MSGID_RadioWriteLinkProfileRegister,
RFID_REQEND_TYPE_MSGID_RadioReadLinkProfileRegister,
RFID_REQEND_TYPE_MSGID_AntennaPortGetStatus,
RFID_REQEND_TYPE_MSGID_AntennaPortSetState,
RFID_REQEND_TYPE_MSGID_AntennaPortSetConfiguration,
RFID_REQEND_TYPE_MSGID_AntennaPortGetConfiguration,
RFID_REQEND_TYPE_MSGID_18K6CSetQueryTagGroup,
RFID_REQEND_TYPE_MSGID_18K6CGetQueryTagGroup,
RFID_REQEND_TYPE_MSGID_18K6CSetCurrentSingulationAlgorithm,
RFID_REQEND_TYPE_MSGID_18K6CGetCurrentSingulationAlgorithm,
RFID_REQEND_TYPE_MSGID_18K6CSetSelectCriteria,
RFID_REQEND_TYPE_MSGID_18K6CGetSelectCriteria,
RFID_REQEND_TYPE_MSGID_18K6CSetPostMatchCriteria,
RFID_REQEND_TYPE_MSGID_18K6CGetPostMatchCriteria,
RFID_REQEND_TYPE_MSGID_18K6CSetQueryParameters,
RFID_REQEND_TYPE_MSGID_18K6CGetQueryParameters,
RFID_REQEND_TYPE_MSGID_18K6CTagInventory,
RFID_REQEND_TYPE_MSGID_18K6CTagRead,
RFID_REQEND_TYPE_MSGID_18K6CTagWrite,
RFID_REQEND_TYPE_MSGID_18K6CTagKill,
RFID_REQEND_TYPE_MSGID_18K6CTagLock,
RFID_REQEND_TYPE_MSGID_RadioCancelOperation,
RFID_REQEND_TYPE_MSGID_RadioAbortOperation,
RFID_REQEND_TYPE_MSGID_RadioSetResponseDataMode,
RFID_REQEND_TYPE_MSGID_RadioGetResponseDataMode,
RFID_REQEND_TYPE_MSGID_MacUpdateFirmware,
RFID_REQEND_TYPE_MSGID_MacGetVersion,
RFID_REQEND_TYPE_MSGID_MacReadOemData,
RFID_REQEND_TYPE_MSGID_MacWriteOemData,
RFID_REQEND_TYPE_MSGID_MacReset,
RFID_REQEND_TYPE_MSGID_MacClearError,
RFID_REQEND_TYPE_MSGID_MacBypassWriteRegister,
RFID_REQEND_TYPE_MSGID_MacBypassReadRegister,
RFID_REQEND_TYPE_MSGID_MacSetRegion,
RFID_REQEND_TYPE_MSGID_MacGetRegion,
RFID_REQEND_TYPE_MSGID_RadioSetGpioPinsConfiguration,
RFID_REQEND_TYPE_MSGID_RadioGetGpioPinsConfiguration,

```
    RFID_REQEND_TYPE_MSGID_RadioReadGpioPins,
    RFID_REQEND_TYPE_MSGID_RadioWriteGpioPins,
    RFID_REQEND_TYPE_MSGID_END = RFID_REQEND_TYPE_MSGID_RadioWriteGpioPins,
    //////// Packets
    RFID_PACKET_TYPE_MSGID_START
= RFID_REQEND_TYPE_MSGID_END + 0x01, /// 12 Pkt MsgId. 0x0000+ MSGID_START
    RFID_PACKET_TYPE_MSGID_COMMAND_BEGIN
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_COMMAND_BEGIN +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_COMMAND_END
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_COMMAND_END    +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_ANTENNA_CYCLE_BEGIN
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_ANTENNA_CYCLE_BEGIN    +
RFID_PACKET_TYPE_MSGID_START,
  RFID_PACKET_TYPE_MSGID_ANTENNA_BEGIN
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_ANTENNA_BEGIN +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_18K6C_INVENTORY_ROUND_BEGIN
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_18K6C_INVENTORY_ROUND_BEGIN +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_18K6C_INVENTORY
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_18K6C_INVENTORY +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_18K6C_TAG_ACCESS
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_18K6C_TAG_ACCESS    +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_ANTENNA_CYCLE_END
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_ANTENNA_CYCLE_END
+RFID_PACKET_TYPE_MSGID_START,
  RFID_PACKET_TYPE_MSGID_ANTENNA_END
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_ANTENNA_END            +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_18K6C_INVENTORY_ROUND_END
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_18K6C_INVENTORY_ROUND_END +
RFID_PACKET_TYPE_MSGID_START,
    RFID_PACKET_TYPE_MSGID_INVENTORY_CYCLE_BEGIN
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_INVENTORY_CYCLE_BEGIN    +
```



```
RFID_PACKET_TYPE_MSGID_START,
   RFID_PACKET_TYPE_MSGID_INVENTORY_CYCLE_END
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_INVENTORY_CYCLE_END +
RFID_PACKET_TYPE_MSGID_START,
   RFID_PACKET_TYPE_MSGID_CARRIER_INFO
= RFID_PACKET_TYPE.RFID_PACKET_TYPE_CARRIER_INFO          +
RFID_PACKET_TYPE_MSGID_START,
      // non for the diagnostics pkt., for the status pkt. 0x2000+ MSGID_START
RFID_PACKET_TYPE_MSGID_NONCRITICAL_FAULT =
RFID_PACKET_TYPE.RFID_PACKET_TYPE_NONCRITICAL_FAULT    +
RFID_PACKET_TYPE_MSGID_START,
   RFID_PACKET_TYPE_MSGID_END =
RFID_PACKET_TYPE_MSGID_NONCRITICAL_FAULT,

   RFIDMW_REQUEST_TYPE_MSGID_START= RFID_PACKET_TYPE_MSGID_END + 0x01,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_SetAllTaglist =
RFIDMW_REQUEST_TYPE_MSGID_START,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_AddATag,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_FindATag,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_ClearAllTaglist,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_UpdateAllTaglistToFile,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_GetUpdateTaglist,
      RFIDMW_REQUEST_TYPE_MSGID_TagInv_GetAllTaglist,
   RFIDMW_REQUEST_TYPE_MSGID_END    =
RFIDMW_REQUEST_TYPE_MSGID_TagInv_GetAllTaglist,
   RFIDMW_REQEND_TYPE_MSGID_START = RFIDMW_REQUEST_TYPE_MSGID_END +
0x01,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_SetAllTaglist    =
RFIDMW_REQEND_TYPE_MSGID_START,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_AddATag,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_FindATag,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_ClearAllTaglist,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_UpdateAllTaglistToFile ,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_GetUpdateTaglist,
      RFIDMW_REQEND_TYPE_MSGID_TagInv_GetAllTaglist,
   RFIDMW_REQEND_TYPE_MSGID_END    =
RFIDMW_REQEND_TYPE_MSGID_TagInv_GetAllTaglist
};
```

Description:
These are message types for the Rfid request, response & packet messages.
N.B. Rfid request message is not required by the application, request are done by calling the corresponding functions.

RFID_RADIO_OPERATION_MODE

Prototype:

```
public enum RFID_RADIO_OPERATION_MODE : uint {
    RFID_RADIO_OPERATION_MODE_CONTINUOUS,
    RFID_RADIO_OPERATION_MODE_NONCONTINUOUS
};
```

Description:
This is the operation mode of the radio.

RFID_RADIO_POWER_STATE

Prototype:

```
public enum RFID_RADIO_POWER_STATE : uint {
    RFID_RADIO_POWER_STATE_FULL,
    RFID_RADIO_POWER_STATE_STANDBY
};
```

Description:
This is the power state of the radio.

RFID_ANTENNA_PORT_STATE

Prototype:

```
public enum RFID_ANTENNA_PORT_STATE : uint {
    RFID_ANTENNA_PORT_STATE_DISABLED,
    RFID_ANTENNA_PORT_STATE_ENABLED
};
```

Description:
This gives the state of a logical antenna port.

RFID_18K6C_SELECTED

Prototype:

```
public enum RFID_18K6C_SELECTED : uint {
RFID_18K6C_SELECTED_ALL = 0,
RFID_18K6C_SELECTED_OFF = 2,
RFID_18K6C_SELECTED_ON = 3
```


};
Description:
This defines the states for SL flag of a tag.

RFID_18K6C_INVENTORY_SESSION
Prototype:
```
public enum RFID_18K6C_INVENTORY_SESSION : uint {
RFID_18K6C_INVENTORY_SESSION_S0 = 0,
RFID_18K6C_INVENTORY_SESSION_S1 = 1,
RFID_18K6C_INVENTORY_SESSION_S2 = 2,
RFID_18K6C_INVENTORY_SESSION_S3 = 3
};
```
Description:
This defines the valid states for a tag's ISO 18000-6C inventory flags.

RFID_18K6C_INVENTORY_SESSION_TARGET
Prototype:
```
public enum RFID_18K6C_INVENTORY_SESSION_TARGET : uint {
RFID_18K6C_INVENTORY_SESSION_TARGET_A = 0,
RFID_18K6C_INVENTORY_SESSION_TARGET_B = 1
};
```
Description:
This defines the valid states for a tag's ISO 18000-6C inventory flags.

RFID_18K6C_MODULATION_TYPE
Prototype:
```
public enum RFID_18K6C_MODULATION_TYPE : uint {
    RFID_18K6C_MODULATION_TYPE_DSB_ASK,
    RFID_18K6C_MODULATION_TYPE_SSB_ASK,
    RFID_18K6C_MODULATION_TYPE_PR_ASK
};
```
Description:
This defines ISO 18000-6C modulation types.

RFID_18K6C_DATA_0_1_DIFFERENCE
Prototype:
```
public enum RFID_18K6C_DATA_0_1_DIFFERENCE : uint {
    RFID_18K6C_DATA_0_1_DIFFERENCE_HALF_TARI,
```

RFID_18K6C_DATA_0_1_DIFFERENCE_ONE_TARI
};
Description:
This is the Tari between data zero.

RFID_18K6C_DIVIDE_RATIO
Prototype:
```
public enum RFID_18K6C_DIVIDE_RATIO : uint {
    RFID_18K6C_DIVIDE_RATIO_8,
    RFID_18K6C_DIVIDE_RATIO_64DIV3
};
```
Description:
This is the ISO 18000-6C divide ratios.

RFID_18K6C_MILLER_NUMBER
Prototype:
```
public enum RFID_18K6C_MILLER_NUMBER : uint {
    RFID_18K6C_MILLER_NUMBER_FM0,
    RFID_18K6C_MILLER_NUMBER_2,
    RFID_18K6C_MILLER_NUMBER_4,
    RFID_18K6C_MILLER_NUMBER_8
};
```
Description:
This is the ISO 18000-6C Miller encoding sub-carrier.

RFID_RADIO_PROTOCOL
Prototype:
```
public enum RFID_RADIO_PROTOCOL : uint {
    RFID_RADIO_PROTOCOL_ISO18K6C
};
```
Description:
The is the tag protocol.

RFID_18K6C_MEMORY_BANK
Prototype:
```
public enum RFID_18K6C_MEMORY_BANK : uint {
    RFID_18K6C_MEMORY_BANK_RESERVED,
    RFID_18K6C_MEMORY_BANK_EPC,
```



```
    RFID_18K6C_MEMORY_BANK_TID,
    RFID_18K6C_MEMORY_BANK_USER
};
```

Description:

This is the RFID tag's memory bank.

RFID_18K6C_TARGET

Prototype:

```
public enum RFID_18K6C_TARGET : uint {
    RFID_18K6C_TARGET_INVENTORY_S0,
    RFID_18K6C_TARGET_INVENTORY_S1,
    RFID_18K6C_TARGET_INVENTORY_S2,
    RFID_18K6C_TARGET_INVENTORY_S3,
    RFID_18K6C_TARGET_SELECTED
};
```

Description:

This defines the tag's flags that will be modified.

RFID_18K6C_ACTION

Prototype:

```
public enum RFID_18K6C_ACTION : uint {
    RFID_18K6C_ACTION_ASLINVA_DSLINVB,
    RFID_18K6C_ACTION_ASLINVA_NOTHING,
    RFID_18K6C_ACTION_NOTHING_DSLINVB,
    RFID_18K6C_ACTION_NSLINVS_NOTHING,
    RFID_18K6C_ACTION_DSLINVB_ASLINVA,
    RFID_18K6C_ACTION_DSLINVB_NOTHING,
    RFID_18K6C_ACTION_NOTHING_ASLINVA,
    RFID_18K6C_ACTION_NOTHING_NSLINVS
};
```

Description:

This is the action performed upon the tag populations (i.e, matching and non-matching) during the select operation.

The constants are named RFID_18K6C_ACTION_xxx_yyy where "xxx" is the action to be applied to matching tags and "yyy" is the action to be applied to non-matching tags.

Actions are:

ASL - Assert SL

INVA - Set inventoried flag to A

DSL - Deassert SL
INVB - Set inventoried flag to B
NSL - Negate SL
INVS - Switch inventoried flag (A -> B, B -> A)
NOTHING - Do nothing.

RFID_18K6C_SELECTED

Prototype:

```
public enum RFID_18K6C_SELECTED: uint {
    RFID_18K6C_SELECTED_ALL = 0,
    RFID_18K6C_SELECTED_OFF = 2,
    RFID_18K6C_SELECTED_ON    = 3
};
```

Description:

This is the states for a tag's SL flag.

RFID_18K6C_INVENTORY_SESSION

Prototype:

```
public enum RFID_18K6C_INVENTORY_SESSION : uint {
    RFID_18K6C_INVENTORY_SESSION_S0,
    RFID_18K6C_INVENTORY_SESSION_S1,
    RFID_18K6C_INVENTORY_SESSION_S2,
    RFID_18K6C_INVENTORY_SESSION_S3
};
```

Description:

This is the ISO 18000-6C inventory session flags that are available.

RFID_18K6C_INVENTORY_SESSION

Prototype:

```
public enum RFID_18K6C_INVENTORY_SESSION_TARGET : uint {
    RFID_18K6C_INVENTORY_SESSION_TARGET_A,
    RFID_18K6C_INVENTORY_SESSION_TARGET_B
};
```

Description:

This is the valid states for a tag's ISO 18000-6C inventory flags.

RFID_18K6C_SINGULATION_ALGORITHM

Prototype:

```
public enum RFID_18K6C_SINGULATION_ALGORITHM : uint{
    RFID_18K6C_SINGULATION_ALGORITHM_FIXEDQ                  = 0,
    RFID_18K6C_SINGULATION_ALGORITHM_DYNAMICQ                = 1,
    RFID_18K6C_SINGULATION_ALGORITHM_DYNAMICQ_ADJUST         = 2,
    RFID_18K6C_SINGULATION_ALGORITHM_DYNAMICQ_THRESH         = 3
};
```

Description:

This is the valid singulation algorithms.

RFID_18K6C_WRITE_TYPE

Prototype:

```
public enum RFID_18K6C_WRITE_TYPE: uint {
    RFID_18K6C_WRITE_TYPE_SEQUENTIAL,
    RFID_18K6C_WRITE_TYPE_RANDOM
};
```

Description:

This is the type of tag write operation to be performed..

RFID_18K6C_TAG_PWD_PERM

Prototype:

```
public enum RFID_18K6C_TAG_PWD_PERM : uint {
    RFID_18K6C_TAG_PWD_PERM_ACCESSIBLE,
    RFID_18K6C_TAG_PWD_ALWAYS_ACCESSIBLE,
    RFID_18K6C_TAG_PWD_SECURED_ACCESSIBLE,
    RFID_18K6C_TAG_PWD_ALWAYS_NOT_ACCESSIBLE,
    RFID_18K6C_TAG_PWD_PERM_NO_CHANGE
};
```

Description:

This is the ISO 18000-6C tag password permission values..

RFID_18K6C_TAG_MEM_PERM

Prototype:

```
public enum RFID_18K6C_TAG_MEM_PERM : uint {
    RFID_18K6C_TAG_MEM_PERM_WRITEABLE,
    RFID_18K6C_TAG_MEM_ALWAYS_WRITEABLE,
    RFID_18K6C_TAG_MEM_SECURED_WRITEABLE,
    RFID_18K6C_TAG_MEM_ALWAYS_NOT_WRITEABLE,
```



```
    RFID_18K6C_TAG_MEM_NO_CHANGE
};
```

Description:

This is the ISO 18000-6C tag memory bank permission values.

RFID_RESPONSE_TYPE

Prototype:

```
public enum RFID_RESPONSE_TYPE: uint {
    RFID_RESPONSE_TYPE_DATA = 0xFFFFFFFF
};
```

Description:

This is the tag-access operation response type.

RFID_RESPONSE_MODE

Prototype:

```
public enum RFID_RESPONSE_MODE: uint {
    RFID_RESPONSE_MODE_COMPACT      = 0x00000001,
    RFID_RESPONSE_MODE_NORMAL       = 0x00000003,
    RFID_RESPONSE_MODE_EXTENDED     = 0x00000007
};
```

Description:

This is the tag-access operation data-reporting mode.

RFID_MAC_RESET_TYPE

Prototype:

```
public enum RFID_MAC_RESET_TYPE: uint {
    RFID_MAC_RESET_TYPE_SOFT
};
```

Description:

This is the types of resets available on the MAC.

RFID_MAC_REGION

Prototype:

```
public enum RFID_MAC_REGION: uint{
    RFID_MAC_REGION_FCC_GENERIC,
    RFID_MAC_REGION_ETSI_GENERIC
};
```

Description:

This is the regulatory mode regions.

RFID_RADIO_GPIO_PIN

Prototype:

```
public enum RFID_RADIO_GPIO_PIN: uint {
    RFID_RADIO_GPIO_PIN_0     = 0x00000001<< 0, // SET_BIT(0),
    RFID_RADIO_GPIO_PIN_1     = 0x00000001<< 1, // SET_BIT(1),
    RFID_RADIO_GPIO_PIN_2     = 0x00000001<< 2, // SET_BIT(2),
    RFID_RADIO_GPIO_PIN_3     = 0x00000001<< 3, // SET_BIT(3),
    RFID_RADIO_GPIO_PIN_4     = 0x00000001<< 4, // SET_BIT(4),
    RFID_RADIO_GPIO_PIN_5     = 0x00000001<< 5, // SET_BIT(5),
    RFID_RADIO_GPIO_PIN_6     = 0x00000001<< 6, // SET_BIT(6),
    RFID_RADIO_GPIO_PIN_7     = 0x00000001<< 7, // SET_BIT(7),
    RFID_RADIO_GPIO_PIN_8     = 0x00000001<< 8, // SET_BIT(8),
    RFID_RADIO_GPIO_PIN_9     = 0x00000001<< 9, // SET_BIT(9),
    RFID_RADIO_GPIO_PIN_10    = 0x00000001<<10, // SET_BIT(10),
    RFID_RADIO_GPIO_PIN_11    = 0x00000001<<11, // SET_BIT(11),
    RFID_RADIO_GPIO_PIN_12    = 0x00000001<<12, // SET_BIT(12),
    RFID_RADIO_GPIO_PIN_13    = 0x00000001<<13, // SET_BIT(13),
    RFID_RADIO_GPIO_PIN_14    = 0x00000001<<14, // SET_BIT(14),
    RFID_RADIO_GPIO_PIN_15    = 0x00000001<<15, // SET_BIT(15),
    RFID_RADIO_GPIO_PIN_16    = 0x00000001<<16, // SET_BIT(16),
    RFID_RADIO_GPIO_PIN_17    = 0x00000001<<17, // SET_BIT(17),
    RFID_RADIO_GPIO_PIN_18    = 0x00000001<<18, // SET_BIT(18),
    RFID_RADIO_GPIO_PIN_19    = 0x00000001<<19, // SET_BIT(19),
    RFID_RADIO_GPIO_PIN_20    = 0x00000001<<20, // SET_BIT(20),
    RFID_RADIO_GPIO_PIN_21    = 0x00000001<<21, // SET_BIT(21),
    RFID_RADIO_GPIO_PIN_22    = 0x00000001<<22, // SET_BIT(22),
    RFID_RADIO_GPIO_PIN_23    = 0x00000001<<23, // SET_BIT(23),
    RFID_RADIO_GPIO_PIN_24    = 0x00000001<<24, // SET_BIT(24),
    RFID_RADIO_GPIO_PIN_25    = 0x00000001<<25, // SET_BIT(25),
    RFID_RADIO_GPIO_PIN_26    = 0x00000001<<26, // SET_BIT(26),
    RFID_RADIO_GPIO_PIN_27    = 0x00000001<<27, // SET_BIT(27),
    RFID_RADIO_GPIO_PIN_28    = 0x00000001<<28, // SET_BIT(28),
    RFID_RADIO_GPIO_PIN_29    = 0x00000001<<29, // SET_BIT(29),
    RFID_RADIO_GPIO_PIN_30    = 0x00000001<<30, // SET_BIT(30),
    RFID_RADIO_GPIO_PIN_31    = 0x80000000 //1<<31    // SET_BIT(31)
};
```

Description:
This is the bit mask values for the radio module GPIO pins.

RFID_Startup_EMULATION_FLAG
Prototype:
```
public enum RFID_Startup_EMULATION_FLAG{
    RFID_FLAG_LIBRARY_EMULATION = 0x00000001
};
```
Description:
This is the flag for the RFID_Startup function.
User can set to system emulation mode during RfidStartup.

RFID_RadioOpen_EMULATION_FLAG
Prototype:
```
public enum RFID_RadioOpen_EMULATION_FLAG{
    RFID_FLAG_MAC_EMULATION        = 0x00000001
};
```
Description:
This is the flag for the RFID_RadioOpen function.
In system emulation mode, user can set to MAC emulation mode while calling RadioOpen .

RFID_18K6CTag_FLAG
Prototype:
```
public enum RFID_18K6CTag_FLAG{
    RFID_FLAG_PERFORM_SELECT            = 0x00000001,
    RFID_FLAG_PERFORM_POST_MATCH        = 0x00000002
};
```
Description:
This is the flag for the RFID_18K6CTag* functions.

4. Structures:

RFID_Startup_T
Prototype:
```
public struct RFID_Startup_T{
    public RFID_VERSION           LibraryVersion; //[out] RFID_VERSION*
    public UInt32                 flags;
    public HRESULT_RFID_STATUS    status;    //[ret]
```

```
    };
```

Fields:

[out] LibraryVersion:

[in] flags: emulation mode or live.

Description:

This is the data structure for f_RfidDev_Startup operation.

RFID_Shutdown_T

Prototype:

```
public struct RFID_Shutdown_T{
    public HRESULT_RFID_STATUS    status;
    };
```

Fields:

--

Description:

This is the data structure for f_RfidDev_Shutdown operation.

RFID_RetrieveAttachedRadiosList_T

Prototype:

```
public struct RFID_RetrieveAttachedRadiosList_T{
public RFID_RADIO_ENUM_T        radio_enum;
    public UInt32                           flags;
    public HRESULT_RFID_STATUS    status;
    };
```

Fields:

[in] radio_enum: enum of radio object.

[in] flags: 0. reserved.

Description:

This is the data structure for f_RfidDev_RetrieveAttachedRadiosList operation.

CS101 only has a single radio object for RFID_RADIO_ENUM, array of objects is not required.

RFID_RadioOpen_T

Prototype:

```
public struct RFID_RadioOpen_T{
    public UInt32                     cookie;
    public RFID_RADIO_HANDLE          handle; //[out]
    public UInt32                     flags;
    public HRESULT_RFID_STATUS        status;
```

```
};
```

Fields:

[in] cookie: cookie in radio_enum above.

[out] handle: the rfid_handle to be returned.

[in] flags: MAC emulation mode or live.

Description:

This is the data structure for f_RfidDev_RadioOpen operation.

RFID_RadioClose_T

Prototype:

```
public struct RFID_RadioClose_T{
    public RFID_RADIO_HANDLE        handle;
    public HRESULT_RFID_STATUS      status;
    };
```

Fields:

--

Description:

This is the data structure for f_RfidDev_RadioClose operation.

RFID_RadioGetSetConfigurationParameter_T

Prototype:

```
public struct RFID_RadioGetSetConfigurationParameter_T{
    public RFID_RADIO_HANDLE        handle;
    public UInt16                   parameter;
    public UInt32                   value;   //[out/in]
    public HRESULT_RFID_STATUS      status;
    };
```

Fields:

[in] parameter: The parameter address to set.

[out/in] value: The value content to get/set.

Description:

This is the data structure for

f_RfidDev_RadioGetConfigurationParameter /

f_RfidDev_RadioSetConfigurationParameter operation.

RFID_RadioGetSetOperationMode_T

Prototype:

```
public struct RFID_RadioGetSetOperationMode_T{
```



```
    public RFID_RADIO_HANDLE              handle;
    public RFID_RADIO_OPERATION_MODE      mode;
    public HRESULT_RFID_STATUS            status;
    };
```

Fields:

[out/in] mode: continuous or non-continuous.

Description:

This is the data structure for
f_RfidDev_RadioGetOperationMode /
f_RfidDev_RadioSetOperationMode operation.

RFID_RadioGetSetPowerState_T

Prototype:

```
public struct RFID_RadioGetSetPowerState_T{
    public RFID_RADIO_HANDLE              handle;
    public RFID_RADIO_POWER_STATE         state;
    public HRESULT_RFID_STATUS            status;
    };
```

Fields:

[out/in] state: power on/off state.

Description:

This is the data structure for
f_RfidDev_RadioGetPowerState /
f_RfidDev_RadioSetPowerState operation.

RFID_RadioGetSetCurrentLinkProfile_T

Prototype:

```
public struct RFID_RadioGetSetCurrentLinkProfile_T{
    public RFID_RADIO_HANDLE        handle;
    public UInt32                   profile;
    public HRESULT_RFID_STATUS      status;
    };
```

Fields:

[out/in] profile: profile 0--5 .

Description:

This is the data structure for
f_RfidDev_RadioGetCurrentLinkProfile /
f_RfidDev_RadioSetCurrentLinkProfile operation.

RFID_RadioGetLinkProfile_T

Prototype:

```
public struct RFID_RadioGetLinkProfile_T{
        public RFID_RADIO_HANDLE              handle;
        public UInt32                          profile;
        public RFID_RADIO_LINK_PROFILE         linkProfileInfo; //[out]
        public HRESULT_RFID_STATUS             status;
        };
```

Fields:

[in] profile: profile 0-- 5.

[out/in] linkProfileInfo: link profile information.

Description:

This is the data structure for f_RfidDev_RadioGetLinkProfile operation.

RFID_RadioReadWriteLinkProfileRegister_T

Prototype:

```
public struct RFID_RadioReadWriteLinkProfileRegister_T{
        public RFID_RADIO_HANDLE              handle;
        public UInt32                          profile;
        public UInt16                          address;
        public UInt16                          value;
        public HRESULT_RFID_STATUS             status;
        };
```

Fields:

[in] profile: profile id (0--5) for the link-profile register to be accessed.

[in] address: address of the register.

[out/in] value: content.

Description:

This is the data structure for f_RfidDev_RadioRead(/Write)LinkProfileRegister operation.

RFID_AntennaPortGetStatus_T

Prototype:

```
public struct RFID_AntennaPortGetStatus_T{
        public RFID_RADIO_HANDLE                    handle;
        public UInt32                                antennaPort;
        public RFID_ANTENNA_PORT_STATUS              portStatus;     //[out]
        public HRESULT_RFID_STATUS                   status;
```

```
};
```

Fields:

[in] antennaPort: always = 0 for CS101.

[out] portStatus: enabled/disabled.

Description:

This is the data structure for f_RfidDev_AntennaPortGetStatus operation.

RFID_AntennaPortSetState_T

Prototype:

```
public struct RFID_AntennaPortSetState_T{
    public RFID_RADIO_HANDLE            handle;
    public UInt32                       antennaPort;
    public RFID_ANTENNA_PORT_STATE      state;
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] antennaPort: always = 0 for CS101.

[in] state: enabled / disabled.

Description:

This is the data structure for f_RfidDev_AntennaPortSetState operation.

RFID_AntennaPortGetSetConfiguration_T

Prototype:

```
public struct RFID_AntennaPortGetSetConfiguration_T{
    public RFID_RADIO_HANDLE            handle;
    public UInt32                       antennaPort;
    public RFID_ANTENNA_PORT_CONFIG     config; // [const struct*]
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] antennaPort: always 0 for CS101 .

[in/out] config: the structure to be configured.

Description:

This is the data structure for

f_RfidDev_AntennaPortGetConfiguration /

f_RfidDev_AntennaPortSetConfiguration operation.

RFID_18K6CSetSelectCriteria__T

Prototype:

```
public struct RFID_18K6CSetSelectCriteria__T{
        public RFID_RADIO_HANDLE                handle;
        public UInt32                           countCriteria;
        public RFID_18K6C_SELECT_CRITERIA       criteria; //[in] const*
        public UInt32                           flags;
        public HRESULT_RFID_STATUS              status;
        };
```

Fields:

[in] countCriteria: criteria count.

[in] criteria: criteria to set.

[in] flags: flags.

Description:

This is the data structure for f_RfidDev_18K6CSetSelectCriteria… operation.

RFID_18K6CGetSelectCriteria__T

Prototype:

```
public struct RFID_18K6CGetSelectCriteria__T{
        public RFID_RADIO_HANDLE                handle;
        public UInt32                           countCriteria;
        public RFID_18K6C_SELECT_CRITERIA criteria;
        public HRESULT_RFID_STATUS              status;
        };
```

Fields:

[in] countCriteria: criteria count.

[in] criteria: criteria to get.

Description:

This is the data structure for f_RfidDev_18K6CGetSelectCriteria… operation.

RFID_18K6CSetPostMatchCriteria__T

Prototype:

```
public struct RFID_18K6CSetPostMatchCriteria__T{
        public RFID_RADIO_HANDLE                        handle;
        public UInt32                                   countCriteria;
public RFID_18K6C_SINGULATION_CRITERIA     criteria; //[in] const*
        public UInt32                                   flags;
        public HRESULT_RFID_STATUS                      status;
        };
```

Fields:

[in] countCriteria: criteria count.

[in] criteria: criteria to set.

Description:

This is the data structure for f_RfidDev_18K6CSetPostMatchCriteria… operation.

RFID_18K6CGetPostMatchCriteria__T

Prototype:

```
public struct RFID_18K6CGetPostMatchCriteria__T{
        public RFID_RADIO_HANDLE                    handle;
        public UInt32                               countCriteria;
public RFID_18K6C_SINGULATION_CRITERIA     criteria;
        public HRESULT_RFID_STATUS                  status;
        };
```

Fields:

[in] countCriteria: criteria count.

[in] criteria: criteria to get.

Description:

This is the data structure for f_RfidDev_18K6CGetPostMatchCriteria… operation.

RFID_18K6CGetSetQueryTagGroup_T

Prototype:

```
public struct RFID_18K6CGetSetQueryTagGroup_T{
      public RFID_RADIO_HANDLE            handle;
      public RFID_18K6C_TAG_GROUP         group;
      public HRESULT_RFID_STATUS          status;
        };
```

Fields:

[out / in] Group: the tag group for subsequent tag-protocol operations applied to it. This is not NULL.

Description:

This is the data structure for f_RfidDev_18K6CGet(/Set)QueryTagGroup operation.

RFID_18K6CGetSetCurrentSingulationAlgorithm_T

Prototype:

```
public struct RFID_18K6CGetSetCurrentSingulationAlgorithm_T{
        public RFID_RADIO_HANDLE                        handle;
        public RFID_18K6C_SINGULATION_ALGORITHM         algorithm;
```

```
    public HRESULT_RFID_STATUS                    status;
    };
```

Fields:

[out/ in] Algorithm: enum of the Q type of interest.

0 = fixedQ;

1 = dynamicQ

2 = dynamicQAdjust

3 = dynamicQThresh;

Description:

This is the data structure for f_RfidDev_18K6CGet(/Set)CurrentSingulationAlgorithm operation.

RFID_18K6CGetSetSingulationAlgorithmParameters_T

Prototype:

```
public struct RFID_18K6CGetSetSingulationAlgorithmParameters_T{
    public RFID_RADIO_HANDLE                                handle;
public RFID_18K6C_SINGULATION_ALGORITHM_PARMS_T             singulationParms;
    public HRESULT_RFID_STATUS                              status;
    };
```

Fields:

[in] parms: singulation algorithm parameters

Description:

This is the data structure for f_RfidDev_18K6CGet(/Set)SingulationAlgorithmParameters operation.

RFID_18K6CSetQueryParameters_T

Prototype:

```
public struct RFID_18K6CSetQueryParameters_T{
    public RFID_RADIO_HANDLE          handle;
    public RFID_18K6C_QUERY_PARMS     parms; //[in] const*
    public UInt32                     flags;
    public HRESULT_RFID_STATUS        status;
    };
```

Fields:

[in] parms: structure containing the query parameters..

[in] flags: flags.

Description:

This is the data structure for f_RfidDev_18K6CSetQueryParameters operation.

RFID_18K6CGetQueryParameters_T

Prototype:

```
public struct RFID_18K6CGetQueryParameters_T{
    public RFID_RADIO_HANDLE            handle;
    public RFID_18K6C_QUERY_PARMS       parms;
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] parms: structure obtaining the query parameters..

Description:

This is the data structure for f_RfidDev_18K6CGetQueryParameters operation.

RFID_18K6CTagInventory_T

Prototype:

```
public struct RFID_18K6CTagInventory_T{
    public RFID_RADIO_HANDLE            handle;
    public RFID_18K6C_INVENTORY_PARMS    invenParms; //[in] const*
    public UInt32                       flags;
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] invenParms: INVENTORY_PARMS

[in] flags: . 0 | RFID_FLAG_PERFORM_SELECT |& RFID_FLAG_PERFORM_POST_MATCH

Description:

This is the data structure for f_RfidDev_18K6CTagInventory operation.

RFID_18K6CTagRead_T

Prototype:

```
public struct RFID_18K6CTagRead_T{
    public RFID_RADIO_HANDLE         handle;
    public RFID_18K6C_READ_PARMS     readParms; //[in] const*
    public UInt32                    flags;
    public HRESULT_RFID_STATUS       status;
    };
```

Fields:

[in] readParms: READ_PARMS

[in] flags: 0 | RFID_FLAG_PERFORM_SELECT |& RFID_FLAG_PERFORM_POST_MATCH

Description:

This is the data structure for f_RfidDev_18K6CTagRead operation.

RFID_18K6CTagWrite_T

Prototype:

```
public struct RFID_18K6CTagWrite_T{
    public RFID_RADIO_HANDLE            handle;
    public RFID_18K6C_WRITE_PARMS_T     writeParms; //[in] const*
    public UInt32                       flags;
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] writeParms: PARMS

[in] flags: 0 | RFID_FLAG_PERFORM_SELECT |& RFID_FLAG_PERFORM_POST_MATCH

Description:

This is the data structure for f_RfidDev_18K6CTagWrite operation.

RFID_18K6CTagKill_T

Prototype:

```
public struct RFID_18K6CTagKill_T{
    public RFID_RADIO_HANDLE            handle;
    public RFID_18K6C_KILL_PARMS        killParms; //[in] const
    public UInt32                       flags;
    public HRESULT_RFID_STATUS          status;
    };
```

Fields:

[in] killParms: PARMS

[in] flags: 0 | RFID_FLAG_PERFORM_SELECT |& RFID_FLAG_PERFORM_POST_MATCH

Description:

This is the data structure for f_RfidDev_18K6CTagKill operation.

RFID_18K6CTagLock_T

Prototype:

```
public struct RFID_18K6CTagLock_T{
    public RFID_RADIO_HANDLE            handle;
    public RFID_18K6C_LOCK_PARMS        lockParms; //[in] const*
    public UInt32                       flags;
    public HRESULT_RFID_STATUS          status;
    };
```


Fields:
[in] lockParms: PARMS
[in] flags: 0 | RFID_FLAG_PERFORM_SELECT |& RFID_FLAG_PERFORM_POST_MATCH
Description:
This is the data structure for f_RfidDev_18K6CTagLock operation.

RFID_RadioGetSetResponseDataMode_T
Prototype:
```
public struct RFID_RadioGetSetResponseDataMode_T {
    public RFID_RADIO_HANDLE      handle;
    public UInt32                 responseType; //RFID_RESPONSE_TYPE
    public UInt32                 responseMode; //[in] |[out] RFID_RESPONSE_MODE
    public HRESULT_RFID_STATUS    status;
    };
```
Fields:
[in] responseType: currently always RFID_RESPONSE_TYPE_DATA (0xffffffff)
[out,in] responseMode: **Compact**, Normal(default), extended.
Description:
This is the data structure for
f_RfidDev_RadioGetResponseDataMode /
f_RfidDev_RadioSetResponseDataMode operation.

RFID_MacUpdateFirmware_T
Prototype:
```
public struct RFID_MacUpdateFirmware_T {
    public RFID_RADIO_HANDLE      handle;
    public UInt32                 length;
    public UIntPtr                pImage; //const INT8U*
    public UInt32                 flags;
    public HRESULT_RFID_STATUS    status;
    };
```
Fields:
To Be Designed .
Description:
This is the data structure for f_RfidDev_MacUpdateFirmware operation.

RFID_MacGetVersion_T
Prototype:


```
public struct RFID_MacGetVersion_T {
    public RFID_RADIO_HANDLE      handle;
    public RFID_VERSION           version;
    public HRESULT_RFID_STATUS status;
    };
```

Fields:

[out] version: Rfid MAC version.

Description:

This is the data structure for f_RfidDev_MacGetVersion operation.

RFID_MacReadWriteOemData_T

Prototype:

```
public struct RFID_MacReadWriteOemData_T {
    public RFID_RADIO_HANDLE handle;
    public UInt32 address;
    public UInt32 count;
    public UIntPtr pData;    //UI32* ptr to an BYTE-array[count*4+1]
    public HRESULT_RFID_STATUS status;
    };
```

Fields:

To Be Designed.

Description:

This is the data structure for
f_RfidDev_MacReadOemData /
f_RfidDev_MacWriteOemData operation.

RFID_MacReset_T

Prototype:

```
public struct RFID_MacReset_T {
    public RFID_RADIO_HANDLE       handle;
    public RFID_MAC_RESET_TYPE    resetType; //
    public HRESULT_RFID_STATUS      status;
    };
```

Fields:

[in] resetType: soft_reset.

Description:

This is the data structure for f_RfidDev_MacReset operation.

RFID_MacClearError_T

Prototype:

```
public struct RFID_MacClearError_T {
        public RFID_RADIO_HANDLE        handle;
        public HRESULT_RFID_STATUS    status;
        };
```

Fields:

-- .

Description:

This is the data structure for f_RfidDev_MacClearError operation.

RFID_MacBypassReadWriteRegister_T

Prototype:

```
public struct RFID_MacBypassReadWriteRegister_T{
        public RFID_RADIO_HANDLE        handle;
        public UInt16                   address;
        public UInt16                   value;
        public HRESULT_RFID_STATUS    status;
        };
```

Fields:

[in]   address: UINT16 register address.

[out,in] value: UINT32 value.

Description:

This is the data structure for
f_RfidDev_MacBypassReadRegister /
f_RfidDev_MacBypassWriteRegister operation.

RFID_MacGetSetRegion_T

Prototype:

```
public struct RFID_MacGetSetRegion_T {
        public RFID_RADIO_HANDLE        handle;
        public UInt32                   region; //RFID_MAC_REGION
        public IntPtr                   pRegionConfig; //void*
        public HRESULT_RFID_STATUS    status;
        };
```

Fields:

To Be Designed.

Description:

This is the data structure for
f_RfidDev_MacGetRegion /
f_RfidDev_MacSetRegion operation.

RFID_RadioSetGpioPinsConfiguration_T

Prototype:

```
public struct RFID_RadioSetGpioPinsConfiguration_T {
    public RFID_RADIO_HANDLE      handle;
    public UInt32                 mask;
    public UInt32                 configuration;
    public HRESULT_RFID_STATUS    status;
    };
```

Fields:

[in] mask: bit mask of GPIO’s Ids.

[in] configuration: GPIO In or Out.

Description:

This is the data structure for f_RfidDev_RadioSetGpioPinsConfiguration operation.

RFID_RadioGetGpioPinsConfiguration_T

Prototype:

```
public struct RFID_RadioGetGpioPinsConfiguration_T {
    public RFID_RADIO_HANDLE      handle;
    public UInt32                 configuration;
    public HRESULT_RFID_STATUS    status;
    };
```

Fields:

configuration: bit masked status of the 32 GPIOs (as Input or Output pin).

Description:

This is the data structure for f_RfidDev_RadioGetGpioPinsConfiguration operation.

RFID_RadioReadWriteGpioPins_T

Prototype:

```
public struct RFID_RadioReadWriteGpioPins_T {
    public RFID_RADIO_HANDLE      handle;
    public UInt32                 mask;
    public UInt32                 value;
    public HRESULT_RFID_STATUS    status;
    };
```

Fields:

[in] mask: bit mask of GPIOs to be affected.

[out,in] value: values

Description:

This is the data structure for f_RfidDev_RadioReadGpioPins / f_RfidDev_RadioWriteGpioPins operation.

RFID_RadioCancelOperation_T

Prototype:

```
public struct RFID_RadioCancelOperation_T {
        public RFID_RADIO_HANDLE        handle;
        public UInt32                   flags;
        public HRESULT_RFID_STATUS      status;
        };
```

Fields:

[in] flags: unreferenced.

Description:

This is the data structure for f_RfidDev_RadioCancelOperation operation.

RFID_RadioAbortOperation_T

Prototype:

```
public struct RFID_RadioAbortOperation_T {
        public RFID_RADIO_HANDLE        handle;
        public UInt32                   flags;
        public HRESULT_RFID_STATUS      status;
        };
```

Fields:

[in] flags: unreferenced.

Description:

This is the data structure for f_RfidDev_RadioAbortOperation operation.

RFID_RadioIssueCommand_T

Prototype:

```
public struct RFID_RFID_RadioIssueCommand_T {
        public RFID_RADIO_HANDLE        handle;
        public UInt32                   command; //e.g. 0x17
        public HRESULT_RFID_STATUS      status;
```

};

Fields:

[in] flags: unreferenced.

Description:

This is the data structure for f_RfidDev_RadioAbortOperation operation.

Note: {RFID_PACKET_CALLBACK_FUNCTION Callback; void* context; INT32S* pCallbackCode;} is handled in rfid lib.

RFID_PACKETMSG_COMMON_T

Prototype:

```
public struct RFID_PACKETMSG_COMMON_T {
    public Byte     pkt_ver;    //INT8U   Packet specific version number
    public Byte     flags;      //        Packet specific flags
    public UInt16   pkt_type;   //        Packet type identifier
    public UInt16   pkt_len;    // Packet length preamble: number of 32-bit words that follow the
common
    public UInt16   res0;       // Reserved for future use
    };
```

Fields:

pkt_ver: Packet specific version number

flags: Packet specific flags

pkt_type: Packet type identifier

pkt_len: Packet length preamble: number of 32-bit words that follow the common

res0: Reserved for future use

Description:

This is the common packet preamble that contains fields that are common to all packets.

RFID_PACKETMSG_COMMAND_BEGIN_T

Prototype:

```
public struct RFID_PACKETMSG_COMMAND_BEGIN_T {
        public RFID_PACKETMSG_COMMON_T    cmn;
        public UInt32                     command;
        public UInt32                     ms_ctr;
    };
```

Fields:

cmn: The command context

command: The command for which the packet sequence is in response to

ms_ctr: Current millisecond counter.

Description:
This is the command-begin packet.

RFID_PACKETMSG_COMMAND_END_T

Prototype:

```
public struct RFID_PACKETMSG_COMMAND_END_T {
        public RFID_PACKETMSG_COMMON_T      cmn;
        public UInt32                       ms_ctr; // Current millisecond counter
        public UInt32                       status; // Command status indicator
    };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

Description:
This is the command-end packet.

RFID_PACKETMSG_ANTENNA_CYCLE_BEGIN_T

Prototype:

```
public struct RFID_PACKETMSG_ANTENNA_CYCLE_BEGIN_T {
    public RFID_PACKETMSG_COMMON_T      cmn;
    // No other packet specific fields
    };
```

Fields:

cmn: The command context.

Description:
This is the antenna-cycle-begin packet.

RFID_PACKETMSG_ANTENNA_CYCLE_END_T

Prototype:

```
public struct RFID_PACKETMSG_ANTENNA_CYCLE_END_T {
    public RFID_PACKETMSG_COMMON_T      cmn;
    // No other packet specific fields
    };
```

Fields:

cmn: The command context.

Description:
This is the antenna-cycle-begin packet..

RFID_PACKETMSG_ANTENNA_BEGIN_T

Prototype:

```
public struct RFID_PACKETMSG_ANTENNA_BEGIN_T {
    public RFID_PACKETMSG_COMMON_T cmn;    // The logical antenna ID
    public UInt32                  antenna;
    };
```

Fields:

cmn: The command context.

antenna: The antenna id .

Description:

This is the antenna-begin packet.

RFID_PACKETMSG_ANTENNA_END_T

Prototype:

```
public struct RFID_PACKETMSG_ANTENNA_END_T {
    public RFID_PACKETMSG_COMMON_T    cmn;// No other packet specific fields
    };
```

Fields:

cmn: The command context.

Description:

This is the antenna-end packet.

RFID_PACKETMSG_INVENTORY_CYCLE_BEGIN_T

Prototype:

```
public struct RFID_PACKETMSG_INVENTORY_CYCLE_BEGIN_T {
    public RFID_PACKETMSG_COMMON_T    cmn;
    public UInt32                     ms_ctr;.
}
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

Description:

This is the inventory-cycle-begin packet.

RFID_PACKETMSG_INVENTORY_CYCLE_END_T

Prototype:

```
public struct RFID_PACKETMSG_INVENTORY_CYCLE_END_T {
    public RFID_PACKETMSG_COMMON_T    cmn;
```

```
    public UInt32                ms_ctr;
    };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

Description:

This is the inventory-cycle-end packet.

RFID_PACKETMSG_18K6C_INVENTORY_ROUND_BEGIN_T

Prototype:

```
public struct RFID_PACKETMSG_18K6C_INVENTORY_ROUND_BEGIN_T {
        public RFID_PACKETMSG_COMMON_T     cmn;
        // No packet specific fields
    };
```

Fields:

cmn: The command context.

Description:

This is the data structure in the message.

.

RFID_PACKETMSG_18K6C_INVENTORY_ROUND_END_T

Prototype:

```
public struct RFID_PACKETMSG_18K6C_INVENTORY_ROUND_END_T {
        public RFID_PACKETMSG_COMMON_T     cmn;
        // No packet specific fields
    };
```

Fields:

cmn: The command context.

Description:

This is the ISO 18000-6C inventory round end packet.

RFID_PACKETMSG_18K6C_TAG_ACCESS_T

Prototype:

```
// Pointers and fixed size buffers may only be used in an unsafe context
public unsafe struct RFID_PACKETMSG_18K6C_TAG_ACCESS_T {
        public RFID_PACKETMSG_COMMON_T     cmn;
        public UInt32                ms_ctr; //UInt32
        public Byte                  command;   //INT8U
```

```
    public Byte            error_code; //INT8U Error code from tag access
    public UInt16          res0;     //public UInt16
    public UInt32          res1;     //UInt32
    public fixed UInt32
tag_data[ RfidSp.RFID_PACKET_18K6C_TAG_ACCESS__DATA_MAXSIZ ]; // Variable length
access data; 2
+16Byte for EPC Gen2
  };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

command: The command for which the packet sequence is in response to.

error_code: Error code from tag access: 0=NoError,
RFID_18K6C_TAG_ACCESS_CRC_INVALID;ACCESS_TIMEOUT;
BACKSCATTER_ERROR;ACCESS_ERROR

res0: reserved.

res1: reserved.

tag_data[]: Variable length access data; 2+16Byte for EPC Gen2

Description:

This is the ISO 18000-6C tag-access packet.

RFID_PACKETMSG_18K6C_INVENTORY_AND_DATA_T

Prototype:

```
public unsafe struct RFID_PACKETMSG_18K6C_INVENTORY_AND_DATA_T {
    public RFID_PACKETMSG_COMMON_T cmn;
    public UInt32          ms_ctr;
    public UInt16          rssi;     //public UInt16
    public UInt16          ana_ctrl1;
    public UInt32          res0;     //UInt32
    public fixed UInt32     inv_data[4];   //_18K6C_INVENTORY__DATA_MAXSIZ
  };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

rssi: RSSI

ana_ctrl1: The antenna control data

res0: Always 0; reserved.

inv_data[4]: integer array of data.

Description:

This is the ISO 18000-6C inventory packet.

It is Obsolete to the App, since RfidMw sends AddTag messages to App instead.

RFID_PACKETMSG_NONCRITICAL_FAULT_T

Prototype:

```
public struct RFID_PACKETMSG_NONCRITICAL_FAULT_T {
    public RFID_PACKETMSG_COMMON_T    cmn;
    public UInt32          ms_ctr;         // Current millisecond counter
    public UInt16          fault_type;     // Fault type
    public UInt16          fault_subtype;  // Fault subtype
    public UInt32          context;        // Context specific data for fault
    };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

fault_type: Fault type.

fault_subtype: Fault subtype.

context: Context specific data for fault.

Description:

This is the non-critical-fault packet.

RFID_PACKETMSG_CARRIER_INFO_T

Prototype:

```
public struct RFID_PACKETMSG_CARRIER_INFO_T {
    public RFID_PACKETMSG_COMMON_T    cmn;
    public UInt32          ms_ctr;         // Current millisecond counter
    public UInt32          plldivmult;     // current plldivmult setting
    public UInt16          chan;           // channel
    public UInt16          cw_flags;       // carrier flags
    };
```

Fields:

cmn: The command context.

ms_ctr: Current millisecond counter.

plldivmult: Current plldivmult setting.

chan: Channel number.

cw_flags: Carrier flags.

Description:

This contains info related to the transmitted carrier.

~~~~~~

(The following RfidSp structures are used by the structures above, see RfidSp_enums.cs)

RFID_VERSION

Prototype:

```
public struct RFID_VERSION {
        public UInt32                   major;
        public UInt32                   minor;
        public UInt32                   patch;
};
```

Fields:

major: The major version (i.e, in 1.x.x, the 1)

minor: The minor version (i.e., in x.1.x, the 1)

patch: The patch level (i.e., in x.x.1, the 1)

Description:

This represents the version information for components in the system.

RFID_RADIO_INFO

Prototype:

```
public struct RFID_RADIO_INFO {
        public UInt32                   length;
        public RFID_VERSION             driverVersion;
        public UInt32                   cookie;
        public UInt32                   idLength;
        public IntPtr                   pUniqueId;
};
```

Fields:

length: The length of the structure in bytes (=sizeof(RFID_RADIO_INFO)).

driverVersion: The version information for the radio's bus driver.

cookie: The unique cookie for the radio.

This cookie is passed to RFID_RadioOpen() when the application wishes to take control of the radio.

pUniqueId: A pointer to a byte array (ansi string) that contain the radio module's unique ID (=serial number).

Description:

This is used to represent the information for the attached radio.
~~~~~~

RFID_RADIO_ENUM_T

Prototype:

```
public struct RFID_RADIO_ENUM_T {
    public UInt32                 length;
    public UInt32                 totalLength;
    public UInt32                 countRadios;
    public RFID_RADIO_INFO        _RadioInfo;
};
```

Fields:

length: The length of the structure in bytes (= sizeof(RFID_RADIO_ENUM)).

totalLength: The total length, in bytes, of radio enumeration structure.

Application should fill in this with the length of the radio enumeration buffer.

countRadios: The number of radio objects that are attached to the system.

_RadioInfo: The RFID_RADIO_INFO structure.

Description:

This is used in the RetrieveAttachedRadiosList function.

The data that will be returned from a request to list the radios that are attached to the system

On CS101, a process should only able to get a single radio object.

RFID_RADIO_LINK_PROFILE_ISO18K6C_CONFIG

Prototype:

```
public struct RFID_RADIO_LINK_PROFILE_ISO18K6C_CONFIG {
    public UInt32                               length;
    public RFID_18K6C_MODULATION_TYPE           modulationType;
    public UInt32                               tari;
    public RFID_18K6C_DATA_0_1_DIFFERENCE       data01Difference;
    public UInt32                               pulseWidth;
    public UInt32                               rtCalibration;
    public UInt32                               trCalibration;
    public RFID_18K6C_DIVIDE_RATIO              divideRatio;
    public RFID_18K6C_MILLER_NUMBER             millerNumber;
    public UInt32                               trLinkFrequency;
    public UInt32                               varT2Delay;
    public UInt32                               rxDelay;
    public UInt32                               minT2Delay;
    public UInt32                               txPropagationDelay;
};
```

Fields:

length: The length of the structure in bytes.
modulationType: The modulation type used by the link profile.
tari: The duration, in nanoseconds, of the Tari.
data01Difference: The difference, in Taris, between a data zero and a data one.
pulseWidth: The duration, in nanoseconds, of the low-going portion of the radio-to-tag PIE symbol
rtCalibration: The width, in nanoseconds, of the radio-to-tag calibration.
trCalibration: The width, in nanoseconds, of the tag-to-radio calibration.
divideRatio: The divide ratio used.
millerNumber: The miller number (i.e., cycles per symbol);
trLinkFrequency: The tag-to-radio link frequency in Hz.
varT2Delay: The delay, in microseconds, inserted to ensure meeting the minimum T2 timing.
rxDelay:: The amount of time, in 48MHz cycles, a radio module will wait between transmitting and then attempting to receive the backscattered signal from tags.
minT2Delay: The minimum amount of ISO 18000-6C T2 time, in microseconds,
after receiving a tag response, before a radio may transmit again.
txPropagationDelay: The number of microseconds for a signal to propagate through the radio's transmit chain.
Description:
This is used in the RFID_RadioGetLinkProfile function.

RFID_RADIO_LINK_PROFILE

Prototype:

```
public struct RFID_RADIO_LINK_PROFILE {
    public UInt32                  length;
    public UInt32                  enabled; // BOOL32
    public UInt64                  profileId;
    public UInt32                  profileVersion;
    public RFID_RADIO_PROTOCOL     profileProtocol;
    public UInt32                  denseReaderMode; // BOOL32
    public UInt32                  widebandRssiSamples;
    public UInt32                  narrowbandRssiSamples;
    public UInt32                  realtimeRssiEnabled;
    public UInt32                  realtimeWidebandRssiSamples;
    public UInt32                  realtimeNarrowbandRssiSamples;
    public RFID_RADIO_LINK_PROFILE_ISO18K6C_CONFIG     iso18K6C;
};
```

Fields:
length: The length of the structure in bytes (= sizeof(RFID_RADIO_LINK_PROFILE)).

enabled: This indicates if the profile is active. A zero value indicates that the profile is inactive.
profileId: This is the identifier for the profile.
This, in combination with profileVersion, provides a unique identifier for the profile.
profileVersion: This is the version for the profile.
This field, in combination with profileId, provides a unique identifier for the profile.
profileProtocol: This is the tag protocol for which this profile was configured.
This value is the discriminator for the profileConfig union.
denseReaderMode: This indicates if the profile is a dense-reader-mode (DRM) profile.
A zero value indicates a non-DRM profile.
widebandRssiSamples: Number of wide band Receive Signal Strength Indication (RSSI) samples to be averaged.
narrowbandRssiSamples: Number of narrow band Receive Signal Strength Indication (RSSI) samples to be averaged.
realtimeRssiEnabled: Reserved for future use.
realtimeWidebandRssiSamples: Reserved for future use.
realtimeNarrowbandRssiSamples: Reserved for future use.
iso18K6C : This is the link profile configuration information.
A union of { RFID_RADIO_LINK_PROFILE_ISO18K6C_CONFIG iso18K6C;};
Description:
This is used in the RadioGetLinkProfile function.
This has the information about a radio link profile.
This is returned in repsonse to a request for link profile information.

RFID_ANTENNA_PORT_STATUS

Prototype:

```
public struct RFID_ANTENNA_PORT_STATUS{
    public UInt32                      length;
    public RFID_ANTENNA_PORT_STATE     state;
    public UInt32                      antennaSenseValue;
};
```

Fields:
length: The length of the structure in bytes (=sizeof(RFID_ANTENNA_PORT_STATUS)).
state: The state (enabled/disabled) of the antenna port.
antennaSenseValue: The last measurement of the antenna-sense resistor for the logical antenna port physical transmit antenna port.
Description:
This is the status of an logical antenna port.

RFID_ANTENNA_PORT_CONFIG

Prototype:

```
public struct RFID_ANTENNA_PORT_CONFIG {
    public UInt32    length;
    public UInt32    powerLevel;
    public UInt32    dwellTime;
    public UInt32    numberInventoryCycles;
    public UInt32    physicalRxPort;
    public UInt32    physicalTxPort;
    public UInt32    antennaSenseThreshold;
};
```

Fields:

length: The length of the structure in bytes (=sizeof(RFID_ANTENNA_PORT_CONFIG)).

powerLevel: The power level for the logical antenna port physical transmit antenna. This is specified in 0.1 (i.e., 1/10th) dBm.

dwellTime: The number of milliseconds to spend on this antenna port during a cycle. Zero indicates that antenna usage will be controlled by the numberInventoryCycles field.

numberInventoryCycles: The number of inventory rounds to perform with this antenna port. Zero indicates that the antenna usage will be controlled by the dwellTime field.

physicalRxPort: The physical receive antenna port associated with the logical antenna port (between 0 and 3).

physicalTxPort: The physical transmit antenna port associated with the logical antenna port (between 0 and 3).

antennaSenseThreshold: The measured resistance, specified in ohms, above which the antenna-sense resistance should be considered to be an open circuit: .

Description:

This is the configuration parameters for a logical antenna port.

RFID_18K6C_SELECT_MASK

Prototype:

```
public unsafe struct RFID_18K6C_SELECT_MASK {
    public RFID_18K6C_MEMORY_BANK    bank;
    public UInt32                    offset;
    public UInt32                    count;
    public fixed Byte     mask[RfidSp.RFID_18K6C_SELECT_MASK_BYTE_LEN];
};
```

Fields:

bank: The memory bank to match against

offset: The offset of the first bit to match
count: The number of bits in the mask
mask[RFID_18K6C_SELECT_MASK_BYTE_LEN]: The bit pattern to match.
Description:
This is the select mask for partitioning a tag population.

RFID_18K6C_SELECT_ACTION

Prototype:

```
public struct RFID_18K6C_SELECT_ACTION {
    public RFID_18K6C_TARGET      target;
    public RFID_18K6C_ACTION      action;
    public UInt32                 enableTruncate; //BOOL32
};
```

Fields:
target: The target affected by the action.
action: The actions to be performed on the tag populations (i.e., matching and non-matching.)
enableTruncate: truncate EPC when the tag is singulated.
    A non-zero value requestes that the EPC is truncated.
    A zero value requests the entire EPC.
Description:
This is the matching and non-matching action to take when a selection mask matches/doesn't match

RFID_18K6C_SELECT_CRITERION

Prototype:

```
public struct RFID_18K6C_SELECT_CRITERION {
    public RFID_18K6C_SELECT_MASK      mask;
    public RFID_18K6C_SELECT_ACTION    action;
};
```

Fields:
mask: The selection mask to test for RFID_18K6C_SELECT_MASK.
action: The actions to perform: .
Description:
This is the single selection criterion, as a combination of selection mask and action.

RFID_18K6C_SINGULATION_MASK

Prototype:

```
public unsafe struct RFID_18K6C_SINGULATION_MASK {
```



```
    public UInt32    offset;
    public UInt32    count;
    public fixed Byte    mask[ RfidSp.RFID_18K6C_SINGULATION_MASK_BYTE_LEN ];
};
```

Fields:

offset: Offset in bits, from the start of the EPC, of the first bit to match against the mask.

count: The number of bits in the mask. A length of zero causes all tags to match. If (offset + count) falls beyond the end of the mask, the tag is considered non-matching.

mask[]: The bit pattern to match.

Description:

This is a single post-singulation match mask.

RFID_18K6C_SINGULATION_CRITERION

Prototype:

```
public struct RFID_18K6C_SINGULATION_CRITERION {
    public UInt32 match;    //BOOL32
    public RFID_18K6C_SINGULATION_MASK mask;      //UCHAR[32]
};
```

Fields:

match: Indicates if the associated tag-protocol operation will be applied to matching or non-matching tags.

A non-zero value indicates that the tag-protocol operation is applied to matching tags.

A zero value of indicates that tag-protocol operation is applied to non- matching tags.

mask: The mask to be applied to EPC.

Description:

This is a single post-singulation match criterion.

RFID_18K6C_TAG_GROUP

Prototype:

```
public struct RFID_18K6C_TAG_GROUP{
    public RFID_18K6C_SELECTED                      selected;
    public RFID_18K6C_INVENTORY_SESSION             session;
    public RFID_18K6C_INVENTORY_SESSION_TARGET target;
};
```

Fields:

selected: The state of the SL flag.

session: The inventory session (S0, S1, etc.).

target: The state of the inventory session specified by the session field .
Description:
This specifies which tag population will be singulated.

RFID_18K6C_COMMON_PARMS
Prototype:
```
public struct RFID_18K6C_COMMON_PARMS{
    public UInt32 tagStopCount;
    public IntPtr pCallback;        //RFID_PACKET_CALLBACK_FUNCTION
    public IntPtr context;          //void*     //Nullable
    public IntPtr pCallbackCode;    //INT 32S*
};
```
Fields:
tagStopCount: The maximum number of tags to which the tag-protocol operation will be applied.
If this number is zero, then the operation is applied to all tags that match the selection, and optionally post-singulation, match criteria.
If this number is non-zero, the antenna-port dwell-time and inventory-round-count constraints still apply, however the operation will be prematurely terminated if the maximum number of tags have the tag-protocol operation applied to them.
pCallback: Callback function assigned in the library.
context: An value that is passed through unmodified to the application-specified callback function. It is usually = 0 .
pCallbackCode: A pointer to a 32-bit integer that upon return will contain the return code from the last call to the application-supplied callback function. This can be set to NULL.
Description:
This is the common parameters for ISO 18000-6C tag-protocol operation

RFID_18K6C_SINGULATION_FIXEDQ_PARMS
Prototype:
```
public struct RFID_18K6C_SINGULATION_FIXEDQ_PARMS{
    public UInt32 length;
    public UInt32 qValue;
    public UInt32 retryCount;
    public UInt32 toggleTarget; //BOOL32
    public UInt32 repeatUntilNoTags; //BOOL32
};
```
Fields:
length: The length of the structure in bytes.

When calling
RFID_18K6CSetQueryParameters,
RFID_18K6CSetSingulationAlgorithmParameters, or
RFID_18K6CGetSingulationAlgorithmParameters
the application must set this to sizeof(RFID_18K6C_SINGULATION_FIXEDQ_PARMS).
When calling RFID_18K6CGetQueryParameters, the library will fill in this field.
qValue: The Q value to use. Valid values are 0 to 15.
retryCount: Specifies the number of times to try another execution of the singulation algorithm for the specified session / target before either toggling the target (if toggleTarget is non-zero) or terminating the inventory / tag access operation. Valid values are 0-255.
toggleTarget: A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e., A to B or B to A) and another inventory cycle run.
A non-zero value indicates that the target should be toggled.
Note that if the target is toggled, retryCount and repeatUntilNoTags will also apply to the new target.
repeatUntilNoTags: A flag that indicates whether or not the singulation algorithm should continue performing inventory rounds until no tags are singulated.
A non-zero value indicates that, for each execution of the singulation algorithm, inventory rounds should be performed until no tags are singulated.
A zero value indicates that a single inventory round should be performed for each execution of the singulation algorithm.
Description:
This is the parameters for the fixed Q (i.e., RFID_18K6C_SINGULATION_ALGORITHM_FIXEDQ) algorithm.

RFID_18K6C_SINGULATION_DYNAMICQ_PARMS
Prototype:

```
public struct RFID_18K6C_SINGULATION_DYNAMICQ_PARMS{
    public UInt32 length;
    public UInt32 startQValue;
    public UInt32 minQValue;
    public UInt32 maxQValue;
    public UInt32 retryCount;
    public UInt32 maxQueryRepCount;
    public UInt32 toggleTarget; //BOOL32
};
```

Fields:
length: The length of the structure in bytes.
When calling

RFID_18K6CSetQueryParameters,
RFID_18K6CSetSingulationAlgorithmParameters, or
RFID_18K6CGetSingulationAlgorithmParameters
the application must set this to sizeof(RFID_18K6C_SINGULATION_DYNAMICQ_PARMS).
When calling RFID_18K6CGetQueryParameters, the library will fill in this field
startQValue: The starting Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
minQValue: The minimum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
maxQValue: The maximum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
retryCount: Specifies the number of times to try another execution of the singulation algorithm for the specified session/target before either toggling the target (if toggleTarget is non-zero) or terminating the inventory/tag access operation. Valid values are 0-255.
maxQueryRepCount: The maximum number of ISO 18000-6C QueryRep commands that will follow the ISO 18000-6C Query command during a single inventory round. Valid values are 0-255
toggleTarget: A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e.A to B or B to A) and another inventory cycle run.
A non-zero value indicates that the target should be toggled.
A zero value indicates that the target should not be toggled.
Note that if the target is toggled, retryCount and maxQueryReps will also apply to the new target.
Description:
This is for the parameters for the dynamic Q (i.e., RFID_18K6C_SINGULATION_ALGORITHM_DYNAMICQ) algorithm.

RFID_18K6C_SINGULATION_DYNAMICQ_ADJUST_PARMS

Prototype:

```
public struct RFID_18K6C_SINGULATION_DYNAMICQ_ADJUST_PARMS {
    public UInt32 length;
    public UInt32 startQValue;
    public UInt32 minQValue;
    public UInt32 maxQValue;
    public UInt32 retryCount;
    public UInt32 maxQueryRepCount;
    public UInt32 toggleTarget; //BOOL32
};
```

Fields:
length: sizeof(RFID_18K6C_SINGULATION_DYNAMICQ_ADJUST_PARMS) The length of the

structure in bytes.
When calling
RFID_18K6CSetQueryParameters,
RFID_18K6CSetSingulationAlgorithmParameters, or
RFID_18K6CGetSingulationAlgorithmParameters
the application must set this to sizeof(RFID_18K6C_SINGULATION_DYNAMICQ_PARMS).
When calling RFID_18K6CGetQueryParameters, the library will fill in this field
startQValue: The starting Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
minQValue: The minimum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
maxQValue: The maximum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
retryCount: Specifies the number of times to try another execution of the singulation algorithm for the specified session/target before either toggling the target (if toggleTarget is non-zero) or terminating the inventory/tag access operation. Valid values are 0-255.
maxQueryRepCount: The maximum number of ISO 18000-6C QueryRep commands that will follow the ISO 18000-6C Query command during a single inventory round. Valid values are 0-255
toggleTarget: A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e.A to B or B to A) and another inventory cycle run.
A non-zero value indicates that the target should be toggled.
A zero value indicates that the target should not be toggled.
Note that if the target is toggled, retryCount and maxQueryReps will also apply to the new target.
Description:
This is for the dynamic Q (adjust) algorithm operation;

RFID_18K6C_SINGULATION_DYNAMICQ_THRESH_PARMS

Prototype:

```
public struct RFID_18K6C_SINGULATION_DYNAMICQ_THRESH_PARMS {
    public UInt32 length;
    public UInt32 startQValue;
    public UInt32 minQValue;
    public UInt32 maxQValue;
    public UInt32 retryCount;
    public UInt32 maxQueryRepCount;
    public UInt32 toggleTarget; //BOOL32
    public UInt32 thresholdMultiplier;
```

};
Fields:
length: sizeof(RFID_18K6C_SINGULATION_DYNAMICQ_ADJUST_PARMS) The length of the structure in bytes.
When calling
RFID_18K6CSetQueryParameters,
RFID_18K6CSetSingulationAlgorithmParameters, or
RFID_18K6CGetSingulationAlgorithmParameters
the application must set this to sizeof(RFID_18K6C_SINGULATION_DYNAMICQ_PARMS).
When calling RFID_18K6CGetQueryParameters, the library will fill in this field
startQValue: The starting Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
minQValue: The minimum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
maxQValue: The maximum Q value to use. Valid values are 0 to 15.
minQValue <= startQValue <= maxQValue
retryCount: Specifies the number of times to try another execution of the singulation algorithm for the specified session/target before either toggling the target (if toggleTarget is non-zero) or terminating the inventory/tag access operation. Valid values are 0-255.
maxQueryRepCount: The maximum number of ISO 18000-6C QueryRep commands that will follow the ISO 18000-6C Query command during a single inventory round. Valid values are 0-255
toggleTarget: A flag that indicates if, after performing the inventory cycle for the specified target (i.e., A or B), if the target should be toggled (i.e.A to B or B to A) and another inventory cycle run.
A non-zero value indicates that the target should be toggled.
A zero value indicates that the target should not be toggled.
Note that if the target is toggled, retryCount and maxQueryReps will also apply to the new target.
thresholdMultiplier: The multiplier, specified in units of fourths (i.e., 0.25), that will be applied to the Q-adjustment threshold as part of the dynamic-Q algorithm.
Valid values are 0-255, inclusive.
Description:
This is for the dynamic Q with Q-threshold adjustment algorithm operation;

RFID_18K6C_SINGULATION_ALGORITHM_PARMS_T

Prototype:

```
public struct RFID_18K6C_SINGULATION_ALGORITHM_PARMS_T{
    public RFID_18K6C_SINGULATION_ALGORITHM            singulationAlgorithm;
    public RFID_18K6C_SINGULATION_FIXEDQ_PARMS         fixedQ;
    public RFID_18K6C_SINGULATION_DYNAMICQ_PARMS       dynamicQ;
```

public RFID_18K6C_SINGULATION_DYNAMICQ_ADJUST_PARMS dynamicQAdjust; /// Aug2007
public RFID_18K6C_SINGULATION_DYNAMICQ_THRESH_PARMS dynamicQThresh; ///
};

Fields:

singulationAlgorithm: enum of the singulation algorithm to use.

This value is a discriminator that determines the exact structure that pSingulationParms points to.

0 = fixedQ;

1 = dynamicQ

2 = dynamicQAdjust

3 = dynamicQThresh

fixedQ: this contains the FixedQ singulation algorithm parameters.

dynamicQ: this contains the DynamicQ singulation algorithm parameters.

dynamicQAdjust: this contains the DynamicQ singulation algorithm parameters with adjust.

dynamicQThresh: this contains the DynamicQ singulation algorithm parameters with threshold.

Description:

This is the ISO 18000-6C tag singulation algorithm parameters.

RFID_18K6C_QUERY_PARMS

Prototype:

```
public struct RFID_18K6C_QUERY_PARMS{
    public RFID_18K6C_TAG_GROUP                          tagGroup;
    public RFID_18K6C_SINGULATION_ALGORITHM_PARMS_T    singulationParms;
};
```

Fields:

tagGroup: tag group to be used.

singulationParms: parameters to be used.

Description:

This is for the query tag singulation algorithm parameters.

RFID_18K6C_INVENTORY_PARMS

Prototype:

```
public struct RFID_18K6C_INVENTORY_PARMS{
    public UInt32 length; //= sizeof(RFID_18K6C_INVENTORY_PARMS);
    public RFID_18K6C_COMMON_PARMS common;
};
```

Fields:

length: The length of the structure in bytes (=sizeof(RFID_18K6C_INVENTORY_PARMS)).

common: The ISO 18000-6C tag-protocol operation common parameters.
Description:
This is the parameters for ISO 18000-6C 18K6CTagInventory operation.

RFID_18K6C_READ_PARMS
Prototype:
```
public struct RFID_18K6C_READ_PARMS{
    public UInt32      length;
    public RFID_18K6C_COMMON_PARMS common;
    public RFID_18K6C_MEMORY_BANK    bank;
    public UInt16      offset;
    public UInt16      count;
    public UInt32      accessPassword;
};
```
Fields:
length: The length of the structure in bytes (=sizeof(RFID_18K6C_READ_PARMS)).
common: The ISO 18000-6C tag-protocol operation common parameters.
bank: The memmory bank to read from.
offset: The offset of the first 16-bit word to read.
count: The nubmer of 16-bit words to read.

If this value is zero and bank is RFID_18K6C_MEMORY_BANK_EPC, the read will return the contents of the tag EPC memory starting at the 16-bit word specified by offset through the end of the EPC.

If this value is zero and bank is not RFID_18K6C_MEMORY_BANK_EPC, the read will return, for the chosen memory bank, data starting from the 16-bit word specified by offset to the end of the memory bank.

accessPassword: The access password. A value of zero indicates no access password.
Description:
This is the parameters for ISO 18000-6C tag-read (18K6CTagRead) operation.

RFID_18K6C_WRITE_SEQUENTIAL_PARMS_T
Prototype:
```
public unsafe struct RFID_18K6C_WRITE_SEQUENTIAL_PARMS_T{
    public UInt32          length; //= sizeof(RFID_18K6C_WRITE_SEQUENTIAL_PARMS);
    public RFID_18K6C_MEMORY_BANK     bank;
    public UInt16                     count;    //1-8
    public UInt16                     offset;
    public fixed ushort               pData[8];    //fixed
```

};

Fields:

length: The length of this structure (=sizeof(RFID_18K6C_WRITE_SEQUENTIAL_PARMS)).

bank: The memory bank to write to.

count: The number of 16-bit words that will be written.
Valid values are 1 to 8.

offset: The offset, in the memory bank, of the first 16-bit word to write.

pData[]: Array of values to write to the tag's memory bank.: .

Description:

This is the parameters for specifying the data for a sequential tag write.

RFID_18K6C_WRITE_RANDOM_PARMS_T

Prototype:

```
public unsafe struct RFID_18K6C_WRITE_RANDOM_PARMS_T {
        public UInt32                               length; //=
sizeof(RFID_18K6C_WRITE_RANDOM_PARMS) (typo);
        public RFID_18K6C_MEMORY_BANK      bank;
        public UInt16                               count;      //1-8
        public UInt16                               reserved;//=0
        public fixed ushort                 pOffset[8]; //fixed
        public fixed ushort                 pData[8];
};
```

Fields:

length: The length of this structure (= sizeof(RFID_18K6C_WRITE_SEQUENTIAL_PARMS)).

bank: The memory bank to write to.

count: The number of 16-bit words that will be written. Valid values are 1 to 8.

reserved: Reserved. Set to zero.

pOffset: Pointer to an array of 16-bit offsets in the tag's memory bank where the corresponding array entry in pData will be written.

pData: Pointer to an array of 16-bit values that will be written to the tag memory offset specified in the corresponding array entry in pOffset.

Description:

This is the parameters for specifying the data for a random, single-memory-bank tag write.

RFID_18K6C_WRITE_PARMS_T

Prototype:

```
public struct RFID_18K6C_WRITE_PARMS_T{
        public UInt32                                           length; //= sizeof(RFID_18K6C_WRITE_PARMS);
```

```
    public RFID_18K6C_COMMON_PARMS    common;
    public RFID_18K6C_WRITE_TYPE          writeType;
    ///
    public UInt32                                 verify; //BOOL32 0 write-only; >0 w+r verify data
    public UInt32                                 verifyRetryCount; //0-7
    public UInt32                                 accessPassword;
    public RFID_18K6C_WRITE_SEQUENTIAL_PARMS_T sequential;
    public RFID_18K6C_WRITE_RANDOM_PARMS_T       random;
};
```

Fields:

length: The length of the structure in bytes (= sizeof(RFID_18K6C_WRITE_PARMS)).

common: The ISO 18000-6C tag-protocol operation common parameters.

writeType: The type of write.

verify: A flag to indicate if write should be verified by reading back the data written to the tag. A non-zero value indicates that a verify should be performed.

verifyRetryCount: If verify is non-zero, this is the number of retries in the event of a verification failure. Valid values are 0 to 7.

accessPassword: The access password. A value of zero indicates no access password.

sequential: random tag-write parameters.

random: random tag-write parameters.

Description:

This is the parameter for ISO 18000-6C tag-write operation.

RFID_18K6C_KILL_PARMS

Prototype:

```
public struct RFID_18K6C_KILL_PARMS{
    public UInt32                                 length; //= sizeof(RFID_18K6C_KILL_PARMS);
    public RFID_18K6C_COMMON_PARMS    common;
    public UInt32                                 accessPassword;
    public UInt32                                 killPassword;
};
```

Fields:

length: The length of the structure in bytes (=sizeof(RFID_18K6C_KILL_PARMS)).

common: The ISO 18000-6C tag-protocol operation common parameters.

accessPassword: The access password. A value of zero indicates no access password.

killPassword: The kill password. Must not be zero.

Description:

This is the parameter for ISO 18000-6C tag-kill operation.

RFID_18K6C_LOCK_PARMS

Prototype:

```
public struct RFID_18K6C_LOCK_PARMS{
    public UInt32                     length; //= sizeof(RFID_18K6C_LOCK_PARMS)
    public RFID_18K6C_COMMON_PARMS    common;
    public RFID_18K6C_TAG_PERM        permissions;
    public UInt32                     accessPassword;
};
```

Fields:

length: The length of the structure in bytes (=(RFID_18K6C_LOCK_PARMS)).

common: The ISO 18000-6C tag-protocol operation common parameters.

permissions: The access permissions for the tag.

accessPassword: The access password. A value of zero indicates no access password.

Description:

This is the parameter for ISO 18000-6C tag-lock operation.

RFID_18K6C_TAG_PERM

Prototype:

```
public struct RFID_18K6C_TAG_PERM{
    public RFID_18K6C_TAG_PWD_PERM killPasswordPermissions;
    public RFID_18K6C_TAG_PWD_PERM accessPasswordPermissions;
    public RFID_18K6C_TAG_MEM_PERM epcMemoryBankPermissions;
    public RFID_18K6C_TAG_MEM_PERM tidMemoryBankPermissions;
    public RFID_18K6C_TAG_MEM_PERM userMemoryBankPermissions;
};
```

Fields:

killPasswordPermissions: Permissions for the tag's kill password

accessPasswordPermissions: Permissions for the tag's access password

epcMemoryBankPermissions: Permissions for the tag's EPC memory bank

tidMemoryBankPermissions: Permissions for the tag's TID memory bank

userMemoryBankPermissions: Permissions for the tag's user memory bank.

Description:

This is the permission values for performing an ISO18000-6C tag lock operation.

~~~~~~

RfidMw_CmdCommon_T

Prototype:
~~~~~~

}. Fields::Description:
This is a structure used by RfidMw_Cmd_T.

RfidMw_Cmd_T
Prototype:
Fields:
: .
Description:
This is the Cmd structure to be used
in the further.

UINT96_T
Prototype:

```
public struct UINT96_T{
    public UInt32 m_MSB;
    public UInt32 m_CSB;
    public UInt32 m_LSB;
    };
```

Fields:
m_MSB: most significant QWord.
m_CSB: center significant QWord.
m_LSB: least significant QWord.
Description:
This is a structure used by PECRECORD_T.
e.g. if the 96 bits Gen2 EPC contains 0x30101234 56789012 12345678
then m_MSB = 0x30101234;
m_CSB = 0x56789012;
m_LSB = 0x12345678;

PECRECORD_T
Prototype:

```
public struct PECRECORD_T{
    public ushort m_seqnum;      //Byte 0- 1 local time-sequence
    public ushort m_Pc;          // 2- 3
    public UINT96_T m_Epc;       // 4-15
    public ushort m_Crc;         //16-17 optional ini 0x0000
    public ushort m_Cnt;         //20-21 ini from Db
    public ushort m_Flg;         //valid states 0(00):exist 2(10):exist,Cnt_chg 3(11):new,Cnt_chg
```

```
    public ushort m_Rssi;       //22-23 ini 0x0000
    public ushort m_AntCtrl;    //24-25 ini 0x0000
  }; ///26B
```

Fields:

m_seqnum: time-sequence number, local to RfidMw internal use only

m_Pc: PC

m_Epc: EPC

m_Crc: optionally filled CRC, initialize 0x0000

m_Cnt: Count

m_Flg: currently, the valid states are:

0(00):exist

2(10):exist,Cnt_chg

3(11):new,Cnt_chg

m_Rssi: unnormalized RSSI value

m_AntCtrl: unnormalized Antenna Control values, initialize as 0x0000

Description:

This PEC (PC-EPC-CRC & inventory data) record is used in the middleware data list.
C# App communicates (find, read, write) with RfidMw using this PEC record format.

Function Definitions:

f_RfidSpDll_Initialize

Prototype:

public static extern HRESULT_RFID f_RfidSpDll_Initialize(
IntPtr hWnd);

Parameters: (Inputs/ Outputs/ ReturnValue)

[in] IntPtr hWnd: handle of the Message Window of the C# Application.

Message:

None.

Description:

This initializes the RfidSp.

e.g. Create C++ threads.

f_RfidSpDll_Uninitialize

Prototype:

public static extern HRESULT_RFID f_RfidSpDll_Uninitialize();

Parameters:

None.

Message:
None.
Description:
This uninitializes the RfidSp.
e.g. Un-initialize & Delete C++ threads.

f_RfidMw_Initialize
Prototype:
public static extern HRESULT_RFID f_RfidMw_Initialize();
Message:
None.
Description:
No operation. Reserved for further use.

f_RfidMw_Uninitialize
Prototype:
public static extern HRESULT_RFID f_RfidMw_Uninitialize();
Message:
None.
Description:
No operation. Reserved for further use.

f_RfidMw_PostCmd
Prototype:
public static extern HRESULT_RFID f_RfidMw_PostCmd(
    ref RfidMw_Cmd_T sRfidMw_cmd); .
Message:
None.
Description:
No operation. Reserved for further use.

f_RfidMw_TagInv_SetAllTaglist
Prototype:
public static extern HRESULT_RFID f_RfidMw_TagInv_SetAllTaglist(
    UInt32 siz,
ref IntPtr parylist);
Parameters:
[in] UInt32 siz:

[in] ref IntPtr parylist:

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_SetAllTaglist notification.

Description:

This set multiple tags to the RfidMw in the format defined by RfidMw.

f_RfidMw_TagInv_AddATag

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_AddATag(
    ref PECRECORD_T st_PecRec);

Parameters:

[in] ref PECRECORD_T st_PecRec:

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_AddATag notification.

Description:

This adds a single tag to the RfidMw in the format defined by RfidMw.

f_RfidMw_TagInv_FindATag

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_FindATag(
    ref PECRECORD_T st_PecRec);.

Parameters:

[in] ref PECRECORD_T st_PecRec: .

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_FindATag notification.

Description:

This finds a tag in the RfidMw cached data list.

f_RfidMw_TagInv_ClearAllTaglist

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_ClearAllTaglist();

Parameters:

None.

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_ClearAllTaglist notification.

Description:

This empties the RfidMw cached data list.

f_RfidMw_TagInv_UpdateAllTaglistToFile

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_UpdateAllTaglistToFile();

Parameters:

None.

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_UpdateAllTaglistToFile notification.

Description:

This writes all data in RfidMw to a text file.

f_RfidMw_TagInv_GetUpdateTaglist

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_GetUpdateTaglist( );

Parameters:

None.

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_GetUpdateTaglist notification.

Description:

This gets only rows of updated tags from RfidMw.

f_RfidMw_TagInv_GetAllTaglist

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_GetAllTaglist( ).

Parameters:

None.

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.

Description:

This gets all rows of tags from RfidMw.

f_RfidMw_TagInv_SetMsgMode

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_SetMsgMode(int mode);

Parameters:

[in] int mode: .

if mode == 0 all tags' notifications are sent;

if mode == 1, only the "1st read" of each tag is sent.

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.
Description:
This is **optional**…
This sets the RfidMw into an message output mode, where only the 1st read tag data is sent to the C# App. This reduces the number of messages & the amount of data to be processed by C#, in both normal or compact response mode.

f_RfidMw_TagInv_PecRssiMin_Set
Prototype:
public static extern HRESULT_RFID f_RfidMw_TagInv_PecRssiMin_Set (ref UInt16 val);
Parameters:
[in] ref UInt16 val:
Message:
RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.
Description:
This sets the Rssi Threshold of the RfidMw module;
Tag data is not added to the cached data-list if its Rssi element is below threshold.

f_RfidMw_TagInv_PecRssiMin_Get
Prototype:
public static extern HRESULT_RFID f_RfidMw_TagInv_PecRssiMin_Get (ref UInt16 val);
Parameters:
[out] ref UInt16 val:
Message:
RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.
Description:
This gets the Rssi Threshold of the RfidMw module.

f_RfidMw_TagInv_PecAntCtrlMin_Set
Prototype:
public static extern HRESULT_RFID f_RfidMw_TagInv_PecAntCtrlMin_Set(ref UInt16 val);
Parameters:
[in] ref UInt16 val: .
Message:
RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.
Description:
This sets the AntCtrl Threshold of the RfidMw module;
Tag data is not added to the cached data-list if its AntCtrl element is below threshold.

f_RfidMw_TagInv_PecAntCtrlMin_Get

Prototype:

public static extern HRESULT_RFID f_RfidMw_TagInv_PecAntCtrlMin_Get(ref UInt16 val);.

Parameters:

[out] ref UInt16 val: .

Message:

RFIDMW_REQEND_TYPE_MSGID_TagInv_ notification.

Description:

This gets the AntCtrl Threshold of the RfidMw module.

f_RfidDev_Initialize

Prototype:

public static extern HRESULT_RFID f_RfidDev_Initialize();

Description:

No operation. Reserved for further use.

f_RfidDev_Uninitialize

Prototype:

public static extern HRESULT_RFID f_RfidDev_Uninitialize();

Description:

No operation. Reserved for further use.

f_RfidDev_Startup

Prototype:

public static extern HRESULT_RFID f_RfidDev_Startup(
    ref RFID_Startup_T st_RfidSpReq_Startup);

Parameters:

[in] ref RFID_Startup_T st_RfidSpReq_Startup: .

Message:

RFID_REQEND_TYPE_MSGID_Startup notification.

Description:

This initializes the Rfid Library.

f_RfidDev_Shutdown

Prototype:

public static extern HRESULT_RFID f_RfidDev_Shutdown(
    ref RFID_Shutdown_T st_RfidSpReq_Shutdown);

Parameters:
[in] ref RFID_Shutdown_T st_RfidSpReq_Shutdown: .
Message:
RFID_REQEND_TYPE_MSGID_Shutdown notification.
Description:
This shuts down the Rfid Library.
This also cleans up all resources including closing all open radio handles and returning radios to idle.

f_RfidDev_RetrieveAttachedRadiosList
Prototype:
public static extern HRESULT_RFID f_RfidDev_RetrieveAttachedRadiosList(
    ref RFID_RetrieveAttachedRadiosList_T st_RfidSpReq_RetrieveAttachedRadiosList);
Parameters:
[out] ref RFID_RetrieveAttachedRadiosList_T st_RfidSpReq_RetrieveAttachedRadiosList: .
Message:
RFID_REQEND_TYPE_MSGID_RetrieveAttachedRadiosList notification.
Description:
Retrieves the list of radio modules attached to the system.
If succeeded, application can open the Radio Object then.

f_RfidDev_RadioOpen
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioOpen(
    ref RFID_RadioOpen_T st_RfidSpReq_RadioOpen);
Parameters:
[out] ref RFID_RadioOpen_T st_RfidSpReq_RadioOpen: .
Message:
RFID_REQEND_TYPE_MSGID_RadioOpen notification.
Description:
This requests explicit control of a radio. The following function calls will use the return rfid_handle the access the radio object.

f_RfidDev_RadioClose
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioClose(
    ref RFID_RadioClose_T st_RfidSpReq_RadioClose);
Parameters:
[in] ref RFID_RadioClose_T st_RfidSpReq_RadioClose.

Message:
RFID_REQEND_TYPE_MSGID_RadioClose notification.
Description:
This releases the rfid_handle, thus also releasing the radio object of the opened radio.
On close, any executing or outstanding requests are cancelled and the radio is returned to idle.

f_RfidDev_RadioSetConfigurationParameter
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioSetConfigurationParameter(
ref RFID_RadioGetSetConfigurationParameter_T st_RfidSpReq_RadioSetConfigurationParameter);.
Parameters:
[in] ref RFID_RadioGetSetConfigurationParameter_T
st_RfidSpReq_RadioSetConfigurationParameter: .
Message:
RFID_REQEND_TYPE_MSGID_RadioSetConfigurationParameter notification.
Description:
This sets the low-level configuration parameter of the radio.

f_RfidDev_RadioGetConfigurationParameter
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioGetConfigurationParameter(
ref RFID_RadioGetSetConfigurationParameter_T st_RfidSpReq_RadioGetConfigurationParameter);.
Parameters:
[out] ref RFID_RadioGetSetConfigurationParameter_T
st_RfidSpReq_RadioGetConfigurationParameter: .
Message:
RFID_REQEND_TYPE_MSGID_RadioGetConfigurationParameter notification.
Description:
This gets the low-level configuration parameter of the radio.

f_RfidDev_RadioSetOperationMode
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioSetOperationMode(
ref RFID_RadioGetSetOperationMode_T st_RfidSpReq_RadioSetOperationMode);
Parameters:
[in] ref RFID_RadioGetSetOperationMode_T st_RfidSpReq_RadioSetOperationMode: .
Message:
RFID_REQEND_TYPE_MSGID_RadioSetOperationMode notification.

Description:
.

f_RfidDev_RadioGetOperationMode
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioGetOperationMode(
ref RFID_RadioGetSetOperationMode_T st_RfidSpReq_RadioGetOperationMode);.
Parameters:
[out] ref RFID_RadioGetSetOperationMode_T st_RfidSpReq_RadioGetOperationMode: .
Message:
RFID_REQEND_TYPE_MSGID_RadioGetOperationMode notification.
Description:
This sets the radio's operation mode.
The operation mode will remain in effect until it is explicitly changed via RFID_RadioSetOperationMode.

f_RfidDev_RadioSetPowerState
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioSetPowerState(
ref RFID_RadioGetSetPowerState_T st_RfidSpReq_RadioSetPowerState);
Parameters:
[in] : .
Message:
RFID_REQEND_TYPE_MSGID_ RadioSetPowerState notification.
Description:
.

f_RfidDev_RadioGetPowerState
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioGetPowerState(
ref RFID_RadioGetSetPowerState_T st_RfidSpReq_RadioGetPowerState);.
Parameters:
[out] : .
Message:
RFID_REQEND_TYPE_MSGID_RadioGetPowerState notification.
Description:
This retrieves the radio's power state (not the antenna RF power.).

f_RfidDev_RadioSetCurrentLinkProfile

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioSetCurrentLinkProfile(
ref RFID_RadioGetSetCurrentLinkProfile_T st_RfidSpReq_RadioSetCurrentLinkProfile);

Parameters:

[out] ref RFID_RadioGetSetCurrentLinkProfile_T st_RfidSpReq_RadioSetCurrentLinkProfile:.

Message:

RFID_REQEND_TYPE_MSGID_RadioSetCurrentLinkProfile notification.

Description:

This sets the current link profile for the radio module.

f_RfidDev_RadioGetCurrentLinkProfile

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioGetCurrentLinkProfile(
ref RFID_RadioGetSetCurrentLinkProfile_T st_RfidSpReq_RadioGetCurrentLinkProfile);

Parameters:

[out] ref RFID_RadioGetSetCurrentLinkProfile_T st_RfidSpReq_RadioGetCurrentLinkProfile:.

Message:

RFID_REQEND_TYPE_MSGID_RadioGetCurrentLinkProfile notification.

Description:

This gets the current link profile for the radio module.

f_RfidDev_RadioGetLinkProfile

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioGetLinkProfile(
ref RFID_RadioGetLinkProfile_T st_RfidSpReq_RadioGetLinkProfile);

Parameters:

[out] ref RFID_RadioGetLinkProfile_T st_RfidSpReq_RadioGetLinkProfile:

Message:

RFID_REQEND_TYPE_MSGID_RadioGetLinkProfile notification.

Description:

This retrieves the information for the specified link profile for the radio.

f_RfidDev_RadioWriteLinkProfileRegister

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioWriteLinkProfileRegister(
ref RFID_RadioReadWriteLinkProfileRegister_T
st_RfidSpReq_RadioWriteLinkProfileRegister);

Parameters:
[out] ref RFID_RadioReadWriteLinkProfileRegister_T
st_RfidSpReq_RadioWriteLinkProfileRegister:
Message:
RFID_REQEND_TYPE_MSGID_RadioWriteLinkProfileRegister notification.
Description:
This writes a value to a link-profile register for the specified link profile.

f_RfidDev_RFID_RadioReadLinkProfileRegister
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioReadLinkProfileRegister(
ref RFID_RadioReadWriteLinkProfileRegister_T
st_RfidSpReq_RadioReadLinkProfileRegister);
Parameters:
[out] ref RFID_RadioReadWriteLinkProfileRegister_T st_RfidSpReq_RadioReadLinkProfileRegister:
Message:
RFID_REQEND_TYPE_MSGID_RadioReadLinkProfileRegister notification.
Description:
This retrieves the contents of a link-profile register for the specified link profile.

f_RfidDev_AntennaPortGetStatus
Prototype:
public static extern HRESULT_RFID f_RfidDev_AntennaPortGetStatus(
ref RFID_AntennaPortGetStatus_T st_RfidSpReq_AntennaPortGetStatus);
Parameters:
[out] ref RFID_AntennaPortGetStatus_T st_RfidSpReq_AntennaPortGetStatus:
Message:
RFID_REQEND_TYPE_MSGID_AntennaPortGetStatus notification.
Description:
This retrieves the status of a radio module's antenna port.

f_RfidDev_AntennaPortSetState
Prototype:
public static extern HRESULT_RFID f_RfidDev_AntennaPortSetState(
ref RFID_AntennaPortSetState_T st_RfidSpReq_AntennaPortSetState);
Parameters:
[in] ref RFID_AntennaPortSetState_T st_RfidSpReq_AntennaPortSetState:.
Message:

RFID_REQEND_TYPE_MSGID_AntennaPortSetState notification.

Description:

This sets the state (ON/OFF) of a radio’s antenna port.

f_RfidDev_AntennaPortSetConfiguration

Prototype:

public static extern HRESULT_RFID f_RfidDev_AntennaPortSetConfiguration(
    ref RFID_AntennaPortGetSetConfiguration_T st_RfidSpReq_AntennaPortSetConfiguration);

Parameters:

[in] ref RFID_AntennaPortGetSetConfiguration_T st_RfidSpReq_AntennaPortSetConfiguration:.

Message:

RFID_REQEND_TYPE_MSGID_AntennaPortSetConfiguration notification.

Description:

This sets the configuration for a radio’s antenna port.

f_RfidDev_AntennaPortGetConfiguration

Prototype:

public static extern HRESULT_RFID f_RfidDev_AntennaPortGetConfiguration(
    ref RFID_AntennaPortGetSetConfiguration_T st_RfidSpReq_AntennaPortGetConfiguration);

Parameters:

[out] ref RFID_AntennaPortGetSetConfiguration_T st_RfidSpReq_AntennaPortGetConfiguration:.

Message:

RFID_REQEND_TYPE_MSGID_AntennaPortGetConfiguration notification.

Description:

This retrieves the configuration for a radio's antenna port.

f_RfidDev_18K6CSetSelectCriteria01

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CSetSelectCriteria01(
    ref RFID_18K6CSetSelectCriteria__T st_RfidSpReq_18K6CSetSelectCriteria,
    ref RFID_18K6C_SELECT_CRITERION criteria01);.

Parameters:

[in] ref RFID_18K6CSetSelectCriteria__T st_RfidSpReq_18K6CSetSelectCriteria: .

[in] ref RFID_18K6C_SELECT_CRITERION criteria01: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CSetSelectCriteria notification.

Description:

This 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 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.

f_RfidDev_18K6CGetSelectCriteria01
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CGetSelectCriteria01(
ref RFID_18K6CGetSelectCriteria__T st_RfidSpReq_18K6CGetSelectCriteria,
ref RFID_18K6C_SELECT_CRITERION criteria01);
Parameters:
[out] ref RFID_18K6CGetSelectCriteria__T st_RfidSpReq_18K6CGetSelectCriteria: .
[out] ref RFID_18K6C_SELECT_CRITERION criteria01: .
Message:
RFID_REQEND_TYPE_MSGID _18K6CGetSelectCriteria notification.
Description:
This retrieves the configured tag-selection criteria for the ISO 18000-6C select command.

f_RfidDev_18K6CSetPostMatchCriteria01
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CSetPostMatchCriteria01(
ref RFID_18K6CSetPostMatchCriteria__T st_RfidSpReq_18K6CSetPostMatchCriteria,
ref RFID_18K6C_SINGULATION_CRITERION criteria01);
Parameters:
[in] ref RFID_18K6CSetPostMatchCriteria__T st_RfidSpReq_18K6CSetPostMatchCriteria: .
[in] ref RFID_18K6C_SINGULATION_CRITERION criteria01: .
Message:
RFID_REQEND_TYPE_MSGID_18K6CSetPostMatchCriteria notification.
Description:
This configures the post-singulation match criteria to be used by the radio.
The supplied post-singulation match criteria will be used for any tag-protocol operations 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.

f_RfidDev_18K6CGetPostMatchCriteria01
Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CGetPostMatchCriteria01(
ref RFID_18K6CGetPostMatchCriteria__T st_RfidSpReq_18K6CGetPostMatchCriteria,
ref RFID_18K6C_SINGULATION_CRITERION criteria01);.

Parameters:

[out] ref RFID_18K6CGetPostMatchCriteria__T st_RfidSpReq_18K6CGetPostMatchCriteria: .

[out] ref RFID_18K6C_SINGULATION_CRITERION criteria01: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CGetPostMatchCriteria notification.

Description:

This Retrieves the configured post-singulation match criteria.

f_RfidDev_18K6CSetQueryTagGroup

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CSetQueryTagGroup (
ref RFID_18K6CGetSetQueryTagGroup_T st_RfidSpReq_18K6CSetQueryTagGroup);

Parameters:

[in] ref RFID_18K6CGetSetQueryTagGroup_T st_RfidSpReq_18K6CSetQueryTagGroup: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CSetQueryTagGroup notification.

Description:

This specifies which tag group will have subsequent tag-protocol operations applied to it.

f_RfidDev_18K6CGetQueryTagGroup

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CGetQueryTagGroup (
ref RFID_18K6CGetSetQueryTagGroup_T st_RfidSpReq_18K6CGetQueryTagGroup);

Parameters:

[out] ref RFID_18K6CGetSetQueryTagGroup _T st_RfidSpReq_18K6CGetQueryTagGroup: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CGetQueryTagGroup notification.

Description:

This retrieves the tag group that will have subsequent tag-protocol operations applied to it.

f_RfidDev_18K6CSetCurrentSingulationAlgorithm

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CSetCurrentSingulationAlgorithm (
ref RFID_18K6CGetSetCurrentSingulationAlgorithm_T

st_RfidSpReq_18K6CSetCurrentSingulationAlgorithm);
Parameters:
[in] ref RFID_18K6CGetSetCurrentSingulationAlgorithm_T
st_RfidSpReq_18K6CSetCurrentSingulationAlgorithm: .
Message:
RFID_REQEND_TYPE_MSGID_18K6CSetCurrentSingulationAlgorithm notification.
Description:
This sets the currently-active singulation algorithm (= the algorithm that is used when performing a tag-protocol operation).

f_RfidDev_18K6CGetCurrentSingulationAlgorithm
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CGetCurrentSingulationAlgorithm (
ref RFID_18K6CGetSetCurrentSingulationAlgorithm_T
st_RfidSpReq_18K6CGetCurrentSingulationAlgorithm);
Parameters:
[out] ref RFID_18K6CGetSetCurrentSingulationAlgorithm_T
st_RfidSpReq_18K6CGetCurrentSingulationAlgorithm: .
Message:
RFID_REQEND_TYPE_MSGID_18K6CGetCurrentSingulationAlgorithm notification.
Description:
This retrieves the currently-active singulation algorithm.

f_RfidDev_18K6CSetSingulationAlgorithmParameters
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CSetSingulationAlgorithmParameters(
ref RFID_18K6CGetSetSingulationAlgorithmParameters_T
st_RfidSpReq_18K6CSetSingulationAlgorithmParameters);
Parameters:
[in] ref RFID_18K6CGetSetSingulationAlgorithmParameters_T
st_RfidSpReq_18K6CSetSingulationAlgorithmParameters: .
Message:
RFID_REQEND_TYPE_MSGID_18K6CSetSingulationAlgorithmParameters notification.
Description:
This configures the settings for a particular singulation algorithm.
Please notice that: configuring a singulation algorithm does not automatically set it as the current singulation algorithm.

f_RfidDev_18K6CGetSingulationAlgorithmParameters

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CGetSingulationAlgorithmParameters (
    ref RFID_18K6CGetSetSingulationAlgorithmParameters_T
st_RfidSpReq_18K6CGetSingulationAlgorithmParameters);

Parameters:

[out] ref RFID_18K6CGetSetSingulationAlgorithmParameters_T
st_RfidSpReq_18K6CGetSingulationAlgorithmParameters: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CGetSingulationAlgorithmParameters notification.

Description:

This retrieves the settings for a particular singulation algorithm.

f_RfidDev_18K6CSetQueryParameters

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CSetQueryParameters(
    ref RFID_18K6CSetQueryParameters_T st_RfidSpReq_18K6CSetQueryParameters);

Parameters:

[in] ref RFID_18K6CSetQueryParameters_T st_RfidSpReq_18K6CSetQueryParameters: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CSetQueryParameters notification.

Description:

This configures the parameters for the ISO 18000-6C query command.

Failure to call this prior to executing the first tag-protocol operation will result in the RFID radio module using default values.

Currently, this has been deprecated and replaced by the combination of
RFID_18K6CGetQueryTagGroup, RFID_18K6CSetCurrentSingulationAlgorithm, and
RFID_18K6CSetSingulationAlgorithmParameters.

This remains for backwards compatibility only; code should not use it as this function.

f_RfidDev_18K6CGetQueryParameters

Prototype:

public static extern HRESULT_RFID f_RfidDev_18K6CGetQueryParameters (
    ref RFID_18K6CGetQueryParameters_T st_RfidSpReq_18K6CGetQueryParameters);

Parameters:

[out] ref RFID_18K6CGetQueryParameters_T st_RfidSpReq_18K6CGetQueryParameters: .

Message:

RFID_REQEND_TYPE_MSGID_18K6CGetQueryParameters notification.

Description:
This retrieves the query parameters for the ISO 18000-6C query command.
The query parameters may not be retrieved while a radio module is executing a tag-protocol operation.
Currently, this has been deprecated and replaced by the combination of RFID_18K6CGetQueryTagGroup, RFID_18K6CGetCurrentSingulationAlgorithm, and RFID_18K6CGetSingulationAlgorithmParameters.
This remains for backwards compatibility only; code should not use it as this function.

f_RfidDev_18K6CTagInventory
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CTagInventory(
    ref RFID_18K6CTagInventory_T st_RfidSpReq_18K6CTagInventory);
Parameters:
[in] ref RFID_18K6CTagInventory_T st_RfidSpReq_18K6CTagInventory: .
Message:
RFIDMW_REQEND_TYPE_MSGID_18K6CTagInventory notification.
Description:
This 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.
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..

f_RfidDev_18K6CTagRead
Prototype:
public static extern HRESULT_RFID f_RfidDev_18K6CTagRead(
    ref RFID_18K6CTagRead_T st_RfidSpReq_18K6CTagRead);
Parameters:
[in] ref RFID_18K6CTagRead_T st_RfidSpReq_18K6CTagRead: .
Message:
RFIDMW_REQEND_TYPE_MSGID_18K6CTagRead notification.
Description:
This 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.

f_RfidDev_18K6CTagWrite

Prototype:

```
public static extern HRESULT_RFID f_RfidDev_18K6CTagWrite(
        ref RFID_18K6CTagWrite_T st_RfidSpReq_18K6CTagWrite);
```

Parameters:

[in] ref RFID_18K6CTagWrite_T st_RfidSpReq_18K6CTagWrite: .

Message:

RFIDMW_REQEND_TYPE_MSGID_18K6CTagWrite notification.

Description:

This executes a tag write for the tags of interest.

f_RfidDev_18K6CTagKill

Prototype:

```
public static extern HRESULT_RFID f_RfidDev_18K6CTagKill(
        ref RFID_18K6CTagKill_T st_RfidSpReq_18K6CTagKill);
```

Parameters:

[in] ref RFID_18K6CTagKill_T st_RfidSpReq_18K6CTagKill: .

Message:

RFIDMW_REQEND_TYPE_MSGID_18K6CTagKill notification.

Description:

This executes a tag kill for the tags of interest.

f_RfidDev_18K6CTagLock

Prototype:

```
public static extern HRESULT_RFID f_RfidDev_18K6CTagLock(
        ref    RFID_18K6CTagLock_T st_RfidSpReq_18K6CTagLock);
```

Parameters:
[in] ref RFID_18K6CTagLock_T st_RfidSpReq_18K6CTagLock: .
Message:
RFIDMW_REQEND_TYPE_MSGID_18K6CTagLock notification.
Description:
This executes a tag lock for the tags of interest.

f_RfidDev_RadioSetResponseDataMode
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioSetResponseDataMode(
ref RFID_RadioGetSetResponseDataMode_T st_RfidSpReq_RadioSetResponseDataMode);
Parameters:
[in] ref RFID_RadioGetSetResponseDataMode_T st_RfidSpReq_RadioSetResponseDataMode: .
Message:
RFIDMW_REQEND_TYPE_MSGID_RadioSetResponseDataMode notification.
Description:
This sets the operation response data reporting mode for tag-protocol operations. 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..

f_RfidDev_RadioGetResponseDataMode
Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioGetResponseDataMode(
ref RFID_RadioGetSetResponseDataMode_T st_RfidSpReq_RadioGetResponseDataMode);
Parameters:
[out] ref RFID_RadioGetSetResponseDataMode_T st_RfidSpReq_RadioGetResponseDataMode:..
Message:
RFIDMW_REQEND_TYPE_MSGID_RadioGetResponseDataMode notification.
Description:
This retrieves the operation response data reporting mode.

f_RfidDev_MacGetVersion
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacGetVersion(
ref RFID_MacGetVersion_T st_RfidSpReq_MacGetVersion);;
Parameters:

[in] ref RFID_MacGetVersion_T st_RfidSpReq_MacGetVersion: .
Message:
RFIDMW_REQEND_TYPE_MSGID_MacGetVersion notification.
Description:
This gets the version of the MAC.

f_RfidDev_MacReadOemData
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacReadOemData(
    ref RFID_MacReadWriteOemData_T st_RfidSpReq_MacReadOemData);
Parameters:
[in] ref RFID_MacReadWriteOemData_T st_RfidSpReq_MacReadOemData: .
Message:
RFIDMW_REQEND_TYPE_MSGID_MacReadOemData notification.
Description:
This reads one or more 32-bit words from the MAC's OEM configuration data.

f_RfidDev_MacWriteOemData
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacWriteOemData(
    ref RFID_MacReadWriteOemData_T st_RfidSpReq_MacWriteOemData);
Parameters:
[out] ref RFID_MacReadWriteOemData_T st_RfidSpReq_MacWriteOemData: .
Message:
RFIDMW_REQEND_TYPE_MSGID_MacWriteOemData notification.
Description:
This writes one or more 32-bit words to the MAC's OEM configuration data.

f_RfidDev_MacReset
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacReset(
    ref RFID_MacReset_T st_RfidSpReq_MacReset);
Parameters:
[in] ref RFID_MacReset_T st_RfidSpReq_MacReset.
Message:
RFIDMW_REQEND_TYPE_MSGID_MacReset notification.
Description:
This 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 will return immediately with an RFID_ERROR_OPERATION_CANCELLED error.
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..

f_RfidDev_MacClearError
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacClearError(
    ref RFID_MacClearError_T st_RfidSpReq_MacClearError);
Parameters:
[in] ref RFID_MacClearError_T st_RfidSpReq_MacClearError: .
Message:
RFID_REQEND_TYPE_MSGID_MacClearError notification.
Description:
This attempts to clear the error state for the radio module’s MAC firmware.

f_RfidDev_MacBypassWriteRegister
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacBypassWriteRegister(
    ref RFID_MacBypassReadWriteRegister_T st_RfidSpReq_MacBypassWriteRegister);
Parameters:
[in] ref RFID_MacBypassReadWriteRegister_T st_RfidSpReq_MacBypassWriteRegister.
Message:
RFID_REQEND_TYPE_MSGID_MacBypassWriteRegister notification.
Description:
This allows for direct writing of registers on the radio (i.e. bypassing the MAC & take effect at the RF Front end immediately).

f_RfidDev_MacBypassReadRegister
Prototype:
public static extern HRESULT_RFID f_RfidDev_MacBypassReadRegister(
    ref RFID_MacBypassReadWriteRegister_T st_RfidSpReq_MacBypassReadRegister);
Parameters:
[out] ref RFID_MacBypassReadWriteRegister_T st_RfidSpReq_MacBypassReadRegister.
Message:
RFID_REQEND_TYPE_MSGID_MacBypassReadRegister notification.

Description:
This allows for direct reading of registers.

f_RfidDev_MacSetRegion

Prototype:
public static extern HRESULT_RFID f_RfidDev_MacSetRegion(
    ref RFID_MacGetSetRegion_T st_RfidSpReq_MacSetRegion);
Parameters:
[in] ref RFID_MacGetSetRegion_T st_RfidSpReq_MacSetRegion: .
Message:
RFID_REQEND_TYPE_MSGID_MacSetRegion notification.
Description:
This sets the regulatory mode region for the MAC's operation.

f_RfidDev_MacGetRegion

Prototype:
public static extern HRESULT_RFID f_RfidDev_MacGetRegion(
    ref RFID_MacGetSetRegion_T st_RfidSpReq_MacGetRegion);
Parameters:
[out] ref RFID_MacGetSetRegion_T st_RfidSpReq_MacGetRegion: .
Message:
RFID_REQEND_TYPE_MSGID_MacGetRegion notification.
Description:
This gets the regulatory mode region for the MAC's operation.

f_RfidDev_RadioSetGpioPinsConfiguration

Prototype:
public static extern HRESULT_RFID f_RfidDev_RadioSetGpioPinsConfiguration(
    ref RFID_RadioSetGpioPinsConfiguration_T st_RfidSpReq_RadioSetGpioPinsConfiguration);
Parameters:
[in] ref RFID_RadioSetGpioPinsConfiguration_T st_RfidSpReq_RadioSetGpioPinsConfiguration: .
Message:
RFID_REQEND_TYPE_MSGID_RadioSetGpioPinsConfiguration notification.
Description:
This configures the specified radio module's GPIO pins. GPIO pins 0-3 are valid..

f_RfidDev_RadioGetGpioPinsConfiguration

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioGetGpioPinsConfiguration(
    ref RFID_RadioGetGpioPinsConfiguration_T st_RfidSpReq_RadioGetGpioPinsConfiguration);

Parameters:

[out] ref RFID_RadioGetGpioPinsConfiguration_T st_RfidSpReq_RadioGetGpioPinsConfiguration.

Message:

RFID_REQEND_TYPE_MSGID_RadioGetGpioPinsConfiguration notification.

Description:

This retrieves the configuration for the radio module's GPIO pins.

f_RfidDev_RadioReadGpioPins

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioReadGpioPins(
    ref RFID_RadioReadWriteGpioPins_T st_RfidSpReq_RadioReadGpioPins);

Parameters:

[out] ref RFID_RadioReadWriteGpioPins_T st_RfidSpReq_RadioReadGpioPins.

Message:

RFID_REQEND_TYPE_MSGID_RadioReadGpioPins notification.

Description:

This reads the specified radio module's GPIO pins. Attempting to read from an output GPIO pin results in an error.

f_RfidDev_RadioWriteGpioPins

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioWriteGpioPins(
    ref RFID_RadioReadWriteGpioPins_T st_RfidSpReq_RadioWriteGpioPins);

Parameters:

[in] ref RFID_RadioReadWriteGpioPins_T st_RfidSpReq_RadioWriteGpioPins.

Message:

RFID_REQEND_TYPE_MSGID_RadioWriteGpioPins notification.

Description:

This writes the specified radio module's GPIO pins. Attempting to write to an input GPIO pin results in an error.

f_RfidDev_RadioIssueCommand

Prototype:

public static extern HRESULT_RFID f_RfidDev_RadioIssueCommand (
    ref RFID_RadioIssueCommand_T st_RfidSpReq_RadioIssueCommand);

Parameters:

[in] ref RFID_RadioIssueCommand_T st_RfidSpReq_RadioIssueCommand.
Message:
RFID_REQEND_TYPE_MSGID_ notification.
Description:
This issues a radio command. Application needs to process the resulting message packet and verify if the command was success.

f_RfidDev_RadioCancelOperation
Prototype:

```
public static extern HRESULT_RFID      f_RfidDev_RadioCancelOperation(
            ref RFID_RadioCancelOperation_T st_RfidSpReq_RadioCancelOperation );
```

Parameters:
[in] ref RFID_RadioCancelOperation_T st_RfidSpReq_RadioCancelOperation: .
Message:
RFID_REQEND_TYPE_MSGID_RadioCancelOperation notification.
Description:
This cancels the current RfidSp tag.operation. RfidSp will return after all the pending message for the current operations are sent to the application.

f_RfidDev_RadioAbortOperation
Prototype:

```
public static extern HRESULT_RFID      f_RfidDev_RadioAbortOperation(
            ref RFID_RadioAbortOperation_T    st_RfidSpReq_RadioAbortOperation );
```

Parameters:
[in] ref RFID_RadioAbortOperation_T   st_RfidSpReq_RadioAbortOperation: .
Message:
RFID_REQEND_TYPE_MSGID_RadioAbortOperation notification.
Description:
This aborts the current RfidSp tag.operation. RfidSp will return quickly.

PosSp:

Overview:

PosSp is a C# class in the PosSp_Apis class library.

It provides a managed interface to the C# application to access the pda specific peripherals synchronously.

C# namespace:

PosSp_Apis .

Dependencies:

Program Files\W_PosSp.dll;

Type Definitions:

Not needed.

Enumerations:

PosSp_Apis.BUZZER_SOUND

Prototype:

```
public enum BUZZER_SOUND : uint {
    BUZZER_LOW_SOUND,
    BUZZER_MIDDLE_SOUND,
    BUZZER_HIGH_SOUND
};
```

Description:

This is the volume of the buzzer on pda.

Macros:

Not needed.

Function Definitions:

f_PosSp_Initialize

Prototype:

public static extern int f_PosSp_Initialize();

Parameters:

None.
Description:
Initialization of the POS SP library at the caller thread.

f_PosSp_Uninitialize
Prototype:
public static extern int f_PosSp_Uninitialize();
Parameters:
None.
Description:
Un-initialized the POS SP & free all resources..

f_PosSp_GetDeviceName
Prototype:
public static extern void f_PosSp_GetDeviceName(string DeviceName); .
Parameters:
[out] DeviceName: the device name in WinCE registry. This name can be changed in the "control_panel/system/"
Description:
This gets the DeviceNane of the WinCE from the Registry.

f_PosSp_LedSetOn
Prototype:
public static extern bool f_PosSp_LedSetOn(uint Color);
Parameters:
Color: 32bits COLORREF in the format of 0x00bbggrr (bb=blue, gg=green, rr=red color byte).
Description:
This turns on the Led.

f_PosSp_LedBlink
Prototype:
public static extern bool f_PosSp_LedBlink(uint colorRGB, short Period, short OnTime);
Parameters:
colorRGB: 32bits COLORREF in the format of 0x00bbggrr.
Period: repetition interval [ms].
OnTime: duration of lights-on time in each interval [ms].
Description:
This blinks the Led at maximum brightness.

f_PosSp_LedSetOff

Prototype:

public static extern void f_PosSp_LedSetOff();

Parameters:

None.

Description:

This turns off the Led.

f_PosSp_ToneOn

Prototype:

public static extern void f_PosSp_ToneOn(short freq, short Duration, uint SoundLevel);

Parameters:

freq:	frequency of the tone [Hz]..
Duration:	duration [ms]
SoundLevel:	one of the value in BUZZER_SOUND enumeration.

Description:

This plays a tone at the given frequency, for the given duration. at the buzzer.

f_PosSp_ToneOff

Prototype:

public static extern void f_PosSp_ToneOff();.

Parameters:

None.

Description:

This stops a playing tone.

f_PosSp_MelodyPlay

Prototype:

public static extern void f_PosSp_MelodyPlay(int ToneID, short Duration, uint SoundLevel);

Parameters:

ToneID:	0-4
Duration:	duration [ms]
SoundLevel:	one of the value in BUZZER_SOUND enumeration.

Description:

This plays 1 of the 5 predefined melody for the given duration. at the buzzer.

f_PosSp_MelodyStop

Prototype:

public static extern void f_PosSp_MelodyStop();

Parameters:

None.

Description:

This stops a playing melody.

f_PosSp_WiFiPoweron

Prototype:

public static extern bool f_PosSp_WiFiPoweron();

Parameters:

None.

Description:

This powers up the WiFi device.

f_PosSp_WiFiPowerdown

Prototype:

public static extern bool f_PosSp_WiFiPowerdown();

Parameters:

None.

Description:

This powers down the WiFi device.

f_PosSp_WiFiReset

Prototype:

public static extern bool f_PosSp_WiFiReset();

Parameters:

None.

Description:

This resets the WiFi device.

f_PosSp_GpioIni

Prototype:

public static extern bool f_PosSp_GpioIni();

Parameters:

None.

Description:

This initializes the 4 GPIOs & set them to HI.

f_PosSp_GpioUnini

Prototype:

public static extern bool f_PosSp_GpioUnini();

Parameters:

None.

Description:

This un-initializes the 4 GPIOs.

f_PosSp_GpioSetIo

Prototype:

public static extern bool f_PosSp_GpioSetIo(int iGpio);

Parameters:

iGpio: Set the GPIOs (0-3) to HI.

Description:

Set the IO of Gpio 0--3.

f_PosSp_GpioWrite

Prototype:

public static extern bool f_PosSp_GpioWrite(int iGpio, char iState);

Parameters:

iGpio: index of GPIOs. Valid values are 0 to 3.

iState: the state (1==HI or 0==LO to be written.

Description:

This writes HI or LO state to a GPIO.

f_PosSp_GpioRead

Prototype:

public static extern bool f_PosSp_GpioRead(int iGpio, ref char piState);

Parameters:

iGpio: index of GPIOs. Valid values are 0 to 3.

piState the state (1==HI or 0==LO to be read.

Description:

This reads the current state of a GPIO.

ClsSysUtil:

Overview:

This is a C# class in the ClslibSysUtil library.

It provides the C# managed interface for some OS utilities.

C# namespace:

ClslibSysutil .

Dependencies:

coredll.dll;

iphlpapi.dll;

Type Definitions:

LMEM_ZEROINIT.

Prototype:

const int LMEM_ZEROINIT = 0x40;.

Description:

Equivalent to the LMEM_ZEROINIT flag in WinCE

Structures:

None.

Macros:

None.

Function Definitions:

f_LaunchBlockingApp

Prototype:

public static void f_LaunchBlockingApp(string strPath, string strParms)

Parameters:

strPath: path and executable filename

strParms: .paramter list

Description:

This lauch a blockiong App Launch in a new process

.

f_Ping

Prototype:

public static int f_Ping(string addr,ulong udCount, ref string strResult).

Parameters:

addr: IP address of the destination.
udCount: number of ping packets.
strResult: the ping response text.
Description:
This pings an (url or ip) address using ICMP ipv4 packets

PInvoke lib

The pda also uses some of the generic (non OEM driver specific) APIs from Microsoft PInvoke sample library (2004)

7.4 Application Scenarios with Program Source Codes

Additional end-to-end application scenarios with program source codes will be supplied and included here in future versions of user manuals. They include:

1. Authentication of Tag by Real Time Server Inquiry
2. Collection of Inventory Data and Upload End of Day
3. Dock door fixed reader complement read and search for missing tags
4. etc.

7.5 Unit Tests

Basic unit tests for performance tuning will be added and included here in future versions of user manuals. The demo application itself can of course be treated as a bundle of unit test programs to allow the user to characterize the performance of the handheld RFID reader, particularly in the area of RFID reader to tag relationship. Additional unit tests, such as LED related test, native code interoperability, network connection related tests, are offered.

1. TestGslSdk:

Introduction:
TestGslSdk let the user to experience some of the IO & system devices available on the WinCE platform.
TestGslSdk is a simple C# application; it can:
turn on/off the on-board LED.
play a tone (single-tone-generation) or a build-in melody at a given volume & duration.
start, restart, stop the WiFi device.
get the IPAddress from DNS, ping a remote server.
get the current battery level.
control 4 GPIOs.
control (set options, open, send data, receive data, close) the 2 serial port.
TestGslSdk can be enhanced to a system testing application for field-test, in-factory QA purpose.

Known Limitations:
Power notifications cannot be used on this platform, use polling instead.
High-Level Callfow:
.
Results:
Unit test passed..
Issues & Suggested Solution:
Nil.

2. Native Code Interoperability - Reset Suspend/PDA

Overview:

C# to C++ Interoperability is done by using DllImport (see msdn)

This sample shows how to reset / suspend the pda

Interface Definition:

Dependencies:

Coredll.dll

Function Definitions:

GwesPowerOffSystem:

Prototype:

public extern static void GwesPowerOffSystem();

Description:

This suspends CS101's pda.

ResetPocketPC:

Prototype:

public uint ResetPocketPC()

Description:

This resets CS101's pda using standard KernelIoCtrl Procedure

Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
namespace CsDevReset
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            ResetDevice();
```



```
            }
        ////////////////////////////////////////////////////////////////////////////////
            public const uint FILE_DEVICE_HAL = 0x00000101;
            public const uint METHOD_BUFFERED = 0;
            public const uint FILE_ANY_ACCESS = 0;
            [DllImport("Coredll.dll")]
            public extern static uint KernelIoControl
            (     uint dwIoControlCode,
                  IntPtr lpInBuf,
                  uint nInBufSize,
                  IntPtr lpOutBuf,
                  uint nOutBufSize,
                  ref uint lpBytesReturned
            );
            //void GwesPowerOffSystem(void); //This API is OEM implementation dependent
            [DllImport("Coredll.dll")] // a suspend operation on GslPos pda
            public extern static void GwesPowerOffSystem();

            public uint ResetPocketPC()
            {   /// KernelIo may be called by CF .NET only.
                  uint bytesReturned = 0;
                  uint IOCTL_HAL_REBOOT = CTL_CODE(FILE_DEVICE_HAL, 15,
METHOD_BUFFERED, FILE_ANY_ACCESS);
                  return KernelIoControl(IOCTL_HAL_REBOOT, IntPtr.Zero, 0,
                     IntPtr.Zero, 0, ref bytesReturned);
            }
            private void ResetDevice()
            {
                  DialogResult r = MessageBox.Show( "Execute reset(Yes) or PowerDown(No)?", "Test",
MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2 );
                  if (r == DialogResult.Yes)
                  {     ResetPocketPC();
                  }else{
                        GwesPowerOffSystem();
                  }
            }
      }
}
```


7.6 Build Environment

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 handheld and also on the PC. The following 14 steps are needed to set up the build environment.

Basic Configuration on PC:

1. **Operating System:** Microsoft Windows XP Professional SP2 installed.
2. **Microsoft Visual Studio 2005 Professional Edition SP1 (MSDN subscription)**
3. **Microsoft Platform SDK for Windows Server 2003 SP1 (Free download)**
   http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en
4. **Make sure that Microsoft .Net (Compact) Framework/Smart-Device-Programming is included during VS2005 setup.**
5. **Install SP1 patch for .Net Framework 2.0**
   http://www.microsoft.com/downloads/details.aspx?familyid=79bc3b77-e02c-4ad3-aacf-a7633f706ba5&displaylang=en&tm
6. **For Remote Debugging**

ActiveSync 4.5.0 enables remote debugging on handheld reader via USB (Free Download)
   https://www.microsoft.com/windowsmobile/downloads/eula_activesync45_1033.mspx?ProductID=76

7. **Microsoft .Net Compact Framework 2.0 SP1 (installed during VS2005)**
   http://www.microsoft.com/downloads/details.aspx?FamilyID=7befd787-9b5e-40c6-8d10-d3a43e5856b2&displaylang=en
8. **Microsoft Mobile 5.0 SDK for Pocket PC (Free Download)**
   http://www.microsoft.com/downloads/details.aspx?familyid=83A52AF2-F524-4EC5-9155-717CBE5D25ED&displaylang=en
9. **GSLPOS SDK**
   Unzip the package gsl_sdk.zip to a temporary folder
   Run the GSLPOS_SDK.msi from gsl_sdk\20070130-POS-PDA-SDK-D4\
10. **SQL Server CE**
   Microsoft SQL Server 2005 Compact Edition Developer Software Development Kit (Free download)
   http://www.microsoft.com/downloads/details.aspx?FamilyId=E9AA3F8D-363D-49F3-AE

89-64E1D149E09B&displaylang=en

Remember the installation directory if different from default.

11. **SQL Server 2005 (optional)**

    Microsoft SQL Server 2005 Express Edition (Free Download)

    http://go.microsoft.com/fwlink/?LinkId=65212

    Microsoft SQL Server Management Studio Express (Free Download)

    http://go.microsoft.com/fwlink/?LinkId=65110

    Download page:

    http://msdn2.microsoft.com/en-us/express/bb410792.aspx

12. **Documentation**

    MSDN Library for Visual Studio 2005 (Free to download)

    http://www.microsoft.com/downloads/details.aspx?familyid=B8704100-0127-4D88-9B5D-896B9B388313&displaylang=en

13. **Microsoft® Windows® CE 5.0 Device Emulator (Free download)**

    http://207.46.19.190/downloads/details.aspx?FamilyID=a120e012-ca31-4be9-a3bf-b9bf4f64ce72&displaylang=en

14. **Windows CE 5.0: Standard Software Development Kit (SDK) (Free download)**

    http://www.microsoft.com/downloads/details.aspx?FamilyID=A08F6991-16B0-4019-A174-0C40E6D25FE7&displaylang=en

7.7 Debug Methods

7.7.1 Log File

The log file provides an important method to track problems. The log file should be captured and sent to CSL support team asap if any bug is encountered.

7.7.2 Error Message List

The list of error messages that can be seen on the screen will be listed and included here in future versions.

8 PC Side Demo Programs

8.1 Introduction

8.2 Database Files Manipulation Demo

There is a Windows-based program to help manipulate data collected from the handheld reader.

8.2.1 Installing Demo Program

Please make sure the demo program version is compatible with the firmware version of reader. Refer to the file "compatibility matrix.xls" for the compatibility of demo program and reader firmware.

Please make sure "Microsoft .NET Framework Version 2.0 Redistributable Package" is installed before using the demo program.

Normally, the executed file of demo program is archived as RAR or ZIP file. The archived file is distributed through email, ftp server or website.

Please extract the demo program to a directory (e.g. "C:\CS101-2DEMO\"). Then, run the demo program from the installed directory.

8.2.2 Using Demo Program

Run the demo program from installed directory. More details will be included in future versions of the user manual.

9 Upgrade of Software in CS101-2

9.1 Introduction

Upgrade of software will be described in this chapter.

9.2 Upgrade of Demo Application

CS101-2 comes with a demo application. This demo application upgrade file is a CAB file that can be copied to the RFID reader using simple activesyn and explore. Once there, the user simply double click the CAB file and the installation is done.

9.3 Upgrade of RFID Library

The CS101-2 RFID library is a dll. This can be upgraded also using a CAB file. The library upgrade file is a CAB file that can be copied to the RFID reader using simple activesyn and explore. Once there, the user simply double click the CAB file and the installation is done.

9.4 Upgrade of RFID Firmware

The low level firmware is upgraded using a firmware upgrade program. Again, a CAB file is copied to the RFID reader, then the CAB file is double clicked, then a binary image is copied to the RFID reader. After that the user can run the firmware upgrade program to move that binary down to the low level processor.

10 Usage Tips for CS101-2

10.1 Introduction

The objective of this chapter is to recommend the best practices of using the CSL CS101-2 Reader. The following areas will be covered in this document

- General usage
- Write tag
- Event and alert
- System

10.2 General Tips

CS101-2 is designed to give long read range and high read rate operations for situations where a fixed reader may be inconvenient, or a fixed reader may lack the needed angular and distance flexibility, or a fixed reader design may require too many units whereas a handheld reader can be easily moved around for lower initial infrastructure cost.

10.3 System Tips

The CS101-2 host processing unit is a basic PDA design utilizing WinCE operating system. Since this host processing unit is only 400 MHz clocked, one should not expect it to work as fast as a PC. A lot of consideration of system computation power and connection bandwidth must be taken in the most graceful manner.

10.4 Write Tag Tips

There are many ways to write tags using the EPC Gen2 protocol. Halt filter is among one of the tool to speed up the process. One should study the EPC Gen2 document intensively to understand the meaning of each parameter to enable the most efficient use of this technology. The more proficient a system integrator is on EPC Gen2 protocol, the more successful it can be.

11 RFID Cookbook

11.1 Introduction

RFID (radio frequency identification) is a wireless means to obtain a unique ID that can identify a product (similar to barcode that however requires optical line of sight). Since 2004, it was applied by companies in USA and Europe successfully to various business processes and brought major cost benefits. Because of the success of these early adopters, such as Walmart (USA) and Mark & Spencer (Europe), there is a growing trend throughout the world to replace barcode (or augment) with RFID. The advantages of RFID over barcode are widely publicized, consisting of the following:

Features	**RFID**	**Barcode**
Line of Sight	Line of sight is not required	Must be line-of-sight visible – items must be tediously separated out for reading, very inconvenient
Storage	Store data up to 1 Kbyte	No storage capability
Anti-Counterfeit Ability	Hard to counterfeit, hard to find (can be stowed inside item)	Easy to counterfeit, always exposed outside and therefore easy to copy
Processing Speed	Automatic processing possible at very high speed	Processing has to be manual in most cases, with very low speed and throughput
Bulk Reading	Many tags can be read at the same time – virtually parallel reading	Must be read sequentially
Durability	Durable, usually safely stowed inside item.	Easily scratched, wrinkled or wetted beyond reading.

RFID can be applied with the following purposes:

1. Supply chain optimization
2. Asset tracking
3. Inventory control
4. etc.

Benefits of RFID include:

1. Increase supply chain velocity
2. Reduce human involvement (cost, error, hiring cycle and other issues)
3. Enhanced visibility (tracking, scheduling, planning)
4. Enhanced security (total visibility monitoring, zonal tracking)
5. Real time supply chain re-route (dynamic multi-destination fulfillment)
6. etc.

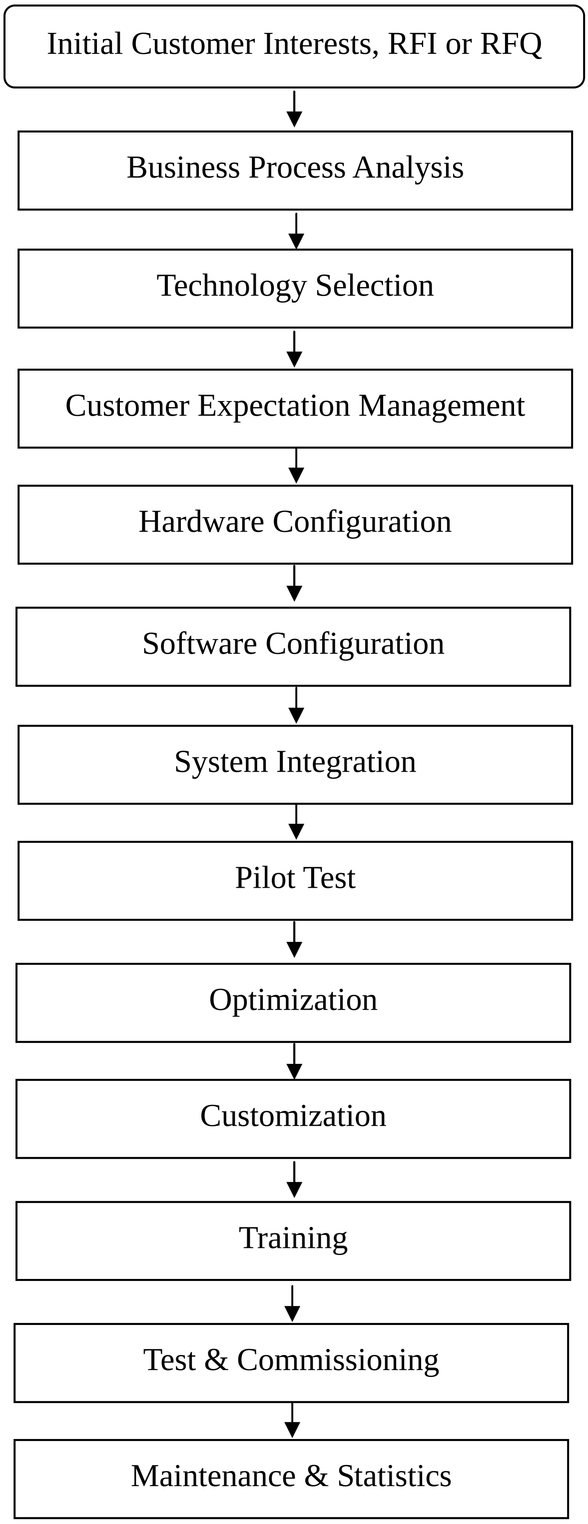
Physical locations where RFID can be applied include:

1. Distribution centers
2. Warehouses Shelves
3. Warehouse Loading/Unloading Zone (Yard Management)
4. Retail shops in conjunction with fulfillment center
5. Returns & warranty processing office
6. Vehicle windshields
7. etc.

It is widely believed that the adoption of RFID will happen in the following sequence in terms of company category:

1. Mandate affected units (suppliers to Walmart, DoD, etc.)
2. High value products
3. Fast moving assets
4. etc.

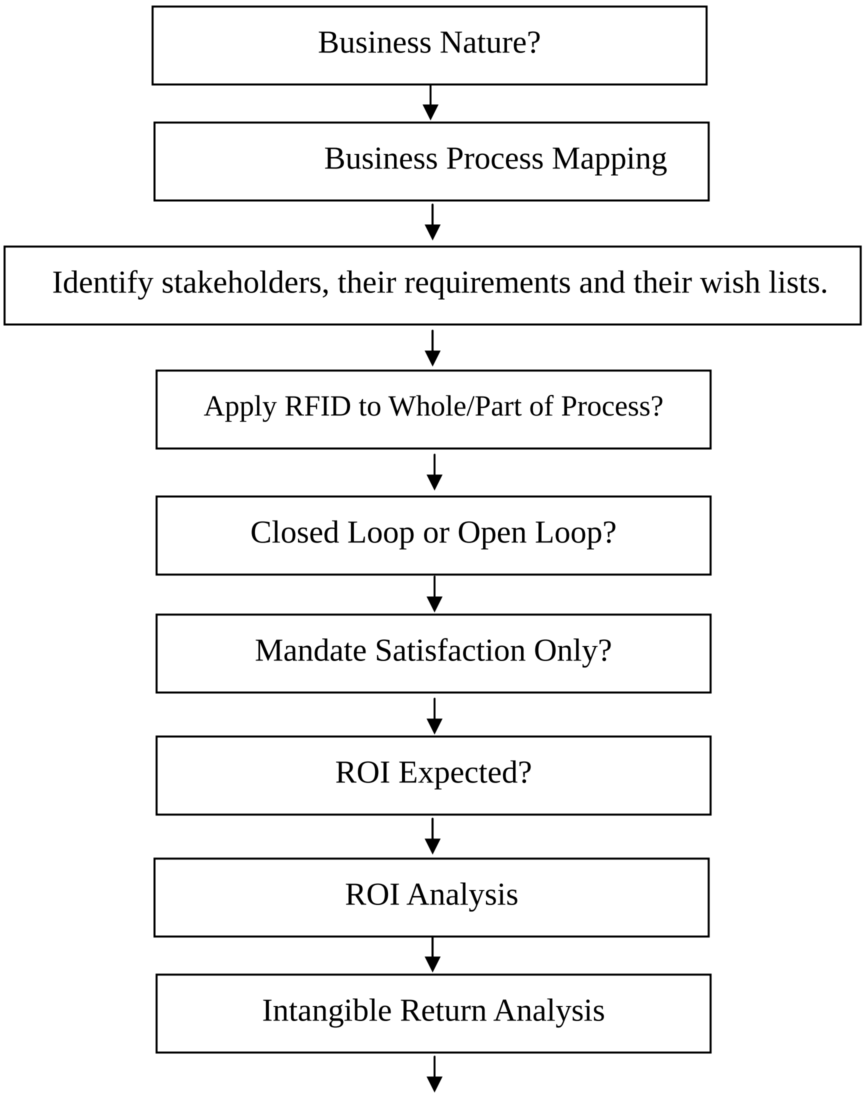
The application of RFID to a company or a group of companies in a supply chain has to be executed systematically and methodically. The following is a flowchart that describes a typical application process:

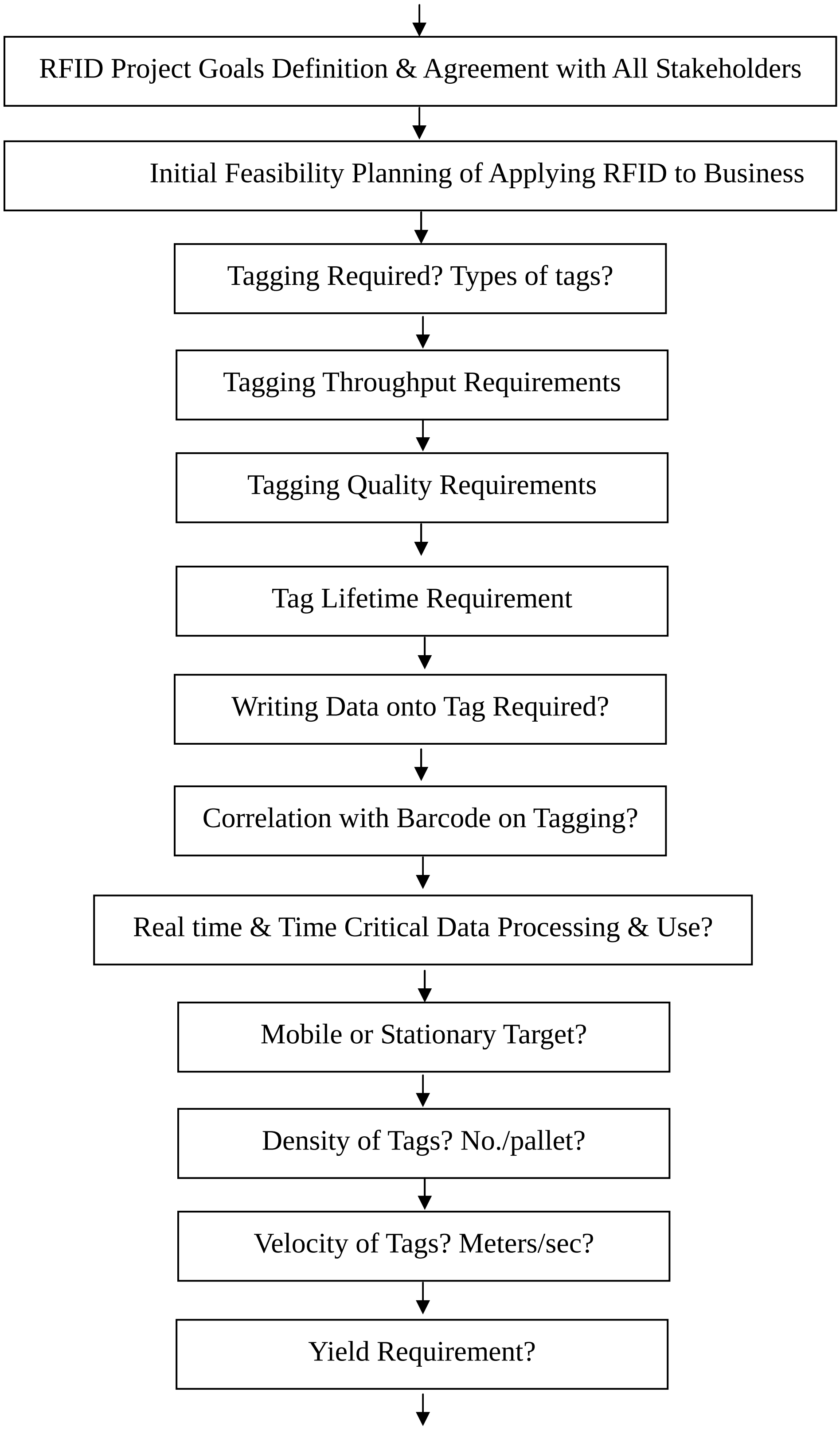

Initial Customer Interests, RFI or RFQ
Business Process Analysis
Technology Selection
Customer Expectation Management
Hardware Configuration
Software Configuration
System Integration
Pilot Test
Optimization
Customization
Training
Test & Commissioning
Maintenance & Statistics

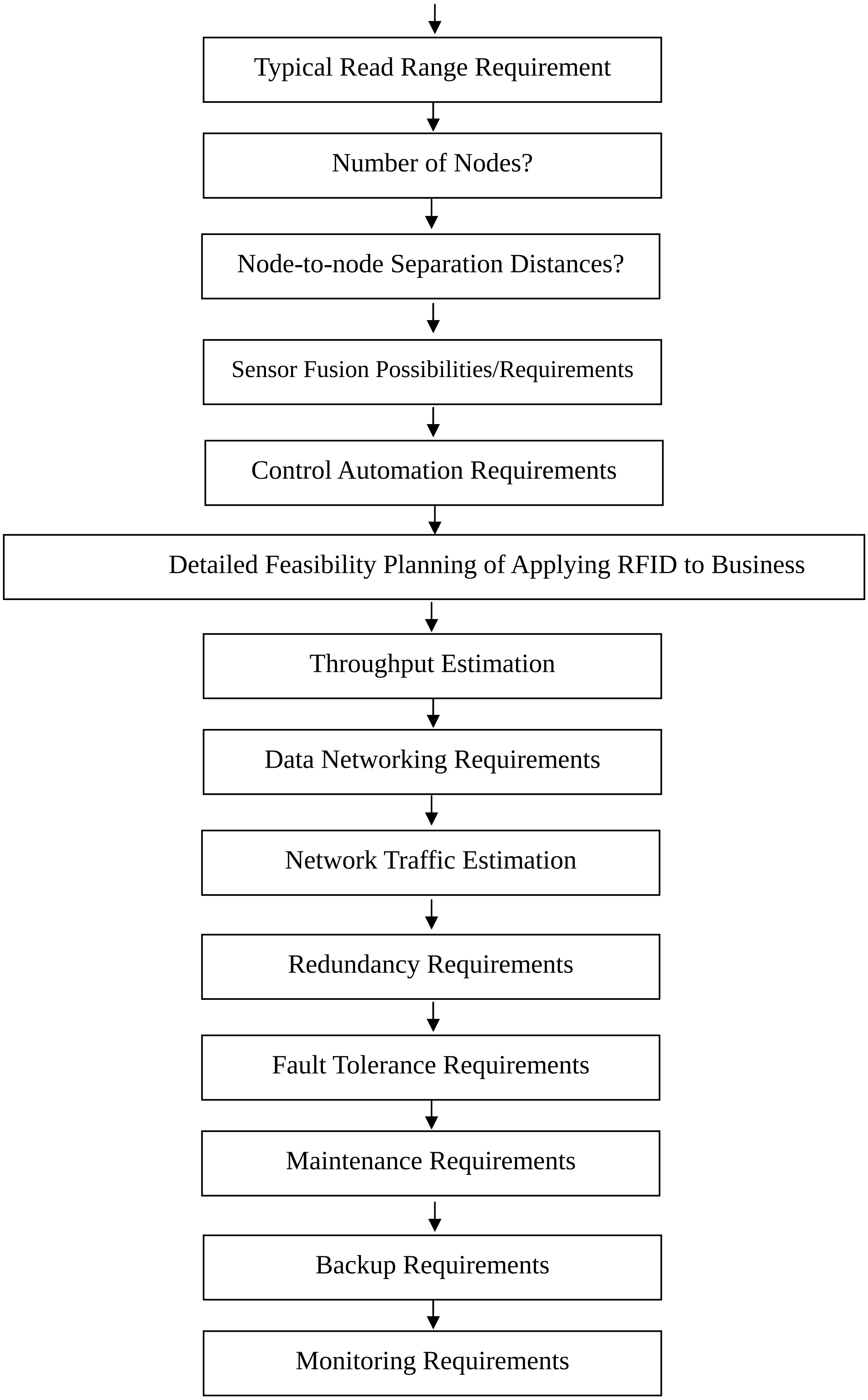
11.2 Application Details

11.2.1 Business Process Analysis

The business process of the customer must be analyzed carefully to find places where the RFID tagging and reading can occur. The system integrator may be applying RFID to the whole process or may only be able to apply RFID to part of the process. The most important principle is NOT to force change the business process to adapt for RFID implementation, but to have RFID implementation slip in as effortlessly and as un-noticeably as possible.

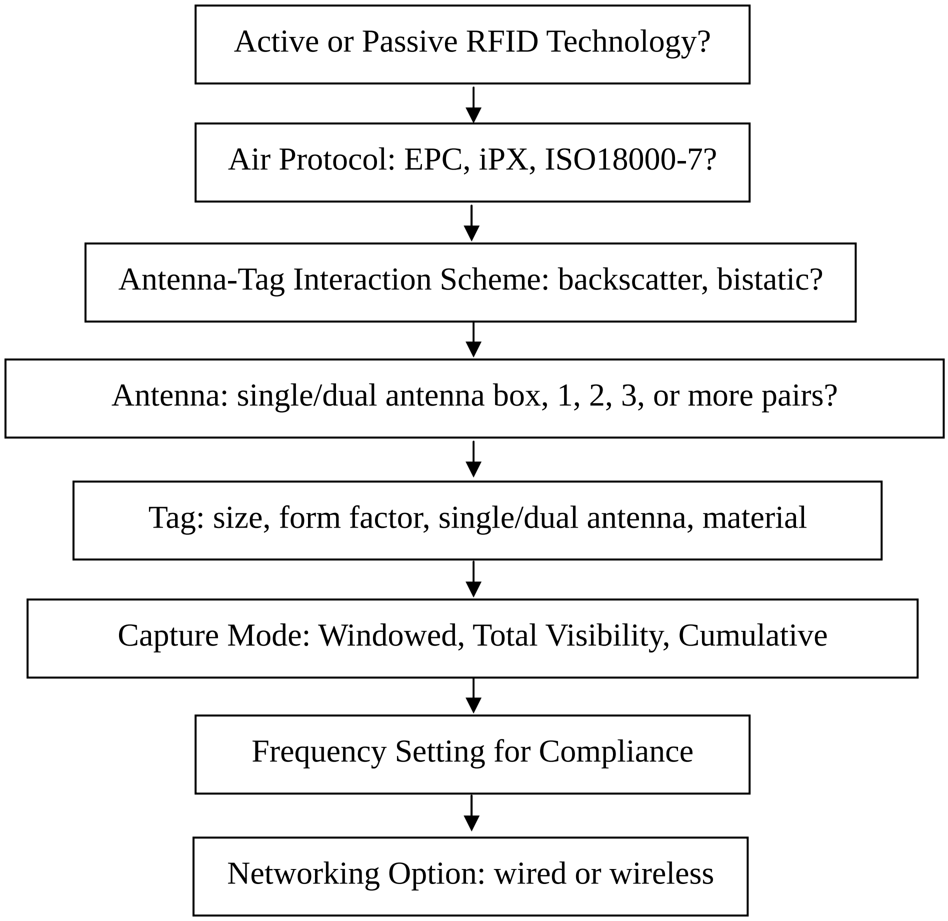



RFID Project Goals Definition & Agreement with All Stakeholders
Initial Feasibility Planning of Applying RFID to Business
Tagging Required? Types of tags?
Tagging Throughput Requirements
Tagging Quality Requirements
Tag Lifetime Requirement
Writing Data onto Tag Required?
Correlation with Barcode on Tagging?
Real time & Time Critical Data Processing & Use?
Mobile or Stationary Target?
Density of Tags? No./pallet?
Velocity of Tags? Meters/sec?
Yield Requirement?


Typical Read Range Requirement
Number of Nodes?
Node-to-node Separation Distances?
Sensor Fusion Possibilities/Requirements
Control Automation Requirements
Detailed Feasibility Planning of Applying RFID to Business
Throughput Estimation
Data Networking Requirements
Network Traffic Estimation
Redundancy Requirements
Fault Tolerance Requirements
Maintenance Requirements
Backup Requirements
Monitoring Requirements

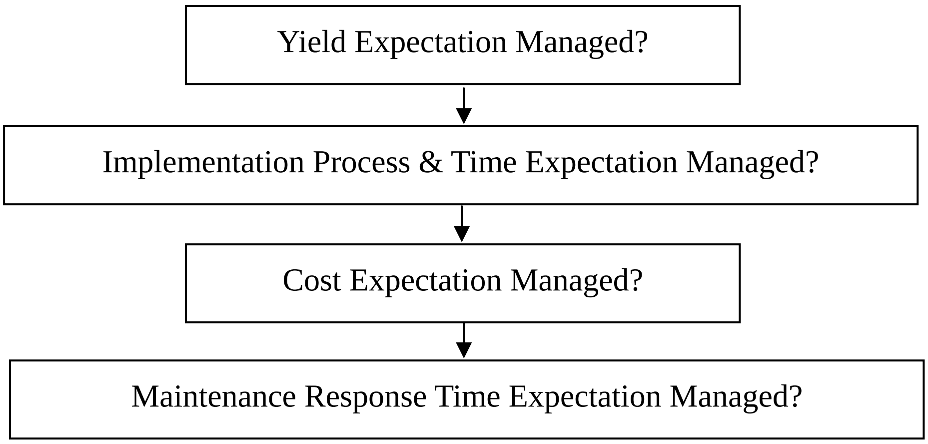
11.2.2 Technology Selection

Once the points where the business process allows for RFID implementation is found, the most appropriate technology must be chosen for the job. The following are questions to help you choose the appropriate technology:



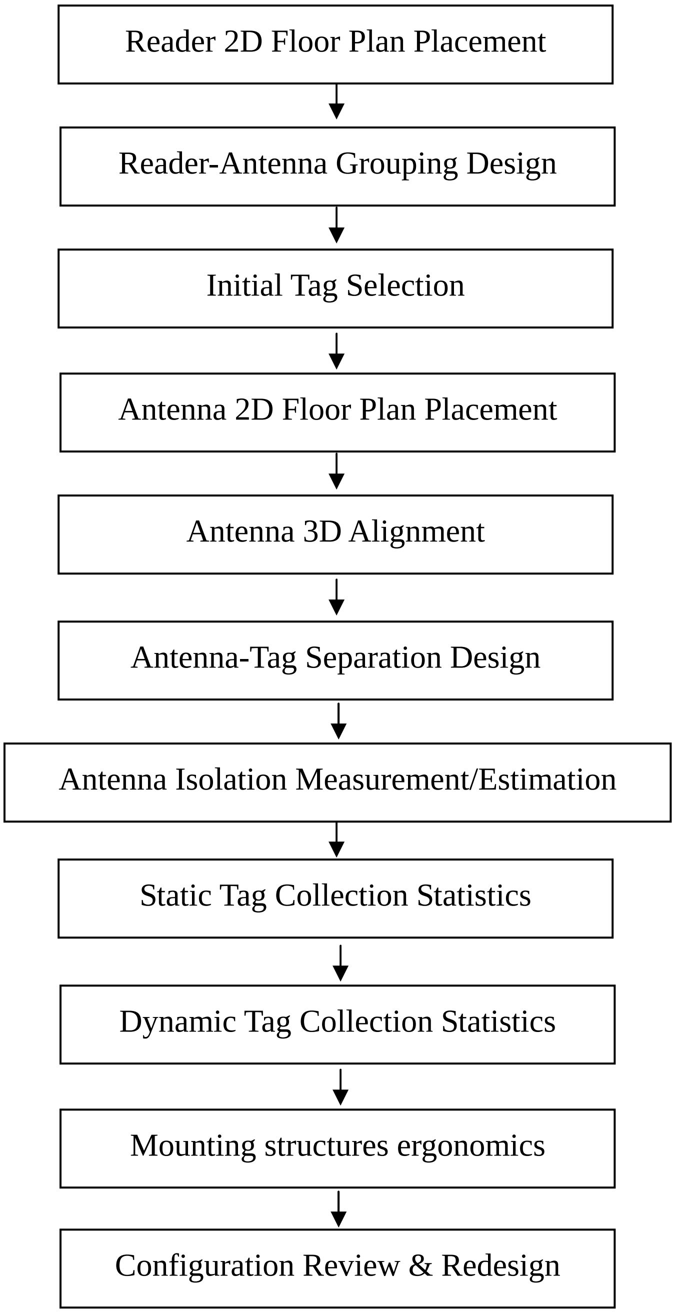
11.2.3 Customer Expectation Management

Customer expectation must be well managed. It is the job of the system integrator, particularly the sales person, to warn the customer away from expecting perfect scores. The truth is, even if 100% read is not achieved, the user can still benefit (in the sense of ROI, efficiency, lead time, cycle time, etc.) to a substantial extent. It is this extent that should be considered as the result, not a 100% score. It is almost like getting married to a man or woman – you will never find the perfect half, but even if she or he is not perfect, you still get to enjoy from the marriage.



11.2.4 Hardware Configuration

Hardware configuration consists of designing and defining what reader, antenna and tag combination will be implemented at each of the nodes in the business process. It is not a pure drawing board exercise, as some kind of minimally realistic testing must be implemented even at this stage to help better define the hardware configuration that in turn can give more insight for software configuration and system integration.



11.2.5 Software Configuration

Software configuration of the reader is very important – it ensures the reader will operate exactly as the business process requires, not more or not less.

The following page has a flowchart that the system integrator needs to go through in order to set up the software.

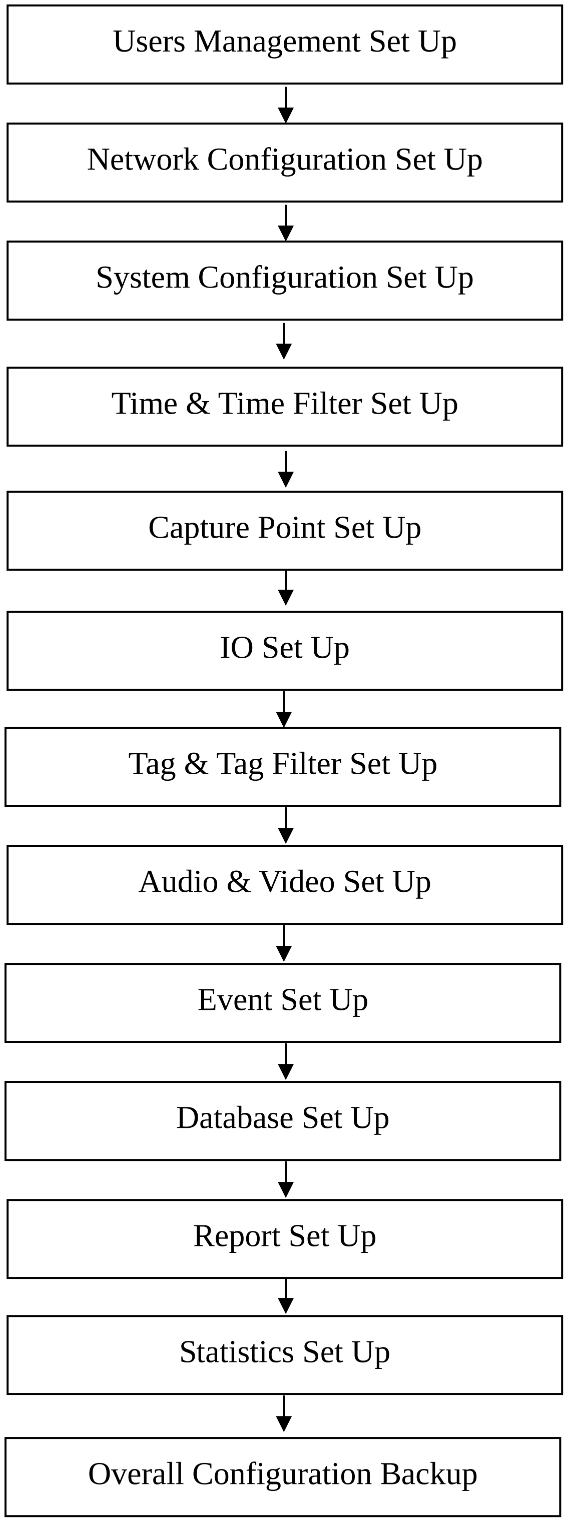
The first step is to configure the users parameter, such as operator name, ID, password, etc. The second step is to configure the networking parameters, such as IP addresses, access point SSID, etc. The third step is to configure system parameters, such as reader ID, frequency setting, tag baud rate, capture mode, etc.

The third step is to configure time and time filter, such as system date and time (hour, minute and second), time filter (define various time intervals, time slots, repeat modes), etc. The fourth step is to configure capture point, such as capture point type, capture point area, capture point details.

The fifth step is to configure IO, such as sensor input name, control output name, default positions, etc. The sixth step is to configure tag and tag filtering, such as tag group, tag filter, etc. The seventh step is to configure audio and video, such as audio messages and video messages resident path (remote or local).

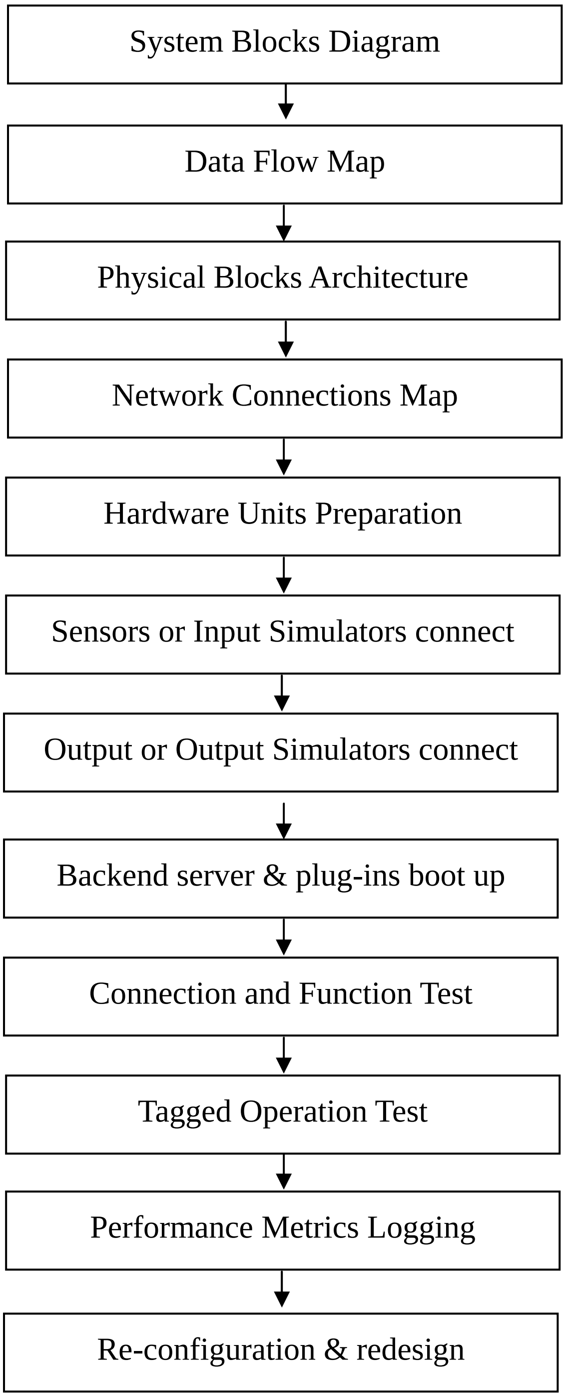
The eighth step is to configure event, such as triggering logic, resultant action, event sequencing, etc. The ninth step is to configure database, such as database fields, etc. The tenth step is to configure report, such as report definition, etc.

The eleventh step is to configure statistics, such as parameters for long term monitoring, etc. The twelfth step is to back up the set up into a standard configuration set up file.


Users Management Set Up
Network Configuration Set Up
System Configuration Set Up
Time & Time Filter Set Up
Capture Point Set Up
IO Set Up
Tag & Tag Filter Set Up
Audio & Video Set Up
Event Set Up
Database Set Up
Report Set Up
Statistics Set Up
Overall Configuration Backup

11.2.6 System Integration

The actual system integration should most desirably be carried out in two steps: 1. in house integration and test; 2. onsite integration and test.

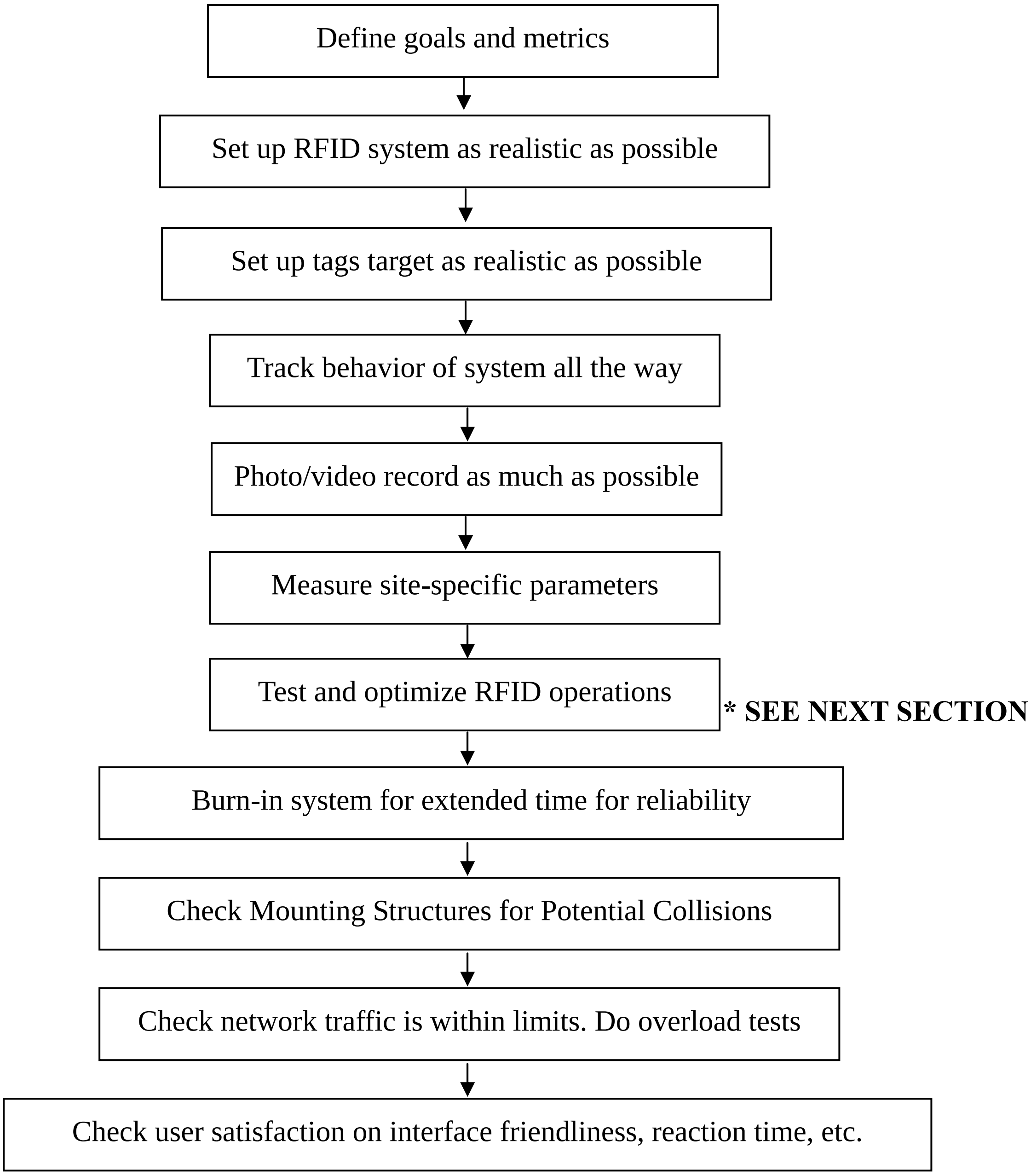


11.2.7 Pilot Test

Pilot test must of course be done on site. The unique building infrastructure and environment of the end-customer venue can result in dramatically different performance (worse, usually) scores compared to that in the system integrator’s own office. Therefore pilot test must be done on site.

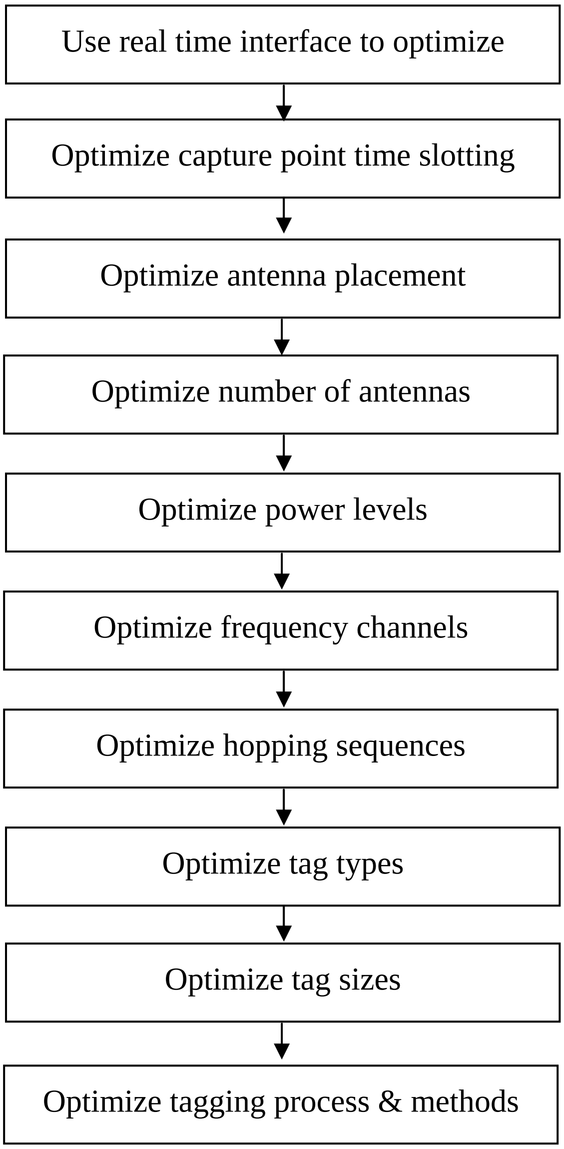
The system integrator, when testing the set up in end-customer’s venue, should endeavor to put the set up directly at the position that it plans to be, or in a place that most closely resembles that of the final site. If the site does not run round-the-clock shifts, then it is OK to do the initial testing when it is off-shift and temporarily clearing up the site for testing (if something is in the way). Eventually when good enough results are obtained through tuning and optimization in off-shift time, then the testing should be conducted in the actual shift when the operation will happen in the future. The emphasis on having the environment as real and true as possible is due to the fact that wireless emission is a very site specific and dynamic event. The propagation and scattering behavior is different from site to site. The noise floor can be different in the day and in the night. There is no pilot test better than doing it right at the spot and right at that time.

The following are basic steps for pilot testing (please also refer to next section of optimization):


Define goals and metrics
Set up RFID system as realistic as possible
Set up tags target as realistic as possible
Track behavior of system all the way
Photo/video record as much as possible
Measure site-specific parameters
Test and optimize RFID operations
* SEE NEXT SECTION
Burn-in system for extended time for reliability
Check Mounting Structures for Potential Collisions
Check network traffic is within limits. Do overload tests
Check user satisfaction on interface friendliness, reaction time, etc.

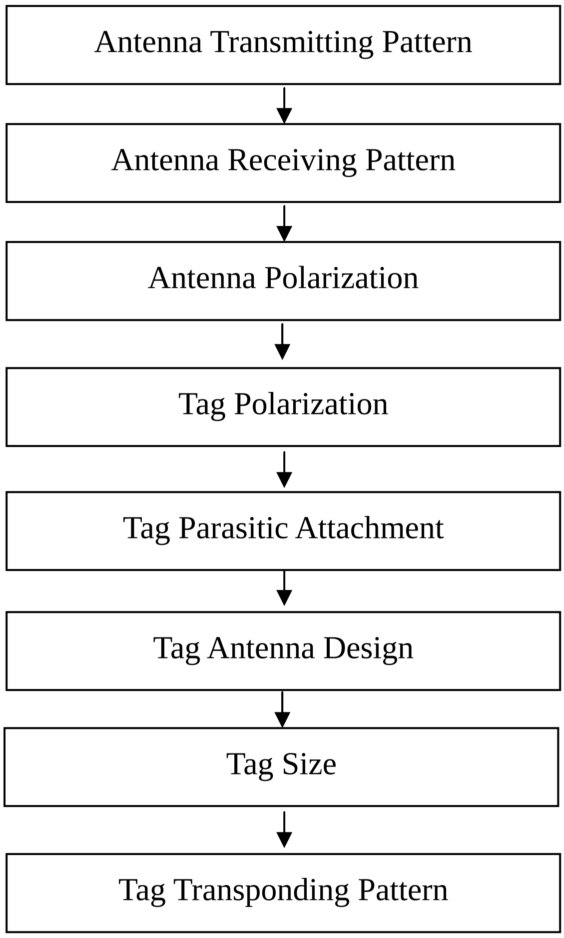
11.2.8 Optimization

Optimization of the performance of the RFID application in business processes is the most difficult step. It is in this step where the variation of performance caused by the law of physics has to be tackled. The following are a few questions that may help. However, due to the unfortunate fact that RFID application involves too many topics: RF transmitter circuits, antennas, propagation (static and dynamic), scattering (backscatter and bistatic scattering), RF receiving circuits, software (all layers), it is not an easy task to give a "10 steps to successful RFID implementation" rule based implementation guideline that works in all environment!



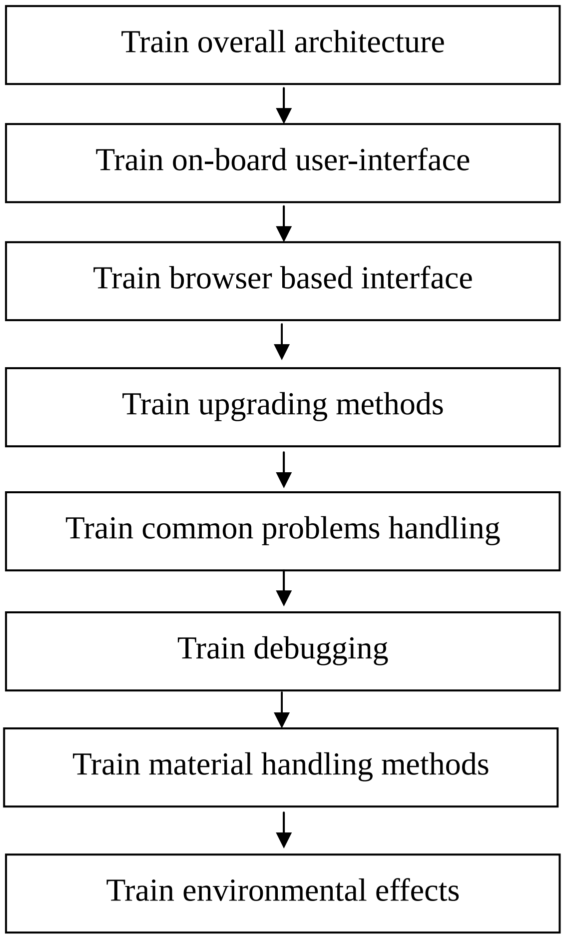
11.2.9 Customization

Customization is the step that comes out of optimization. If, after intense optimization, the performance still is not acceptable (or the customer will not accept a lowering of their performance expectation), then some customization may be necessary. The following are just a few possibilities and suggestions for customization. Note that these customizations require the cooperation of the solution provider (i.e. the manufacturer of the products). Very few solution providers are willing to do this without good business justification, though.



11.2.10 Training

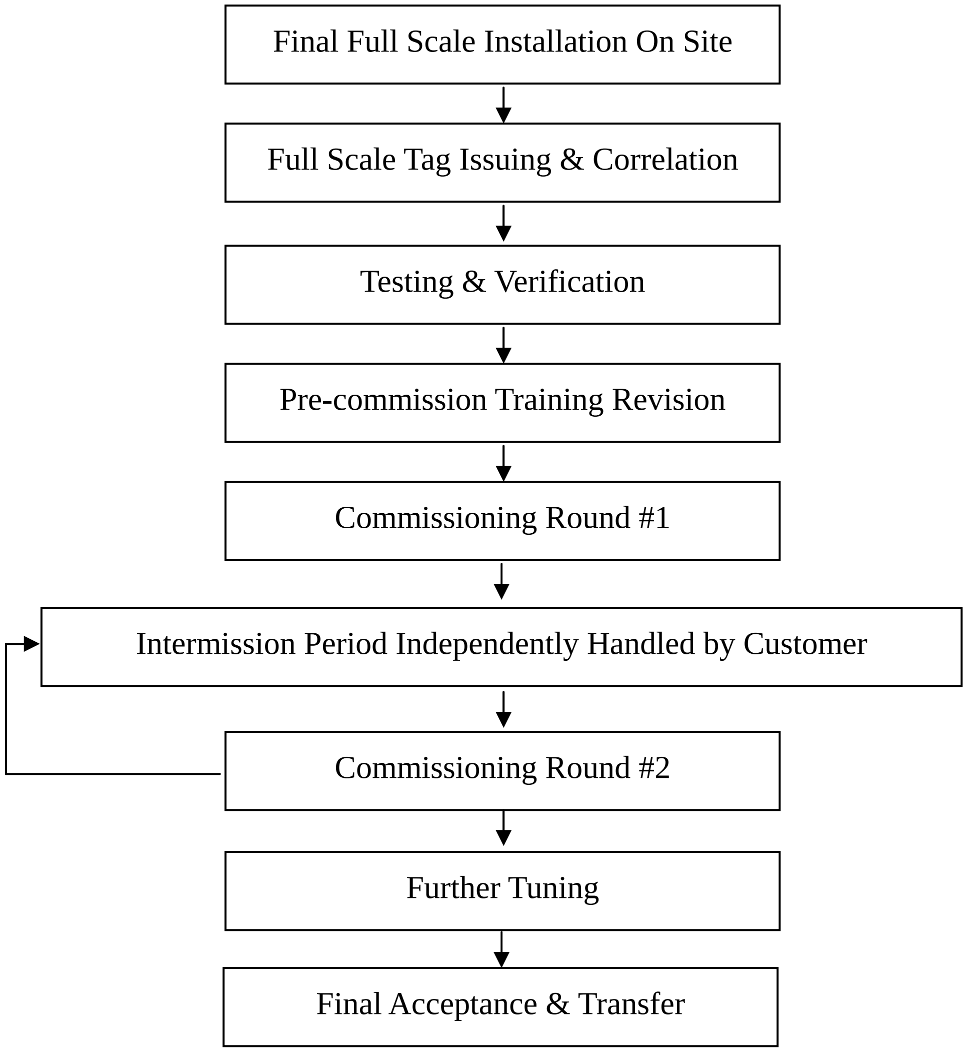
Training is an extremely important step where the operators of the RFID system in the end-customer company must be taught the basics of the operation, plus the necessary tricks in day-to-day trouble shooting and fault isolation – up to a certain extent, of course.



11.2.11Test & Commissioning

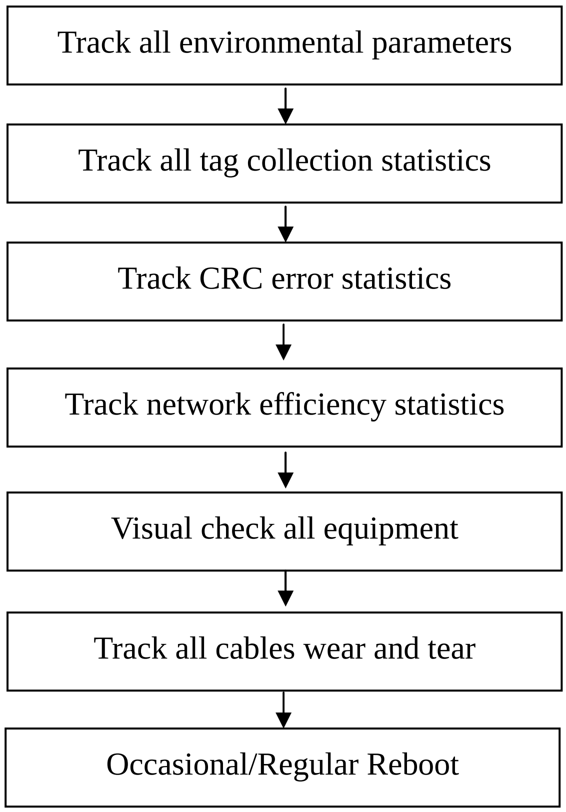
Test and commissioning is an important step to allow customer to verify the performance achieved, and formally approve the system to enter operational status. The most important part of test and commissioning is of course a mutually agreed test plan and commissioning criteria.

The experienced system integrator can probably propose this test and commissioning plan early in the project. This is particularly valid if the system integrator has done similar jobs before. However, sometimes a T&C document too early in the way will make it very difficult to accommodate for surprisingly low performances due to some uncontrollable environmental or business process related factors. So really it is at the system integrator's own discretion and wisdom when it should best be proposed.



11.2.12 Maintenance & Statistics

Maintenance of the RFID system is important. It includes preventive maintenance, collection and analysis of statistics of operation, etc.



11.3 Readers for Different Business Applications

For different business applications, one should use the appropriate corresponding readers, such as multiport fixed reader, integrated reader, handheld reader, embedded reader module, etc.

Products	Part Number	Photo	Business Application
Fixed RFID Reader	CS461-N N= 1: Europe CE 2. USA FCC 3. Japan Telec 4. Taiwan NCC 5. Australia ACMA 6. Korea MIC 7. China SRRC		Logistics Warehouse management Distribution center Transportation management Asset management Baggage management
Handheld RFID Reader	CS101-N N= 1: Europe CE 2. USA FCC 3. Japan Telec 4. Greater China (China SRRC, Taiwan NCC, & HK OFTA) 5. Korea MIC		Logistics Warehouse management Distribution center Transportation management Asset management Baggage management

11.4 Antennas for Different Business Applications

Various antennas have been designed and optimized for different business processes, such as dock door, ware house, access control, and item level tracking.

Products	Part Number	Photo	Business Application
Antenna (Mono-static area or zonal antenna, long range)	CS-771-LHCP CS-771-RHCP		Logistics Warehouse management Distribution center Transportation management Asset management Baggage management
Antenna (Monostatic access control antenna)	CS-713		Access control Human & animal tracking
Antenna (Brickyard near-field antenna)	CS-777		Retail shop POS Document management Blood bag management Pharmaceutical bottle tracking

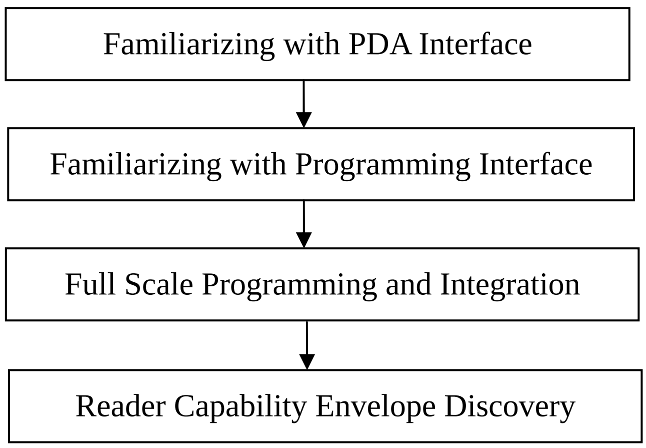
12 RFID Best Practices

12.1 Introduction

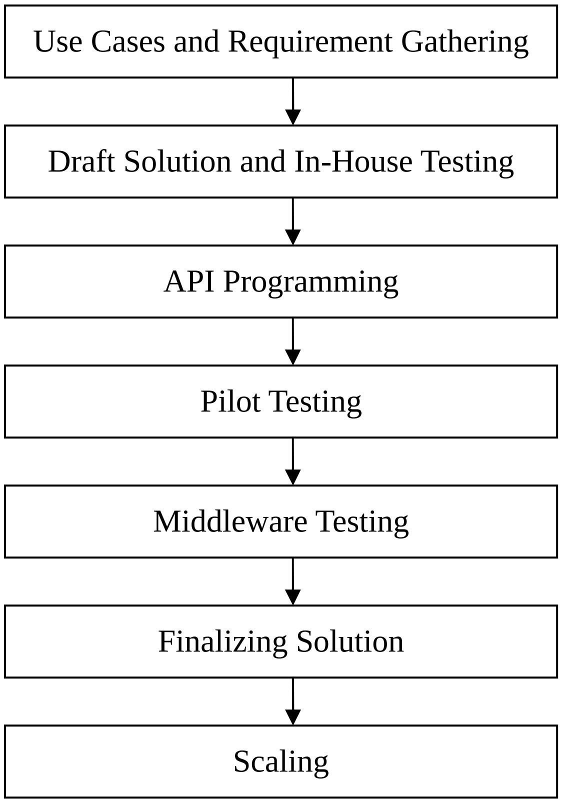
System integration of RFID operation is not a simple task. It involves processes such as software configuration, hardware setting, pilot testing, scaling, and more. A good integration is a crucial step to ensure successful ROI for the RFID investment. Improper integration process could affect the system performance as well as functionality. This section describes the best practice for system integrator to familiarize and integrate with an RFID reader, from getting the reader out of the box to deploying the system in production environment.

The following flowcharts show the typical familiarization and integration process of CSL CS101-2 reader. They represent what a typical system integrator will go through when they adopt the CS101-2 technology. By following the path described, the system integrator can quickly deploy CS101-2 and earn revenue within a very short period.

1. Familiarization Process



2. Integration Process



12.2 Integration Process Details

12.2.1 Familiarization Process

12.2.1.1 Familiarizing with PDA Interface

The CSL CS101-2 reader comes with a PDA interface. This PDA interface is based on WinCE and is thus very familiar to many programmers and users. Once the reader is powered up, one can invoke the demo CSL program to see how a user interface on the WinCE OS platform can help the user do the RFID operations with ease.

12.2.1.2 Familiarizing with Programming Interface

The CSL CS101-2 reader provides a complete set of Application Programming Interfaces (API).

- Before starting to program the reader, system integrators are recommended to go through the sample codes which are available for download in CSL web site. The sample codes allow ones to learn how to program the reader in a correct and effective way. The example program flow, API request making and result processing give a general idea of how to interface with the reader.

12.2.1.3 Full Scale Programming and Integration

Full scale programming allows one to fully control the reader and receive data from the reader with the final goal of integrating the reader with existing business processes, operations and business intelligence software of the customer, such as middleware, ERP system, database, etc. Every system integrator has his own favorite such program, either developed by themselves or based on platforms available from the market, such as Websphere, Weblogic, Biztalk, SensorEdge, RFIDAnywhere, SAP, Oracle, DB2, Sybase, etc.

Once the system integrator passes through the two initial stages of experimenting with the browser interface and the programming interface, he/she needs to start looking at what subset of API calls are needed to enable RFID use in his/her typical customers' business environment.

The API includes a number of commands with different parameters. When programming the reader, one should understand clearly the command's usage, effect and the meaning of each parameter since they affect the reader performance directly.

12.2.1.4 Reader Capability Envelope Discovery

Once full scale programming is started, the user needs to map out the full "flight envelope" of the reader. Important parameters to figure out includes response time, maximum API sending rate, necessary and optimal combinations and sequences of API to achieve different states of the machines, fastest possible read and/or best possible yields for various profile combinations, etc. Once the capability envelope is discovered, the system integrator can then work on business projects knowing what the reader is capable of doing and knowing the projects are not requiring the reader to do something it cannot handle.

12.2.2 Integration Process

12.2.2.1 Use Cases and Requirements Gathering

Before starting the development process, system integrators should fully understand the requirements from customer, such as the throughput requirement, latency requirement, bandwidth requirement. etc, that are specific to the reader. Besides, they could document the use cases which will help in decision making later on in the development process.

12.2.2.2 Draft Solution and In-House Testing

Once the requirements are gathered and use cases are defined, system integrators can develop a draft solution. Draft solution means that it is subjected to final adjustment or tuning after pilot testing. In-house testing allows system integrators to test the feasibility of the solution before deploying to customer’s site.

12.2.2.3 API Programming

The API Programming process here is different from the one in Familiarization Process. In Familiarization Process, system integrators should familiar with the configurations and functioning modes of the reader by using the API. In System Integration Process, they should determine and focus on the configurations and functioning modes to be used in the solution to fulfill user requirements.

12.2.2.4 Pilot Testing

RFID system is greatly affected by environmental factors. For example, background RF noise and metallic object around may affect the read range of antenna dramatically. The same RFID system may function well in the system integrator’s own office but fail in end-customer’s site. Therefore system integrators should conduct on-site pilot testing.

During the on-site pilot testing, system integrators should tackle the site-specific problems that affect the RFID system. For example, if there is metallic object around, position of the antenna should be adjusted to overcome the effect of it.

Apart from system settings, RFID tags should be tested as well. System integrators should select suitable tags to cater the business requirement. For example, 3D tag can be read from all directions, but it is less sensitive and large in size. Regular tag has better sensitivity but the read result is highly affected by orientation of the tag.

Some problems may not appear instantly, but only after the system running continuously for hours or days. To identify such problems, long time burn-in testing is required. If any problem related to the reader is found, the system integrator could send a bug report with reader settings, antenna setup and site-specific factors to CSL for troubleshooting.

12.2.2.5 Middleware Testing

Usually, a middleware is used between the reader and enterprise application. It plays an important role in the integration of reader and therefore it must be fully tested as well. CSL provides service for such testing. System integrators can give the executable of the middleware to CSL for long term testing to ensure that the middleware is free of problem after running continuously. Moreover, all API calls requested by the middleware are logged in the reader which allows CSL to analyst the cause of problem if there is any.

12.2.2.6 Finalizing Solution

The finalized solution should tackle all of the problems found in pilot test and fine tune the solution if necessary. Then it is ready for production running.

12.2.2.7 Scaling

Scaling process should be done after the system is tested to be stable. Moreover, scaling gradually at the end-customer site (if end-customer permits, of course) can reduce the chance of system failure due to overloading. For a large scale RFID system that involves hundred of readers, the system integrators should pay attention to the followings:

1. Readers that are close to each other are recommended to use Profile 2 or 3 of Modulation Profile. It allows the readers to work in dense reader mode such that jamming could be

avoided. Remember to select different session numbers for readers to avoid tag replying wrongly to other reader.

2. If dense reader mode is not required, Profile 0 should be used as it allows the fastest tag read.

3. Adjust the power of reader to take a balance between read range and cross read effect.

4. Employ inspection process for identifying malfunction reader. For example, reading testing tags from all readers and then collecting the read data from edge server. Analysis of the data helps assessing the reader health.

5. Remote reboot of reader and remote control of power grid should be supported since the readers may distribute in vast area.

6. During network failure, reader is not able to send tags read to trusted server. If Network Failure Data Backlog is enabled, those tags are buffered in the reader. Backlog tags are sent to trusted server after the TCP connection is re-established. Therefore, system integrators should also provide application level failover for this feature.

13 RFID Use Cases for Handheld Reader

13.1 Store Front Daily Inventory

Use Case

In department store with huge amount of inventory arranged in complicated ways, inventory usually is an annual or biannual event. This is however not conducive to good inventory management, especially in view of inevitable shrink in open shop store. The ability to quickly inventory some or all departments can be most useful to reduce empty shelf situations dramatically. This use case is fully proven and documented and publicly aware as in the case of Mark & Spencer in Europe.

Current Approach

Stocktaking is done manually or using barcode system. The process is costly and slow. Inventory data are inaccurate due to human errors.

Suggested Approach

Use a few handheld readers with long read range and high read rate, scan the aisles quickly everyday after store close.

Recommendation

The CSL CS101-2 reader is an extremely long read range and high read rate handheld reader. If used in store front daily inventory, the inventory time will be dramatically reduced, or the overall yield and accuracy will be much higher given a limited time.

13.2 Human Access Control & ID Authentication

Use Case

Many companies world-wide already use RFID technology for employee access control systems. The access control system can fulfill purposes such as limiting access to a restricted area and capturing entry and exit time information for wages calculation. In addition to access control at the door, the security guard would use a handheld to read the RFID name badge of the person and check the name. This checking is useful for spot check anywhere in the building premise (not necessarily the door) and also in places where door is not available.

Current Approach

High Frequency technology is adopted in many access control systems. The read range of HF is short such that presenting of access card in front of the read point is required. This process can force the security guard to walk up to the person and ask him/her to proffer the badge, which is obviously very inconvenient and slow down or even disrupt traffic.

Suggested Approach

For access control system with high traffic of access, UHF has advantage over HF because the employees do not have to present the access card to the read point one by one, instead they can just walk by the read point and the access card can be read. In the case of security guard using a handheld Reader, a long read range one is particular useful because it can be used to read a person from farther away, thus avoiding disruption of traffic.

Recommendation

The CSL CS101-2 reader is powered by CSL technology with extremely long read range. This ensures the authentication process is fast.

13.3 Dock Door Inventory

Use Case

Everyday millions of pallets pass through all kinds of logistics nodes, such as door of container, dock door of distribution center, etc. On the pallets there may be 40 or more boxes. To be able to inventory these boxes will provide visibility of each and every box of goods throughout the logistics trail. The benefit of that is now so well documented that it does not need any more explanation.

Current Approach

Fixed reader with gateway or portal mounting platform is used. The problem with this is sometimes not all boxes can be read. When that occurs, the operator at the dock door has to roll back and forth the pallet to get a better read.

Suggested Approach

A handheld RFID read should be used to complement the fixed reader portal so that if there are certain tags missing, the handheld RFID reader can be used to point and go near to the pallet and read those tags.

Recommendation

Powered by CSL technology, the CSL CS101-2 reader has extremely high read range and read rate, so that the inventory process at the fast moving logistics nodes can be made much more efficient.

13.4 Work-In-Progress Monitoring & Inventory

Use Case

The manufacturing process in factory can be long and complicated. Once the raw materials are sent into the manufacturing plant, they remain invisible until emerging as a finished product. Better visibility of work-in-progress is needed for production decision-making. This is particularly important for industry where the overall production time of a unit is long – weeks or months where the unit will be moving along the production line. One example is the knitted clothing industry, where WIP units are often lost and forgotten in a heap within a certain part of the overall production line.

Current Approach

Tracking of manufacturing process is not automated. Status of parts and work-in-progress are out-dated, distributed and manually collected.

Suggested Approach

The introduction of RFID technology to the manufacturing process in factory can improve the visibility of the work-in-progress. Parts and subassemblies within the manufacturing plant are tracked precisely such that more accurate part level and work-in-progress records are available. Moreover, automatic monitoring of work-in-progress status on semi-finished assemblies throughout the production cycle can reduce downtime and ensure on-time delivery. Combining RFID reader with output device can also help in decision making. For example, alarm is triggered when semi-finished items or batches are routed to the wrong manufacturing cell. For initial deployment or low cost deployment, the handheld reader can be used to do inventory of WIP units on the production line, with the production line also tagged by RFID tag so that the WIP units and the production line are both read and reported to the server so that the server can then determine the status and location of the WIP unit.

Recommendation

As powered by the advance and intelligent technology from CSL, the CSL CS-101 reader has long read range and high read rate, and it can enable multiple inventory rounds throughout the day (if needed) to mark the location of the WIP units relative to the production line positions. In addition, its long read range will enable quick search of missing units that may be hidden under a heap of units.

13.5 Vehicle Tracking in Maintenance Depot

Use Case

In maintenance depot, vehicles arrive for maintenance and checking. If the activities of vehicles inside the maintenance depot can be tracked, better arrangement of vehicles maintenance can be achieved.

Current Approach

Vehicle maintenance is tracked manually. Human errors may occur such as omitting particular maintenance checking on a vehicle.

Suggested Approach

RFID technology can be applied to track vehicles' activities inside the depot. Once a vehicle is tagged, it's movement can be recorded anywhere in the RFID enabled depot. The process is completely automatic in the sense that the vehicle does not have to stop for being recorded. Moreover, no staff is involved in the process and thus human errors can be eliminated. The vehicles' movement record gives accurate maintenance checking and repairing history which is important for vehicle management such as identifying obsolete parts.

Recommendation

One of the challenges in tracking vehicles in maintenance depot is that high tag resolution is required. Cross reading of tags by different entry points would affect the accuracy of identifying the vehicles in the lane. This problem can be overcome by shielding the capture points such that each capture point would only read tags that are corresponding to it. Furthermore, the CSL CS101-2 reader allows filtering of tags by both RF Signal Strength Indicator (RSSI) and read count to prevent cross reading of tags by read points in multiple lanes.

13.6 Vehicle Information System

Use Case

In many countries, the possibility of using an RFID tag as a license plate is very welcome because that enables a host of analysis, tracking and law enforcement operations.

Current Approach

Vehicle license has traditionally been tracked visually or optically.

Suggested Approach

RFID technology can be applied to the label on the windshield, or to a stand on the dashboard, or to the inside of the Taxi light box on top of a taxi, or even directly onto the front and back license plate. The police or the traffic inspector can point the handheld reader and grab the ID and then check the database for any abnormal status (such as stolen car, car with owner involved in a felony, etc.)

Recommendation

The CS101-2 has long read range so that it allows the policeman to read the license plate at a long distance away. This makes the process a lot safer for the policeman, needless to say.

13.7 Document Inventory & Search

Use Case

In some organizations, costs associated with tracking documents are high. An automatic document management system is especially beneficial in those environments where the documents are of high value to the organization, and the loss of a document would have significant negative impact. Examples include hospitals, lawyer's offices, libraries and government departments.

Current Approach

Documents are tracked and managed manually. Human error may lead to lost of documents. Moreover, time spent in searching for document is long, especially when documents are not systematically well organized.

Suggested Approach

RFID technology has made a dramatic improvement in tracking and managing documents. By tagging the documents and equipping read points for checking in and out, status and location of documents can be traced easily. Other usages such as inventory checking and locating lost documents can also be achieved.

Recommendation

CS101-2, by virtue of its long read range and high read rate, will make the inventory process so much faster that the productivity of the concerned office will be dramatically improved.

Appendix A. RFID Basics

Passive tag RFID technology involves the reader, the antenna and the tag.

The reader sends out energy in the relevant frequency band to the antenna via RF cables, and the antenna radiates the energy out. This energy impinges on an RFID tag.

The RFID tag consists of an antenna coupled to an RFID IC. This IC converts the AC voltage it receives at the antenna port to DC voltage that in turn is used to empower the digital circuit inside.

The digital circuit then turns on and off some components connected to the antenna port, thereby changing its scattering behavior, in a pre-designed clock rate.

This changing of antenna port parameters then causes a “modulation” of the back-scattered RF energy.

This modulated back-scattered energy is detected by the reader and the modulation is captured and analyzed.

Appendix B. Glossary

Air interface
The complete communication link between an Interrogator and a Tag including the physical layer, collision arbitration algorithm, command and response structure, and data-coding methodology.

Autonomous time trigger
Each tag will only be reported once within a duplicate elimination time. See also duplicate elimination time.

Batch alert to server
Collected tag information are sent to server at the end of each duplicate elimination cycle (Time Window)

Capture point
Unique name corresponding to each of the four antennas

Command set
The set of commands used to explore and modify a Tag population.

Continuous wave
Typically a sinusoid at a given frequency, but more generally any Interrogator waveform suitable for powering a passive Tag without amplitude and/or phase modulation of sufficient magnitude to be interpreted by a Tag as transmitted data.

Cover-coding
A method by which an Interrogator obscures information that it is transmitting to a Tag. To cover-code data or a password, an Interrogator first requests a random number from the Tag. The Interrogator then performs a bit-wise EXOR of the data or password with this random number, and transmits the cover-coded (also called
ciphertext) string to the Tag. The Tag uncovers the data or password by performing a bit-wise EXOR of the received cover-coded string with the original random number.

Dense-Interrogator environment
An operating environment (defined below) within which the number of simultaneously active

Interrogators is large relative to the number of available channels (for example, 50 active Interrogators operating in 50 available channels).

Duplicate elimination time

Time span of a duplicate elimination cycle, within which duplicate tags will be removed.

Duplicate Elimination Triggering Method

The method used to trigger inventory with duplicate elimination. See also autonomous time trigger and polling trigger by client.

Estimated tag time in field

An estimation of how long a tag will remain within the read zone of antenna

Event

An event defines action to be performed for a specific triggering logic. See also inventory enabling trigger, trigger, inventory disabling trigger, and resultant action.

Extended temperature range

–40 °C to +65 °C (see nominal temperature range).

Full-duplex communications

A communications channel that carries data in both directions at once. See also half-duplex communications.

Half-duplex communications

A communications channel that carries data in one direction at a time rather than in both directions at once. See also full-duplex communications.

Instant alert to server

Collected tag information are sent to server immediately as it is read

Inventoried flag

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

Inventory enabling trigger

The initial trigger that turns on the RF power of the reader to start doing inventory

Inventory Enabling Cycle
Time between an inventory enabling trigger and inventory disabling trigger.

Inventory disabling trigger
The trigger that turns off the RF power of the reader to stop doing inventory

Inventory round
The period between successive Query commands.

Inventory Search Mode
Method of reading tags by antenna. See also Single Target Large Population Inventory.

Modulation Profile
Way of transmitting information between tags and reader.

Multiple-Interrogator environment
An operating environment (defined below) within which the number of simultaneously active Interrogators is modest relative to the number of available channels (for example, 10 active Interrogators operating in 50 available channels).

Network failure data backlog
Tag data buffered in reader memory during network failure. Buffered tags are sent to trusted server when network is restored.

Nominal temperature range
–25 °C to +40 °C (see extended temperature range).

Operating environment
A region within which an Interrogator's RF transmissions are attenuated by less than 90dB. In free space, the operating environment is a sphere whose radius is approximately 1000m, with the Interrogator located at the © 2004, EPCglobal Inc. Page 13 of 94 31 January 2005 center. In a building or other enclosure, the size and shape of the operating environment depends on factors such as the material properties and shape of the building, and may be less than 1000m in certain directions and greater than 1000m in other directions.

Operating procedure

Collectively, the set of functions and commands used by an Interrogator to identify and modify Tags. (Also known as the Tag-identification layer.)

Passive Tag (or passive Label)
A Tag (or Label) whose transceiver is powered by the RF field.

Permalock or Permalocked
A memory location whose lock status is unchangeable (i.e. the memory location is permanently locked or permanently unlocked) is said to be permalocked.

Persistent memory or persistent flag
A memory or flag value whose state is maintained during a brief loss of Tag power.

Physical layer
The data coding and modulation waveforms used in Interrogator-to-Tag and Tag-to-Interrogator signaling.

Polling Trigger by Client
Tags read are buffered in reader until client application polls the read result. A tag will only be reported once in each polling trigger.

Protocol
Collectively, a physical layer and a Tag-identification layer specification.

Q
A parameter that an Interrogator uses to regulate the probability of Tag response. An Interrogator commands Tags in an inventory round to load a Q-bit random (or pseudo-random) number into their slot counter; the Interrogator may also command Tags to decrement their slot counter. Tags reply when the value in their slot counter (i.e. their slot – see below) is zero. Q is an integer in the range (0,15); the corresponding Tagresponse probabilities range from 20 = 1 to 2–15 = 0.000031.

Resultant Action
Resultant action that will be enforced when an event logic is established

Single Target Large Population Inventory
A mode for reading a large number of tags at a time accurately. When this mode is used, tags that are read already will not respond to the reader for a short period of time. This can avoid the

strong tags from dominating the week ones.

Session

An inventory process comprising an Interrogator and an associated Tag population. An Interrogator chooses one of four sessions and inventories Tags within that session. The Interrogator and associated Tag population operate in one and only one session for the duration of an inventory round (defined above). 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.

Single-Interrogator environment

An operating environment (defined above) within which there is a single active Interrogator at any given time.

Singulation

Identifying an individual Tag in a multiple-Tag environment.

Slot

Slot corresponds to the point in an inventory round at which a Tag may respond. Slot is the value output by a Tag's slot counter; Tags reply when their slot (i.e. the value in their slot counter) is zero. See also Q (above).

Slotted random anticollision

An anticollision algorithm where Tags load a random (or pseudo-random) number into a slot counter, decrement this slot counter based on Interrogator commands, and reply to the Interrogator when their slot counter reaches zero.

Tag-identification layer

Collectively, the set of functions and commands used by an Interrogator to identify and modify Tags (also known as the operating procedure).

Tari

Reference time interval for a data-0 in Interrogator-to-Tag signaling. The mnemonic "Tari" derives from the ISO/IEC 18000-6 (part A) specification, in which Tari is an abbreviation for Type A Reference Interval.

Trigger

A stimulus that causes the reader to recognize it and do something about it.

Trusted Server

Server for automatic data submission by the reader using the event engine.

Appendix C. Federal Communication Commissions Compliance

This equipment has been tested and found to comply with the limits for a class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna
- Increase the separation between the equipment and receiver
- Consult the dealer or an qualified radio/TV technician for assistance

FCC NOTICE: To comply with FCC part 15 rules in the United States, the system must be professionally installed to ensure compliance with the Part 15 certification. It is the responsibility of the operator and professional installer to ensure that only certified systems are deployed in the United States. The use of the system in any other combination (such as co-located antennas transmitting the same information) is expressly forbidden.

Note:

Changes or modifications not expressly approved by the party responsible for compliance could void the user’s authority to operate the equipment.

Appendix D. Maximum Permissible Exposure

Maximum Permissible Exposure Requirement: Section 47 CFR §1.1307

Three different categories of transmitters are defined by the FCC in OET Bulletin 65. These categories are fixed installation, mobile, and portable and are defined as follows:

- **Fixed Installations:** fixed location means that the device, including its antenna, is physically secured at a permanent location and is not able to be easily moved to another location. Additionally, distance to humans from the antenna is maintained to at least 2 meters.

- **Mobile Devices:** a mobile device is defined as a transmitting device designed to be used in other than fixed locations and to be generally used in such a way that a separation distance of at least 20 centimeters is normally maintained between the transmitter's radiating structures and the body of the user or nearby persons. Transmitters designed to be used by consumers or workers that can be easily re-located, such as a wireless modem operating in a laptop computer, are considered mobile devices if they meet the 20 centimeter separation requirement. The FCC rules for evaluating mobile devices for RF compliance are found in 47 CFR §2.1091.

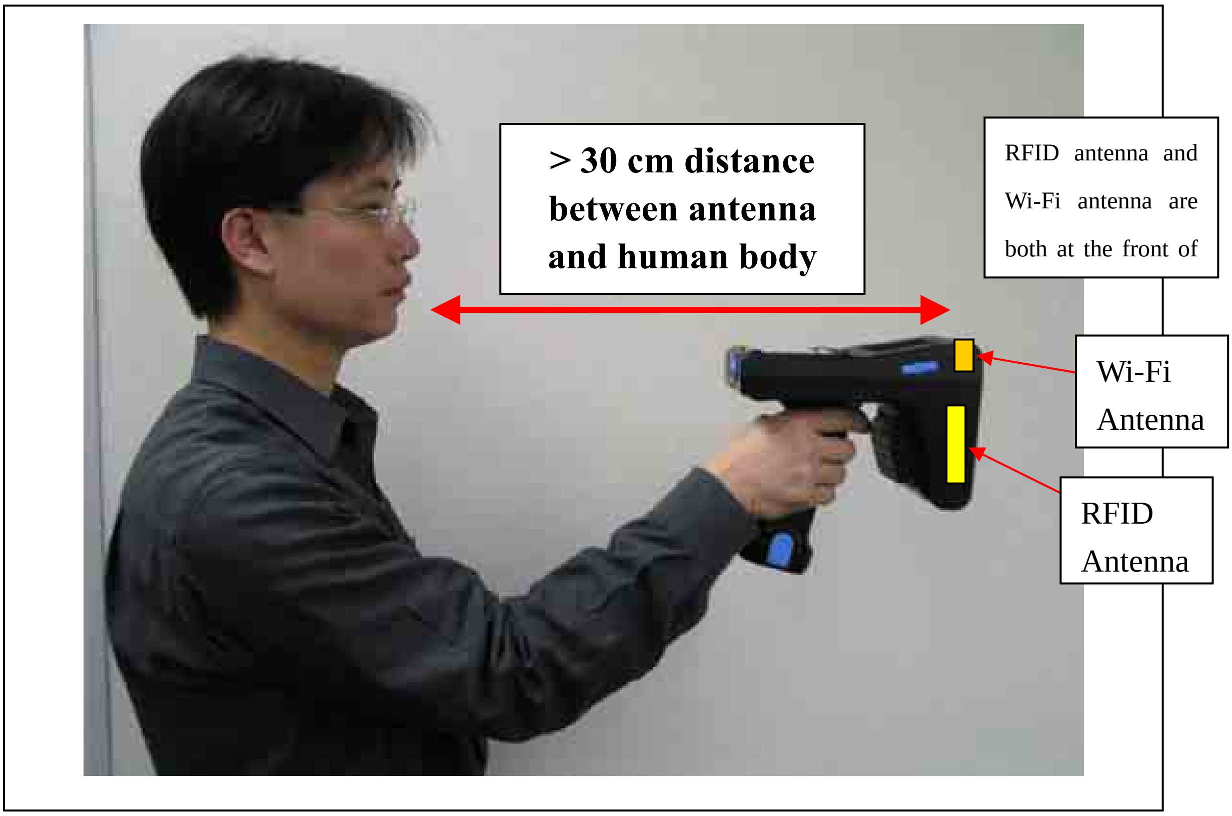
- **Portable Devices:** a portable device is defined as a transmitting device designed to be used so that the radiating structure(s) of the device is/are within 20 centimeters of the body of the user. Portable device requirements are found in Section 2.1093 of the FCC's Rules (47 CFR§2.1093).

The FCC also categorizes the use of the device as based upon the user’s awareness and ability to exercise control over his or her exposure. The two categories defined are Occupational/ Controlled Exposure and General Population/Uncontrolled Exposure. These two categories are defined as follows:

- **Occupational/Controlled Exposure:** In general, occupational/controlled exposure limits are applicable to situations in which persons are exposed as a consequence of their employment, who have been made fully aware of the potential for exposure and can exercise control over their exposure. This exposure category is also applicable when the exposure is of a transient nature due to incidental passage through a location where the exposure levels may be higher than the general population/uncontrolled limits, but the exposed person is fully aware of the potential for exposure and can exercise control over his or her exposure by leaving the area or by some other appropriate means. Awareness of the potential for RF exposure in a workplace or similar environment can be provided through specific training as part of a RF safety program. If appropriate, warning signs and labels can also be used to establish such awareness by providing prominent information on the risk of potential exposure and instructions on methods to minimize such exposure risks.

- **General Population/Uncontrolled Exposure:** The general population / uncontrolled exposure limits are applicable to situations in which the general public may be exposed or in which persons who are exposed as a consequence of their employment may not be made fully aware of the potential for exposure or cannot exercise control over their exposure. Members of the general public would come under this category when exposure is not employment-related; for example, in the case of a wireless transmitter that exposes persons in its vicinity. Warning labels placed on low-power consumer devices such as cellular telephones are not considered sufficient to allow the device to be considered under the occupational/controlled category, and the general population/uncontrolled exposure limits apply to these devices.

The CS-101-2 RFID reader is a handheld reader that is used in a handheld operation manner:



The user takes the handheld reader and moves around the work space and read tags. Since it is not used in a fixed place, it falls into the category of mobile or portable devices. Since the antenna is actually > 30 cm away from the user body torso, it can be categorized as mobile devices. Since the distance between antenna and body is generally > 30 cm, the **simplified method of power density compliance is used in this report to show CS101-2 complies with FCC MPE limit of General Population / Uncontrolled Exposure.**

Radio Frequency Radiation Exposure Evaluation – RFID Mode:

The measured highest RF output power of the EUT feeding to the embedded antenna was 28.6dBm at 927.25MHz. According to §1.1310 of the FCC rules, the power density limit for **General Population/Uncontrolled Exposure** at 927.25 MHz is $f_{(MHz)}/1500$ = 0.6182mW/cm^2. The maximum permissible exposure (MPE) is calculated to show the required separation distance that must be maintained during installation to maintain compliance with the power density limit.

The following formula was used to calculate the Power Density:

$$S = \frac{PG}{4\pi R^2}$$

where:

S = Power density

P = Power feeding to the embedded patch antenna

G = Tx gain of the antenna (linear gain)

R = Distance from the antenna

For the EUT, the calculation is as follows:

P = 28.6dBm = 724.4mW

G = Maximum Antenna Gain = 5.5dBi = anti-log(5.5/10) = 3.55

At 20cm separation,

$$S = \frac{724.4 \times 3.55}{4\pi(20)^2} = 0.5116\text{mW/cm}^2$$

Based on the above calculation for 20cm separation, the power density does not exceed FCC limit of 0.6182mW/cm^2.

Radio Frequency Radiation Exposure Evaluation – WiFi Mode:

The measured highest RF output power of the EUT feeding to the embedded antenna was 11.5dBm at 2412MHz. According to §1.1310 of the FCC rules, the power density limit for **General Population/Uncontrolled Exposure** at 2412MHz is = 1.0 mW/cm^2. The maximum permissible exposure (MPE) is calculated to show the required separation distance that must be maintained during installation to maintain compliance with the power density limit.

The following formula was used to calculate the Power Density:

$$S = \frac{PG}{4\pi R^2}$$

where:

S = Power density

P = Power feeding to the embedded patch antenna

G = Tx gain of antenna (linear gain)

R = Distance from the antenna

For the EUT, the calculation is as follows:

P = 11.5dBm = 14.13mW

G = Maximum Antenna Gain = 2.0dBi = anti-log(2.0/10) = 1.585

At 20cm separation,

$$S = \frac{14.13 \times 1.585}{4\pi(20)^2} = 0.004456 \text{mW/cm}^2$$

Based on the above calculation for 20cm separation, the power density does not exceed FCC limit of 1.0mW/cm^2.