

UHF GUN RFID SDK Version 1.0.1 Manual for Microsoft® .NET CF and C++

Release date: May, 2014

©2013 M3 MOBILE Co., Ltd. All Rights Reserved.

Copyright and Agreement

WARNING: All contents of this SDK manual are protected by the copyright laws and all rights are reserved. Unauthorized distribution or copying is strictly prohibited.

M3 Mobile does not guarantee the quality and performance of the programs written in unsupported programming language. For supported development tools and languages, please refer to Development Tool and Requirements section.

Development Tools and Requirements

Supported Development Tools

- Visual Studio 2005/2008 – Visual C++, Visual C#, Visual Basic .NET

Development System Requirements

- Pentium – 1 Gigahertz (GHz) processor or higher
- Microsoft Windows 98 / ME / 2000 / XP / 2003 / 7
- ActiveSync

Development Platform Requirements

- “M3 Plus Platform SDK” and “Windows Mobile 5.0 SDK for Pocket PC” must be installed on your computer to be able to develop software using this SDK.
- M3Plus Platform SDK : [DOWNLOAD](#)
- Windows Mobile 5.0 SDK for Pocket PC : [LINK](#)

Supported PDA

- M3 ORANGE+ WM
- M3 ORANGE+ CE
- M3 ORANGE S

Table of Contents

1	SDK Manual for UGR .NET	4
1.1	Introduction	4
1.2	Tutorial	4
1.2.1	Init UHF	5
1.2.2	Inventory Start.....	6
1.2.3	Get Data.....	6
1.2.4	Inventory Stop.....	6
1.2.5	Close UHF.....	6
1.3	UHF GUN READER (WM/CE).....	8
1.4	Functions for UHF GUN READER	13
1.4.1	GetError	14
1.4.2	IsReady.....	14
1.4.3	Init.....	15
1.4.4	Close.....	16
1.4.5	Inventory	17
1.4.6	InventoryStop.....	17
1.4.7	Read	18
1.4.8	Write.....	19
1.4.9	Lock	20
1.4.10	Kill	22
1.4.11	GetData.....	23
1.4.12	SetRegionFrequency.....	23
1.4.13	GetRegionalFrequency	24
1.4.14	ReadBattery	25
1.4.15	ReadBatteryStatus	26
1.4.16	Version.....	26
2	SDK Manual for UGR C++	28
2.1	Introduction	28
2.2	Tutorial	28
2.2.1	Init UHF	29
2.2.2	Start Inventory.....	30

2.2.3	Get Data.....	30
2.2.4	Stop Inventory.....	30
2.2.5	Close UHF.....	30
2.3	UHF GUN READER (WM/CE).....	31
2.4	Functions for UHF GUN READER	35
2.4.1	UHF_GetError	36
2.4.2	UHF_IsReady.....	37
2.4.3	UHF_Init	38
2.4.4	UHF_Close	38
2.4.5	UHF_Inventory	39
2.4.6	UHF_InventoryStop	40
2.4.7	UHF_Read	40
2.4.8	UHF_Write.....	42
2.4.9	UHF_Lock.....	43
2.4.10	UHF_Kill.....	45
2.4.11	UHF_GetData.....	46
2.4.12	UHF_SetRegionFrequency	47
2.4.13	UHF_GetRegionFrequency.....	48
2.4.14	UHF_ReadBattery	48
2.4.15	UHF_ReadBatteryStatus	49
2.4.16	UHF_Version	50
3	Demo Manual.....	52
3.1	UHF_GunTest.exe	52
3.1.1	First view of the program	52
3.1.2	Access Menu	52
3.1.3	Config Menu	53

1 SDK Manual for UGR .NET

1.1 Introduction

This document is a reference guide for the software developer's kit (SDK) for UHF Gun Reader.

Module Name	Description			
UHF GUN READER	Insert proper PDA ¹ to read UHF tags.			
	Demo Application	Sample Source	Version	Date
	UHF_Gun_Net_Test.exe	UHF_Gun_Net_Test	1.0.1	2014-04-24

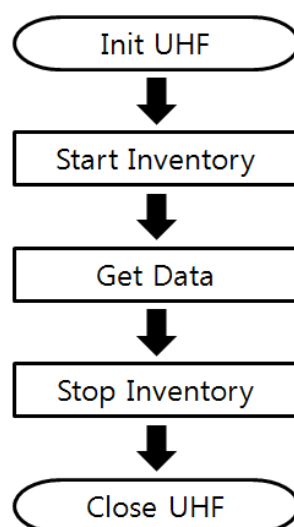
PDA¹ : M3 ORANGE+ WM / M3 ORANGE+ CE / M3 ORANGE S

1.2 Tutorial

This chapter describes the basic usage of M3 Mobile SDK functions in a step-by-step manner. In this tutorial section, the following topics are treated.

Section	Topic
UHF GUN READER	Init UHF Start Inventory Get Data Stop Inventory Close UHF

Basic UHF GUN READER flowchart is shown below.



< UHF RFID flow chart >

Tip!!

- i. It can be used with both Windows Mobile and Windows CE platforms.
- ii. Initialize will return fail in the following conditions:
 - PDA detached from gun reader
 - RFID / SCAN switch is at SCAN position
 - Gun reader's battery is detached or empty
 - PDA in gun reader is synced with PC
- iii. Power status can be obtained using delegate ReceivedPower. When turning off, UHF RFID must be closed. When turning on, UHF RFID must be initialized.
- iv. UHF RFID uses the same virtual key as scanner; VK_H14 for WM and VK_F22 for CE.

1.2.1 Init UHF

```
UHFNet.MessageClass.PowerFunc += new ReceivedPower(OnReceivedPower);
```

```
bool Open()
{
    RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();
    String strstatus = null;

    status = UHFNet.Init();

    if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)
    {
        strstatus = String.Format("RFID Init Error-{0:G}", status);

        return false;
    }
    label_Result.Text = "Open UHF";
    return true;
}
```

```
public void OnReceivedPower(bool a_bPowerOn)
{
    if (a_bPowerOn == true)
    {
        if (m_bOpen == false)
        {
            if (Open())
            {
                m_bOpen = true;
            }
        }
    }
    else
    {
        if (m_bOpen == true)
        {
            Close();
            m_bOpen = false;
        }
    }
}
```

```
}  
}
```

1.2.2 Inventory Start

```
void Inventory()  
{  
    UHFNet.Inventory();  
}
```

1.2.3 Get Data

```
UHFNet.MessageClass.InventoryFunc += new ReceivedInventory(OnReceivedInventory);
```

```
private void OnReceivedInventory()  
{  
    int nTaglenth = 0;  
    String strTagData = null;  
    StringBuilder strData = new StringBuilder(260);  
  
    nTaglenth = UHFNet.GetData(strData);  
  
    if (checkBox_CRC.Checked == true)  
    {  
        strTagData = strData.ToString().Substring(0, nTaglenth);  
    }  
    else if (checkBox_CRC.Checked == false)  
    {  
        strTagData = strData.ToString().Substring(0, nTaglenth - 2);  
    }  
}
```

1.2.4 Inventory Stop

```
void InventoryStop()  
{  
    UHFNet.InventoryStop();  
}
```

1.2.5 Close UHF

```
bool Close()  
{  
    RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();  
    String strstatus = null;  
  
    status = UHFNet.Close();  
    if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)  
    {  
        strstatus = String.Format("{0:G}", status);  
  
        return false;  
    }  
    label_Result.Text = "Close UHF";  
}
```

```
    return true;  
}
```

1.3 UHF GUN READER (WM/CE)

Event
<pre> public delegate void ReceivedInventory(); public delegate void ReceivedPower(bool a_bPowerOn); public delegate void ReceivedMemoryData(); </pre>
Enum
<pre> public enum BATTERY_STATUS { BATTERY_ERROR = 0, BATTERY_0 = 1, BATTERY_1 = 2, BATTERY_2 = 3, BATTERY_3 = 4, BATTERY_FULL = 5, } public enum RFIDErrorcode { TAG_OTHERERROR = 0x0, TAG_SUCCESS = 0x1, TAG_MEMORYOVERRUN = 0x3, TAG_MEMORYLOCKED = 0x4, TAG_INSUFFICIENTPOWER = 0xB, TAG_NONSPECIFICERROR = 0xF, MAC_NOERROR = 0x10, MAC_HANDLEMISSMATCH = 0x11, MAC_CRCERROR = 0x12, MAC_NOTAGREPLY = 0x13, MAC_INVALIDPASSWD = 0x14, MAC_ZEROKILLPASSWD = 0x15, MAC_TAGLOST = 0x16, MAC_COMMANDFORMATERROR = 0x17, MAC_READCOUNTINVALID = 0x18, MAC_OUTOFRETRIES = 0x19 }; public enum RFIDTagBank { TAG_RESERVED = 0, TAG_EPC = 1, </pre>


```

    TAG_TID = 2,
    TAG_USER = 3
};

public enum RFIDRegion
{
    RFID_REGION_KOREA_NEW = 0,
    RFID_REGION_KOREA_WEAK = 1,
    RFID_REGION_KOREA_OLD = 2,
    RFID_REGION_USA = 3,
    RFID_REGION_EURO = 4,
    RFID_REGION_EURO_NEW = 5,
    RFID_REGION_JAPAN = 6,
    RFID_REGION_CHINA = 7,
    RFID_REGION_AUSTRALIA = 8,
    RFID_REGION_BRAZIL = 9,
    RFID_REGION_MALAYSIA = 10,
    RFID_REGION_TAIWAN = 11
};

public enum RFIDLockPermission
{
    PERMISSION_ACCESSIBLE = 0,
    PERMISSION_ALWAYS_ACCESSIBLE = 1,
    PERMISSION_SECURED_ACCESSIBLE = 2,
    PERMISSION_ALWAYS_NOT_ACCESSIBLE = 3,
    PERMISSION_NO_CHANGE = 4
};

public enum RFID_STATUS
{
    /* Success */
    RFID_STATUS_OK,
    /* Attempted to open a radio that is already open */
    RFID_ERROR_ALREADY_OPEN = -9999,
    /* Buffer supplied is too small */
    RFID_ERROR_BUFFER_TOO_SMALL,
    /* General failure */
    RFID_ERROR_FAILURE,
    /* Failed to load radio bus driver */
    RFID_ERROR_DRIVER_LOAD,
    /* Library cannot use version of radio bus driver present on system */
    RFID_ERROR_DRIVER_MISMATCH,

```

```

/* This error code is no longer used, maintain slot in enum in case
   * anyone is using hard-coded error codes for some reason */
RFID_ERROR_RESERVED_01, /* -9994 */

/* Antenna number is invalid */
RFID_ERROR_INVALID_ANTENNA, /* -9993 */

/* Radio handle provided is invalid */
RFID_ERROR_INVALID_HANDLE, /* -9992 */

/* One of the parameters to the function is invalid */
RFID_ERROR_INVALID_PARAMETER, /* -9991 */

/* Attempted to open a non-existent radio */
RFID_ERROR_NO_SUCH_RADIO, /* -9990 */

/* Library has not been successfully initialized */
RFID_ERROR_NOT_INITIALIZED, /* -9989 */

/* Function not supported */
RFID_ERROR_NOT_SUPPORTED, /* -9988 */

/* Operation was cancelled by call to cancel operation, close radio, or
   * shut down the library */
RFID_ERROR_OPERATION_CANCELLED, /* -9987 */

/* Library encountered an error allocating memory */
RFID_ERROR_OUT_OF_MEMORY, /* -9986 */

/* The operation cannot be performed because the radio is currently busy */
RFID_ERROR_RADIO_BUSY, /* -9985 */

/* The underlying radio module encountered an error */
RFID_ERROR_RADIO_FAILURE, /* -9984 */

/* The radio has been detached from the system */
RFID_ERROR_RADIO_NOT_PRESENT, /* -9983 */

/* The RFID library function is not allowed at this time. */
RFID_ERROR_CURRENTLY_NOT_ALLOWED, /* -9982 */

/* The radio module's MAC firmware is not responding to requests. */
RFID_ERROR_RADIO_NOT_RESPONDING, /* -9981 */

/* The MAC firmware encountered an error while initiating the nonvolatile */
/* memory update. The MAC firmware will return to its normal idle state */
/* without resetting the radio module. */
RFID_ERROR_NONVOLATILE_INIT_FAILED, /* -9980 */

/* An attempt was made to write data to an address that is not in the */
/* valid range of radio module nonvolatile memory addresses. */
RFID_ERROR_NONVOLATILE_OUT_OF_BOUNDS, /* -9979 */

/* The MAC firmware encountered an error while trying to write to the */
/* radio module's nonvolatile memory region. */
RFID_ERROR_NONVOLATILE_WRITE_FAILED, /* -9978 */

/* The underlying transport layer detected that there was an overflow */
/* error resulting in one or more bytes of the incoming data being */
/* dropped. The operation was aborted and all data in the pipeline was */

```

```

    /* flushed. */
    RFID_ERROR_RECEIVE_OVERFLOW, /* -9977 */
    /* An unexpected value was returned to this function by the MAC firmware */
    RFID_ERROR_UNEXPECTED_VALUE, /* -9976 */
    /* The MAC firmware encountered CRC errors while trying to
    /* write to the radio module's nonvolatile memory region. */
    RFID_ERROR_NONVOLATILE_CRC_FAILED, /* -9975 */
    /* The MAC firmware encountered unexpected values in the packet header */
    RFID_ERROR_NONVOLATILE_PACKET_HEADER, /* -9974 */
    /* The MAC firmware received more than the specified maximum packet size */
    RFID_ERROR_NONVOLATILE_MAX_PACKET_LENGTH /* -9973 */
};

```

Structure

```

public struct RFID_VERSION
{
    /* The major version (i.e., in 1.x.x.x, the 1) */
    public INT32U major;
    /* The minor version (i.e., in x.1.x.x, the 1) */
    public INT32U minor;
    /* The maintenance number (i.e., in x.x.1.x, the 1) */
    public INT32U maintenance;
    /* The release number (i.e., in x.x.x.1, the 1) */
    public INT32U release;
};

public struct RFIDReadCmd
{
    internal HWND hWnd; //Diag handle Receive for Message
    public RFIDTagBank bank;
    public INT8U wlength;
    public INT8U offset;
    public INT32U accpwd;
};

public struct RFIDWriteCmd
{
    internal HWND hWnd; //Diag handle Receive for Message
    public RFIDTagBank bank;
    public INT8U wlength;
    public INT8U offset;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 32)]

```

```
    public INT16U[] wdata;
    public INT32U accpwd;
};

public struct RFIDLockCmd
{
    internal HWND hWnd; //Diag handle Receive for Message
    public RFIDLockPermission lockkillpwd;
    public RFIDLockPermission lockaccesspwd;
    public RFIDLockPermission lockepc;
    public RFIDLockPermission locktid;
    public RFIDLockPermission lockuser;
    public INT32U accpwd;
};

public struct RFIDKillCmd
{
    internal HWND hWnd; //Diag handle Receive for Message
    public INT32U killpwd;
};
```

1.4 Functions for UHF GUN READER

Name	Description
GetError	Check the last error
IsReady	Check UHF GUN READER's availability by checking the power
Init	Initialize RFID
Close	Close RFID
Inventory	Start Inventory by reading EPC data
InventoryStop	Stop Inventory
Read	Read memory from tag
Write	Write data to tag
Lock	Limit access to tag
Kill	Kill the tag (disabling the tag)
GetData	Get tag data from Inventory and Read
SetRegionFrequency	Set RFID frequency to suit regional regulation
GetRegionFrequency	Check current regional frequency setting
ReadBattery	Check battery voltage and ADC value
ReadBatteryStatus	Check battery level (0~4 level)
Version	Check DLL and F/W version

1.4.1 GetError

Description

Check the last error

Syntax

```
RFIDUHF.RFIDErrorCode GetError();
```

Parameters

None

Return Value

Retuens RFIDErrorCode of enum type

Remarks

None

See Also

None

For C++

Library : RFID_UHF.lib

Function : RFIDErrorCode UHF_GetError();

Example

```
int nLength = 0;
StringBuilder strData = new StringBuilder(260);
RFIDUHF.RFIDErrorCode error;
nLength = _UHFNet.GetData(strData);
if (nLength == 0)
{
    error = _UHFNet.GetError();
    MessageBox.Show("OnReceivedData: " + error.ToString());
}
```

1.4.2 IsReady

Description

Check UHF GUN READER's availability by checking the power

Syntax

```
bool IsReady();
```

Parameters

None

Return Value

TRUE = Success

FALSE = Fail

Remarks

Following cases will return false:

1. PDA detached from gun reader
2. RFID / SCAN switch is at SCAN position
3. Gun reader's battery is detached or empty
4. PDA in gun reader is synced with PC

See Also

None

For C++

Library : RFID_UHF.lib

Function : BOOL UHF_IsReady();

Example

```
if(!_UHFNet.IsReady())  
    return;
```

1.4.3 Init

Description

Initialize RFID

Syntax

```
RFIDUHF.RFID_STATUS Init();
```

Parameters

None

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

Close

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Init (HWND hDlgWnd);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();  
  
String strstatus = null;  
  
status = UHFNet.Init();  
  
if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)  
{  
    strstatus = String.Format("RFID Init Error-{0:G}", status);  
}
```

1.4.4 Close

Description

Close RFID

Syntax

RFIDUHF.RFID_STATUS Close();

Parameters

None

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

Init

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Close();

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();  
  
String strstatus = null;
```



```
status = UHFNet.Close();  
if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)  
{  
    strstatus = String.Format("{0:G}", status);  
}
```

1.4.5 Inventory

Description

Start Inventory by reading EPC data

Syntax

```
void Inventory();
```

Parameters

None

Return Value

None

Remarks

When tag is detected after starting inventory, an event is logged through delegate void ReceivedInventory()

See Also

InventoryStop

For C++

Library : RFID_UHF.lib

Function : void UHF_Inventory(HWND hWnd);

Example

```
UHFNet.Inventory();
```

1.4.6 InventoryStop

Description

Stop Inventory

Syntax

```
void InventoryStop();
```

Parameters

None

Return Value

None

Remarks

None

See Also

Inventory

For C++

Library : RFID_UHF.lib

Function : void UHF_InventoryStop();

Example

```
UHFNet.InventoryStop();
```

1.4.7 Read

Description

Read memory from tag

Syntax

```
RFIDUHF.RFID_STATUS Read(ref RFIDUHF.RFIDReadCmd cmd);
```

Parameters

cmd

RFIDReadCmd structure is used to assign tag memory location

Return Value

Returns RFID_STATUS of enum type

Remarks

When tag is detected using read, an event is logged through delegate void ReceivedMemoryData(). Then, the data can be obtained using GetData.

See Also

GetData

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Read(RFIDReadCmd *cmd);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();  
RFIDUHF.RFIDReadCmd ReadCmd = new RFIDUHF.RFIDReadCmd();  
  
ReadCmd.bank = RFIDUHF.RFIDTagBank.TAG_EPC;  
ReadCmd.wlength = 6;  
ReadCmd.offset = 2;  
ReadCmd.accpwd = Convert.ToUInt16(textBox_RWPwd.Text, 16); // 00000000  
  
status = _UHFNet.Read(ref ReadCmd);  
  
if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)  
{  
    MessageBox.Show("RFID Read Fail!");  
}
```

1.4.8 Write

Description

Write data to tag

Syntax

```
RFIDUHF.RFID_STATUS Write(ref RFIDUHF.RFIDWriteCmd cmd);
```

Parameters

cmd

RFIDWriteCmd structure is used to assign tag memory location

Return Value

Returns RFID_STATUS of enum type

Remarks

An event is logged through delegate void ReceivedMemoryData() after write command. Then, the writing result can be obtained using GetError.

See Also

GetError

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Write(RFIDWriteCmd *cmd);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();
RFIDUHF.RFIDWriteCmd WriteCmd = new RFIDUHF.RFIDWriteCmd();

WriteCmd.accpwd = Convert.ToUInt32(textBox_RWPwd.Text, 16); // ex. 00000000
WriteCmd.wlength = (INT8U)Convert.ToUInt32(textBox_WCnt.Text); // ex. 6
WriteCmd.offset = (INT8U)Convert.ToUInt32(textBox_OffSet.Text); // ex. 2
WriteCmd.bank = RFIDUHF.RFIDTagBank.TAG_EPC;

String WriteData = textBox_Write_Data.Text; //
UInt16[] Data = new UInt16[32];
for (int i = 0; i < Convert.ToUInt32(textBox_WCnt.Text); i++)
{
    Data[i] = Convert.ToUInt16(WriteData.Substring(i * 4, 4), 16);
}

WriteCmd.wdata = Data;

status = _UHFNet.Write(ref WriteCmd);

if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)
{
    MessageBox.Show("RFID WRITE FAIL!");
}
```

1.4.9 Lock

Description

Limit access to tag

Syntax

```
RFIDUHF.RFID_STATUS Lock(ref RFIDUHF.RFIDLockCmd cmd);
```

Parameters

cmd

RFIDLockCmd structure is used to limit access to tag

Return Value

Returns RFID_STATUS of enum type

Remarks

An event is logged through delegate void ReceivedMemoryData() after lock command. Then, the result can be obtained using GetError.

Access password in reserved memory is used to limit access.

RFIDLockPermission Enum value in RFIDLockCmd structure is used.

PERMISSION_ACCESSIBLE:

Read and write of password is possible. Change permission is also possible.

PERMISSION_ALWAYS_ACCESSIBLE: Read and write of password is possible. But, change permission is not possible.

PERMISSION_SECURED_ACCESSIBLE: Read and write of password is not possible. But, change permission is possible.

PERMISSION_ALWAYS_NOT_ACCESSIBLE: Read and write of password is not possible. Change permission is also not possible.

Read / Write accessibility setting is possible in reserved area. EPC, USER area only allow setting for write, where read is always possible. TID is read only.

See Also

GetError

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Lock(RFIDLockCmd *cmd);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();
```

```
RFIDUHF.RFIDLockCmd LockCmd = new RFIDUHF.RFIDLockCmd();
```

```
LockCmd.lockkillpwd = RFIDUHF.RFIDLockPermission.PERMISSION_NO_CHANGE;
```

```
LockCmd.lockaccesspwd = RFIDUHF.RFIDLockPermission.PERMISSION_SECURED_ACCESSIBLE;
```

```
LockCmd.lockepc = RFIDUHF.RFIDLockPermission.PERMISSION_SECURED_ACCESSIBLE;
```

```
LockCmd.locktid = RFIDUHF.RFIDLockPermission.PERMISSION_NO_CHANGE;
```

```
LockCmd.lockuser = RFIDUHF.RFIDLockPermission.PERMISSION_NO_CHANGE;
```

```
LockCmd.accpwd = Convert.ToUInt32(textBox_Lock_Pwd.Text, 16);
```

```
status = _UHFNet.Lock(ref LockCmd);

if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)
{
    MessageBox.Show("RFID LOCK FAIL!");
}
```

1.4.10 Kill

Description

Kill the tag (disabling the tag)

Syntax

```
RFIDUHF.RFID_STATUS Kill(ref RFIDUHF.RFIDKillCmd cmd);
```

Parameters

cmd

RFIDKillCmd structure is used to disable tag

Return Value

Returns RFID_STATUS of enum type

Remarks

An event is logged through delegate void ReceivedMemoryData() after kill command. Then, the result can be obtained using GetError.

See Also

GetError

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_Kill(RFIDKillCmd *cmd);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();
RFIDUHF.RFIDKillCmd KillCmd = new RFIDUHF.RFIDKillCmd();
```

```
KillCmd.killpwd = uint.Parse(textBox_Kill_Pwd.Text);
```

```
status = _UHFNet.Kill(ref KillCmd);
```

```
if (status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)
{
    MessageBox.Show("RFID KILL FAIL!");
}
```

1.4.11 GetData

Description

Get tag data from Inventory and Read

Syntax

```
int GetData(StringBuilder strTagData);
```

Parameters

strTagData

[out] get current data

Return Value

Returns the length of the data

Remarks

Data obtained by inventory or read can be checked. If return value is 0, error can be checked using GetError.

See Also

Inventory, Read, GetError

For C++

Library : RFID_UHF.lib

Function : int UHF_GetData (INT8U *data);

Example

```
int nTaglenth = 0;
String strTagData = null;
StringBuilder strData = new StringBuilder(260);
nTaglenth = UHFNet.GetData(strData);
```

1.4.12 SetRegionFrequency

Description

Set RFID frequency to suit regional regulation

Syntax

```
RFIDUHF.RFID_STATUS SetRegionFrequency(RFIDUHF.RFIDRegion region);
```

Parameters

region

Set regional frequency by assigning Enum type variable

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

GetRegionFrequency

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_SetRegionFrequency(RFIDRegion region);

Example

```
RFIDUHF.RFID_STATUS status = new RFIDUHF.RFID_STATUS();  
status = _UHFNet.SetRegionFrequency(_nRegionIndex[listBox_Region.SelectedIndex]);  
if(status != RFIDUHF.RFID_STATUS.RFID_STATUS_OK)  
{  
    MessageBox.Show("SetRegionFrequency Set Fail!");  
    return;  
}
```

1.4.13 GetRegionalFrequency

Description

Check current regional frequency setting

Syntax

```
RFIDUHF.RFIDRegion GetRegionFrequency();
```

Parameters

None

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

SetRegionFrequency

For C++

Library : RFID_UHF.lib

Function : RFIDRegion UHF_GetRegionFrequency();

Example

```
RFIDUHF.RFIDRegion region = _UHFNet.GetRegionFrequency();
```

1.4.14 ReadBattery

Description

Check battery voltage and ADC value

Syntax

```
RFIDUHF.RFID_STATUS ReadBattery(ref uint nADCValue, ref float fVolt);
```

Parameters

nADCValue

[out] Check ADC value of battery

fVolt

[out] Check current battery voltage

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

ReadBatteryStatus

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_ReadBattery(INT32U *nADCValue, float *fVolt);

Example

None

1.4.15 ReadBatteryStatus

Description

Check battery level (0~4 level)

Syntax

```
RFIDUHF.RFID_STATUS ReadBatteryStatus(ref RFIDUHF.BATTERY_STATUS step);
```

Parameters

step

[out] Check battery level through Enum type BATTERY_STATUS

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

ReadBattery

For C++

Library : RFID_UHF.lib

Function : RFID_STATUS UHF_ReadBatteryStatus(BATTERY_STATUS *step);

Example

None

1.4.16 Version

Description

Check DLL and F/W version

Syntax

```
bool Version(ref RFIDUHF.RFID_VERSION LibVer, ref RFIDUHF.RFID_VERSION MacVer,  
StringBuilder strDllVersion);
```

Parameters

LibVer

[out] Check version of NRMCE.dll

MacVer

[out] Check reader firmware version

strDllVersion

[out] Check RFDI_UHF.dll version

Return Value

TRUE = Success

FALSE = Fail

Remarks

None

See Also

None

For C++

Library : RFID_UHF.lib

Function : RFID BOOL UHF_Version(RFID_VERSION *LibVer, RFID_VERSION *MacVer, TCHAR *tzDllVersion);

Example

```
RFIDUHF.RFID_VERSION LibVer = new RFIDUHF.RFID_VERSION();  
RFIDUHF.RFID_VERSION MacVer = new RFIDUHF.RFID_VERSION();  
StringBuilder strVersion = new StringBuilder(260);
```

```
_UHFNet.Version(ref LibVer, ref MacVer, strVersion);
```

```
label_Version.Text = String.Format("Firmware: {0}.{1}.{2} App: {3}", MacVer.major,  
MacVer.minor, MacVer.maintenance, Program.APP_VERSION);
```

2 SDK Manual for UGR C++

2.1 Introduction

This document is a reference guide for the software developer's kit (SDK) for UHF Gun Reader.

Module Name	Description			
UHF GUN READER	Insert proper PDA ¹ to read UHF tags.			
	Demo Application	Sample Source	Version	Date
	UHF_GunTest.exe	UHF_Gun_Net_Test	1.0.1	2014-04-24

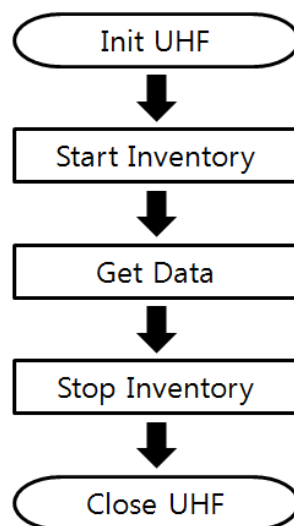
PDA¹ : M3 ORANGE+ WM / M3 ORANGE+ CE / M3 ORANGE S

2.2 Tutorial

This chapter describes the basic usage of M3 Mobile SDK functions in a step-by-step manner. In this tutorial section, the following topics are treated.

Section	Topic
UHF GUN READER	Init UHF Start Inventory Get Data Stop Inventory Close UHF

Basic UHF GUN READER flowchart is shown below.



< UHF RFID flow chart >

Tip!!

- i. It can be used with both Windows Mobile and Windows CE platforms.
- ii. Initialize will return fail in the following conditions:
 - PDA detached from gun reader
 - RFID / SCAN switch is at SCAN position
 - Gun reader's battery is detached or empty
 - PDA in gun reader is synced with PC
- iii. Power status can be obtained using delegate ReceivedPower. When turning off, UHF RFID must be closed. When turning on, UHF RFID must be initialized.
- iv. UHF RFID uses the same virtual key as scanner; VK_H14 for WM and VK_F22 for CE.

2.2.1 Init UHF

```

BEGIN_MESSAGE_MAP(CUHF_GunTestDlg, CDialog)
    ON_MESSAGE(WM_MSG_POWER, ReceivedPower)
END_MESSAGE_MAP()

BOOL Init()
{
    RFID_STATUS status;
    CString strstatus;

    status = UHF_Init(m_hWnd);
    if(status != RFID_STATUS_OK)
    {
        strstatus.Format(_T("RFID Init Error-[%x]"),status);
        // MessageBox(strstatus);

        return FALSE;
    }
    return TRUE;
}

LRESULT ReceivedPower(WPARAM wParam, LPARAM lParam)
{
    BOOL* bPowerOn = (BOOL*)wParam;

    if(*bPowerOn)
    {
        if(m_bOpen == FALSE)
        {
            Init();
            m_bOpen = TRUE;
        }
    }
    else
    {
        if(m_bOpen == TRUE)
        {
            Close();
            m_bOpen = FALSE;
        }
    }
}

```

```
    }  
}  
  
return 0;  
}
```

2.2.2 Start Inventory

```
void Inventory()  
{  
    UHF_Inventory(m_hWnd);  
}
```

2.2.3 Get Data

```
BEGIN_MESSAGE_MAP(CUHF_GunTestDlg, CDialog)  
    ON_MESSAGE(WM_MSG_INVENTORY, ReceivedInventory)  
END_MESSAGE_MAP()  
  
LRESULT ReceivedInventory(WPARAM wParam, LPARAM lParam)  
{  
    int nTaglength = 0;  
    unsigned char czTagData[128] = {0,};  
    CString strTagData;  
  
    nTaglength = UHF_GetData(czTagData);  
  
    if(m_chkCRC.GetCheck())  
    {  
        for(int i=0;i<nTaglength;i++)  
        {  
            strTagData.AppendFormat(_T("%02X"), czTagData[i]);  
        }  
    }  
    else  
    {  
        for(int i=0;i<nTaglength-2;i++)  
        {  
            strTagData.AppendFormat(_T("%02X"), czTagData[i]);  
        }  
    }  
  
    return LRESULT();  
}
```

2.2.4 Stop Inventory

```
void InventoryStop()  
{  
    UHF_InventoryStop();  
}
```

2.2.5 Close UHF

```
BOOL Close()  
{
```

```

RFID_STATUS status;
CString strstatus;

status = UHF_Close();
if(status != RFID_STATUS_OK)
{
    strstatus.Format(_T("%x"),status);
    // MessageBox(_T("RFID Close Error"), strstatus);

    return FALSE;
}

// MessageBox(L"Close");
::SetWindowText(::GetDlgItem(m_hWnd, IDC_STATIC_STATUS), L"Close UHF");
return TRUE;
}

```

2.3 UHF GUN READER (WM/CE)

Define
<pre> typedef unsigned __int8 INT8U; typedef unsigned __int16 INT16U; typedef unsigned __int32 INT32U; typedef signed __int32 BOOL32; typedef unsigned __int32 HANDLE32; typedef HANDLE32 RFID_RADIO_HANDLE; #define WM_MSG_INVENTORY 0x5001 #define WM_MSG_ACCESS 0x5002 #define WM_MSG_POWER 0x5010 #define UHF_OPEN_EVENT L"UHF_OPEN_EVENT" </pre>
Enum
<pre> typedef enum { BATTERY_ERROR, BATTERY_0, BATTERY_1, BATTERY_2, BATTERY_3, BATTERY_FULL }BATTERY_STATUS; typedef enum { TAG_OTHERERROR = 0x0, TAG_SUCCESS = 0x1, TAG_MEMORYOVERRUN = 0x3, TAG_MEMORYLOCKED = 0x4, TAG_INSUFFICIENTPOWER = 0xB, TAG_NONSPECIFICERROR = 0xF, MAC_NOERROR = 0x10, MAC_HANDLEMISSMATCH = 0x11, MAC_CRCERROR = 0x12, MAC_NOTAGREPLY = 0x13, MAC_INVALIDPASSWD = 0x14, MAC_ZEROKILLPASSWD = 0x15, MAC_TAGLOST = 0x16, MAC_COMMANDFORMATERROR = 0x17, </pre>

```

    MAC_READCOUNTINVALID    = 0x18,
    MAC_OUTOFRETRIES          = 0x19,
}RFIDErrorCode;

typedef enum
{
    TAG_RESERVED,
    TAG_EPC,
    TAG_TID,
    TAG_USER
}RFIDTagBank;

typedef enum
{
    RFID_REGION_KOREA_NEW,
    RFID_REGION_KOREA_WEAK,
    RFID_REGION_KOREA_OLD,
    RFID_REGION_USA,
    RFID_REGION_EURO,
    RFID_REGION_EURO_NEW,
    RFID_REGION_JAPAN,
    RFID_REGION_CHINA,
    RFID_REGION_AUSTRALIA,
    RFID_REGION_BRAZIL,
    RFID_REGION_MALAYSIA,
    RFID_REGION_TAIWAN
}RFIDRegion;

typedef enum
{
    SELECT_EPC = 1,
    SELECT_TID = 2,
    SELECT_USER = 3
}RFIDSelectBank;

typedef enum
{
    PERMISSION_ACCESSIBLE,
    PERMISSION_ALWAYS_ACCESSIBLE,
    PERMISSION_SECURED_ACCESSIBLE,
    PERMISSION_ALWAYS_NOT_ACCESSIBLE,
    PERMISSION_NO_CHANGE
}RFIDLockPermission;

enum
{
    /* Success */
    RFID_STATUS_OK,
    /* Attempted to open a radio that is already open */
    RFID_ERROR_ALREADY_OPEN    = -9999,
    /* Buffer supplied is too small */
    RFID_ERROR_BUFFER_TOO_SMALL,
    /* General failure */
    RFID_ERROR_FAILURE,
    /* Failed to load radio bus driver */
    RFID_ERROR_DRIVER_LOAD,
    /* Library cannot use version of radio bus driver present on system */
    RFID_ERROR_DRIVER_MISMATCH,
    /* This error code is no longer used, maintain slot in enum in case
    * anyone is using hard-coded error codes for some reason */
    RFID_ERROR_RESERVED_01,
    /* Antenna number is invalid */
    RFID_ERROR_INVALID_ANTENNA,
    /* Radio handle provided is invalid */

```



```

RFID_ERROR_INVALID_HANDLE,                /* -9992 */
/* One of the parameters to the function is invalid */
RFID_ERROR_INVALID_PARAMETER,             /* -9991 */
/* Attempted to open a non-existent radio */
RFID_ERROR_NO_SUCH_RADIO,                  /* -9990 */
/* Library has not been successfully initialized */
RFID_ERROR_NOT_INITIALIZED,                /* -9989 */
/* Function not supported */
RFID_ERROR_NOT_SUPPORTED,                  /* -9988 */
/* Operation was cancelled by call to cancel operation, close radio, or */
/* shut down the library */
RFID_ERROR_OPERATION_CANCELLED,            /* -9987 */
/* Library encountered an error allocating memory */
RFID_ERROR_OUT_OF_MEMORY,                  /* -9986 */
/* The operation cannot be performed because the radio is currently busy */
RFID_ERROR_RADIO_BUSY,                     /* -9985 */
/* The underlying radio module encountered an error */
RFID_ERROR_RADIO_FAILURE,                  /* -9984 */
/* The radio has been detached from the system */
RFID_ERROR_RADIO_NOT_PRESENT,              /* -9983 */
/* The RFID library function is not allowed at this time. */
RFID_ERROR_CURRENTLY_NOT_ALLOWED,          /* -9982 */
/* The radio module's MAC firmware is not responding to requests. */
RFID_ERROR_RADIO_NOT_RESPONDING,           /* -9981 */
/* The MAC firmware encountered an error while initiating the nonvolatile */
/* memory update. The MAC firmware will return to its normal idle state */
/* without resetting the radio module. */
RFID_ERROR_NONVOLATILE_INIT_FAILED,         /* -9980 */
/* An attempt was made to write data to an address that is not in the */
/* valid range of radio module nonvolatile memory addresses. */
RFID_ERROR_NONVOLATILE_OUT_OF_BOUNDS,       /* -9979 */
/* The MAC firmware encountered an error while trying to write to the */
/* radio module's nonvolatile memory region. */
RFID_ERROR_NONVOLATILE_WRITE_FAILED,        /* -9978 */
/* The underlying transport layer detected that there was an overflow */
/* error resulting in one or more bytes of the incoming data being */
/* dropped. The operation was aborted and all data in the pipeline was */
/* flushed. */
RFID_ERROR_RECEIVE_OVERFLOW,                /* -9977 */
/* An unexpected value was returned to this function by the MAC firmware */
RFID_ERROR_UNEXPECTED_VALUE,               /* -9976 */
/* The MAC firmware encountered CRC errors while trying to */
/* write to the radio module's nonvolatile memory region. */
RFID_ERROR_NONVOLATILE_CRC_FAILED,          /* -9975 */
/* The MAC firmware encountered unexpected values in the packet header */
RFID_ERROR_NONVOLATILE_PACKET_HEADER,       /* -9974 */
/* The MAC firmware received more than the specified maximum packet size */
RFID_ERROR_NONVOLATILE_MAX_PACKET_LENGTH    /* -9973 */
}typedef RFID_STATUS;

```

Structure

```

typedef struct {
    /* The major version (i.e, in 1.x.x.x, the 1) */
    INT32U major;
    /* The minor version (i.e., in x.1.x.x, the 1) */
    INT32U minor;
    /* The maintenance number (i.e., in x.x.1.x, the 1) */
    INT32U maintenance;
    /* The release number (i.e., in x.x.x.1, the 1) */
    INT32U release;
} RFID_VERSION;

typedef struct
{

```

```
HWND hWnd; //Diag handle Receive for Message
RFIDTagBank bank;
INT8U wlength;
INT8U offset;
INT32U accpwd;
}RFIDReadCmd;

typedef struct
{
    HWND hWnd; //Diag handle Receive for Message
    RFIDTagBank bank;
    INT8U wlength;
    INT8U offset;
    INT16U wdata[32];
    INT32U accpwd;
}RFIDWriteCmd;

typedef struct
{
    HWND hWnd; //Diag handle Receive for Message
    RFIDLockPermission lockkillpwd;
    RFIDLockPermission lockaccesspwd;
    RFIDLockPermission lockepc;
    RFIDLockPermission locktid;
    RFIDLockPermission lockuser;
    INT32U accpwd;
}RFIDLockCmd;

typedef struct
{
    HWND hWnd; //Diag handle Receive for Message
    INT32U killpwd;
}RFIDKillCmd;
```

2.4 Functions for UHF GUN READER

Name	Description
UHF_GetError	Check the last error
UHF_IsReady	Check UHF GUN READER's availability by checking the power
UHF_Init	Initialize RFID
UHF_Close	Close RFID
UHF_Inventory	Start Inventory by reading EPC data
UHF_InventoryStop	Stop Inventory
UHF_Read	Read memory from tag
UHF_Write	Write data to tag
UHF_Lock	Limit access to tag
UHF_Kill	Kill the tag (disabling the tag)
UHF_GetData	Get tag data from Inventory and Read
UHF_SetRegionFrequency	Set RFID frequency to suit regional regulation
UHF_GetRegionFrequency	Check current regional frequency setting
UHF_ReadBattery	Check battery voltage and ADC value
UHF_ReadBatteryStatus	Check battery level (0~4 level)
UHF_Version	Check DLL and F/W version

2.4.1 UHF_GetError

Description

Check the last error

Syntax

```
RFIDErrorcode UHF_GetError();
```

Parameters

None

Return Value

Retuens RFIDErrorcode of enum type

Remarks

None

See Also

None

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFIDErrorcode GetError();

Example

```
length = UHF_GetData(data);
if(length == 0)
{
    err = UHF_GetError();
    switch(err)
    {
        case TAG_OTHERERROR:
            MessageBox(_T("Tag Other Error!"));
            break;
        case TAG_SUCCESS:
            MessageBox(_T("SUCCESS!"));
            break;
        case TAG_MEMORYOVERRUN:
            MessageBox(_T("Tag Memory Overrun!"));
            break;
```

```
        case MAC_OUTOFRETRIES:
            MessageBox(_T("Out of Retries Error!"));
            break;
        default:
            MessageBox(_T("Unknown Error!"));
            break;
    }
}
```

2.4.2 UHF_IsReady

Description

Check UHF GUN READER's availability by checking the power

Syntax

```
BOOL UHF_IsReady();
```

Parameters

None

Return Value

TRUE = Success

FALSE = Fail

Remarks

Following cases will return false:

1. PDA detached from gun reader
2. RFID / SCAN switch is at SCAN position
3. Gun reader's battery is detached or empty
4. PDA in gun reader is synced with PC

See Also

None

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : bool IsReady();

Example

```
if(!UHF_IsReady())
```

```
return;
```

2.4.3 UHF_Init

Description

Initialize RFID

Syntax

```
RFID_STATUS UHF_Init (HWND hDlgWnd);
```

Parameters

hDlgWnd

Reader power status message is passed to registered Windows Handle

Return Value

Returns RFID_STATUS of enum type

Remarks

When initializing, WM_MSG_POWER message is passed to registered Windows Handle. On change of power status, this message will return the status. If FALSE is returned, reader should be closed using UHF_Close. If it changes to TRUE, then UHF_Init should be used to re-initialize.

See Also

UHF_Close

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Init();

Example

```
RFID_STATUS status = UHF_Init(m_hWnd);  
if(status != RFID_STATUS_OK)  
{  
    strstatus.Format(_T("RFID Init Error-[%x]"),status);  
    return FALSE;  
}
```

2.4.4 UHF_Close

Description

Close RFID

Syntax

```
RFID_STATUS UHF_Close();
```

Parameters

None

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

UHF_Init

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Close();

Example

```
RFID_STATUS status;  
CString strstatus;  
status = UHF_Close();  
if(status != RFID_STATUS_OK)  
{  
    strstatus.Format(_T("%x"),status);  
    return FALSE;  
}
```

2.4.5 UHF_Inventory

Description

Start Inventory by reading EPC data

Syntax

```
void UHF_Inventory(HWND hWnd);
```

Parameters

hWnd

Register Windows Handle to receive data through inventory

Return Value

None

Remarks

EPC data is passed to registered Windows Handle through WM_MSG_INVENTORY

See Also

UHF_InventoryStop

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : void Inventory();

Example

```
UHF_Inventory(m_hWnd);
```

2.4.6 UHF_InventoryStop

Description

Stop Inventory

Syntax

```
void UHF_InventoryStop();
```

Parameters

None

Return Value

None

Remarks

None

See Also

UHF_Inventory

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : void InventoryStop();

Example

```
UHF_InventoryStop();
```

2.4.7 UHF_Read

Description

Read memory from tag

Syntax

```
RFID_STATUS UHF_Read(RFIDReadCmd *cmd);
```

Parameters

cmd

RFIDReadCmd structure is used to assign tag memory location

Return Value

Returns RFID_STATUS of enum type

Remarks

Windows Handle for receiving data reading message must be registered to hWnd of RFIDReadCmd structure. Data can be obtained using UHF_GetData once WM_MSG_ACCESS message is received.

See Also

UHF_GetData

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Read(ref RFIDUHF.RFIDReadCmd cmd);

Example

```
RFID_STATUS status;
RFIDReadCmd readcmd;
INT32U accpwd = 0;
accpwd = wcstoul(m_strRWPwd, 0, 16);
readcmd.hWnd = this->m_hWnd;
readcmd.bank = (RFIDTagBank)m_cmbBank.GetCurSel();
readcmd.wlength = (INT8U)_ttoi(m_strLength);
readcmd.offset = (INT8U)_ttoi(m_strOffset);
readcmd.accpwd = accpwd;
status = UHF_Read(&readcmd);
if (RFID_STATUS_OK != status)
{
    // Fail!!
}
```

2.4.8 UHF_Write

Description

Write data to tag

Syntax

```
RFID_STATUS UHF_Write(RFIDWriteCmd *cmd);
```

Parameters

cmd

RFIDWriteCmd structure is used to assign tag memory location

Return Value

Returns RFID_STATUS of enum type

Remarks

Windows Handle for receiving data reading message must be registered to hWnd of RFIDWriteCmd structure. Result can be obtained using UHF_GetError once WM_MSG_ACCESS message is received.

See Also

UHF_GetError

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Write(ref RFIDUHF.RFIDWriteCmd cmd);

Example

```
RFID_STATUS status;
RFIDWriteCmd writecmd;
WCHAR *str;
char str1[32][8] = {0,};
INT16U wdata[32] = {0,};
INT8U wlength;
INT8U offset;
INT32U accpwd = 0;
wlength = (INT8U)_ttoi(L"6");
offset = (INT8U)_ttoi(L"2");
accpwd = wcstoul(L"00000000", 0, 16);
str = (WCHAR*)((LPCWSTR)m_strWriteData); // 6 Word – 12 Byte
```

```
for(int i=0;i<wlength;i++)
{
    sprintf(str1[i], "%c%c%c%c", str[i*4], str[i*4+1], str[i*4+2], str[i*4+3]);
    wdata[i] = (INT16U)strtoul(str1[i], 0, 16);
}

writecmd.hWnd = this->m_hWnd;
writecmd.bank = (RFIDTagBank)m_cmbBank.GetCurSel();
writecmd.wlength = wlength;
writecmd.offset = offset;
memcpy(writecmd.wdata, wdata, wlength*2);
writecmd.accpwd = accpwd;

status = UHF_Write(&writecmd);
if (RFID_STATUS_OK != status)
{
    MessageBox(_T("RFID Write Fail!"));
}
```

2.4.9 UHF_Lock

Description

Limit access to tag

Syntax

```
RFID_STATUS UHF_Lock(RFIDLockCmd *cmd);
```

Parameters

cmd

RFIDLockCmd structure is used to limit access to tag

Return Value

Returns RFID_STATUS of enum type

Remarks

Windows Handle for receiving data reading message must be registered to hWnd of RFIDLockCmd structure. Result can be obtained using UHF_GetError once WM_MSG_ACCESS message is received.

Access password in reserved memory is used to limit access.

RFIDLockPermission Enum value in RFIDLockCmd structure is used.

PERMISSION_ACCESSIBLE:

Read and write of password is possible. Change permission is also possible.

PERMISSION_ALWAYS_ACCESSIBLE: Read and write of password is possible. But, change permission is not possible.

PERMISSION_SECURED_ACCESSIBLE: Read and write of password is not possible. But, change permission is possible.

PERMISSION_ALWAYS_NOT_ACCESSIBLE: Read and write of password is not possible. Change permission is also not possible.

Read / Write accessibility setting is possible in reserved area. EPC, USER area only allow setting for write, where read is always possible. TID is read only.

See Also

UHF_GetError

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Lock(ref RFIDUHF.RFIDLockCmd cmd);

Example

```
RFID_STATUS status;
```

```
RFIDLockCmd lock;
```

```
INT32U accpwd;
```

```
accpwd = wcstoul(m_strLockPwd, 0, 16);
```

```
lock.hWnd = this->m_hWnd;
```

```
lock.lockkillpwd = (RFIDLockPermission)m_cmbKillPwd.GetCurSel();
```

```
lock.lockaccesspwd = (RFIDLockPermission)m_cmbAccPwd.GetCurSel();
```

```
lock.lockepc = (RFIDLockPermission)m_cmbEPC.GetCurSel();
```

```
lock.locktid = (RFIDLockPermission)m_cmbTID.GetCurSel();
```

```
lock.lockuser = (RFIDLockPermission)m_cmbUser.GetCurSel();
```

```
lock.accpwd = accpwd;
```

```
status = UHF_Lock(&lock);
```

```
if(RFID_STATUS_OK != status)
{
    MessageBox(_T("RFID Lock Fail!"));
}
```

2.4.10 UHF_Kill

Description

Kill the tag (disabling the tag)

Syntax

```
RFID_STATUS UHF_Kill(RFIDKillCmd *cmd);
```

Parameters

cmd

RFIDKillCmd structure is used to disable tag

Return Value

Returns RFID_STATUS of enum type

Remarks

Windows Handle for receiving data reading message must be registered to hWnd of RFIDKillCmd structure. Result can be obtained using UHF_GetError once WM_MSG_ACCESS message is received.

See Also

UHF_GetError

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS Kill(ref RFIDUHF.RFIDKillCmd cmd);

Example

```
RFID_STATUS status;
RFIDKillCmd kill;
INT32U killpwd;
killpwd = wcstoul(m_strKillPwd, 0, 16);
kill.hWnd = this->m_hWnd;
kill.killpwd = killpwd;
status = UHF_Kill(&kill);
if(RFID_STATUS_OK != status)
```

```
{  
    MessageBox(_T("RFID Kill Fail!"));  
}
```

2.4.11 UHF_GetData

Description

Get tag data from Inventory and Read

Syntax

```
int UHF_GetData (INT8U *data);
```

Parameters

**data*

[out] get current data

Return Value

Returns the length of the data

Remarks

Data obtained by UHF_Inventory or UHF_Read can be checked. If return value is 0, error can be checked using UHF_GetError.

See Also

UHF_Inventory, UHF_Read, UHF_GetError

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : int GetData(StringBuilder strTagData);

Example

```
int length = 0;  
unsigned char data[128] = {0,};  
RFIDErrorcode err;  
CString str;  
length = UHF_GetData(data);  
if(length == 0)  
{  
    err = UHF_GetError();  
}
```

```
else
{
    for(int i=0;i<length;i++)
    {
        str.AppendFormat(_T("%02X"), data[i]);
    }
    m_strReadData = str;
}
```

2.4.12 UHF_SetRegionFrequency

Description

Set RFID frequency to suit regional regulation

Syntax

```
RFID_STATUS UHF_SetRegionFrequency(RFIDRegion region);
```

Parameters

region

Set regional frequency by assigning Enum type variable

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

UHF_GetRegionFrequency

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS SetRegionFrequency(RFIDUHF.RFIDRegion region);

Example

```
RFIDRegion region = RFID_REGION_EURO_NEW;
RFID_STATUS status = UHF_SetRegionFrequency(region);
if(status != RFID_STATUS_OK)
{
    MessageBox(_T("UHF_SetRegionFrequency Fail!"));
}
```

```
    return ;  
}
```

2.4.13 UHF_GetRegionFrequency

Description

Check current regional frequency setting

Syntax

```
RFIDRegion UHF_GetRegionFrequency();
```

Parameters

None

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

UHF_SetRegionFrequency

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFIDRegion GetRegionFrequency();

Example

```
RFIDRegion region = UHF_GetRegionFrequency();
```

2.4.14 UHF_ReadBattery

Description

Check battery voltage and ADC value

Syntax

```
RFID_STATUS UHF_ReadBattery(INT32U *nADCValue, float *fVolt);
```

Parameters

**nADCValue*

[out] Check ADC value of battery

**fVolt*

[out] Check current battery voltage

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

UHF_ReadBatteryStatus

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS ReadBattery(ref uint nADCValue, ref float fVolt);

Example

```
RFID_STATUS status;
INT32U nADC = 0;
float fVolt;
BATTERY_STATUS battstatus;
UHF_ReadBattery(&nADC, &fVolt);
status = UHF_ReadBatteryStatus(&battstatus);
m_strStatus.Format(_T("%.2f(%d)"),fVolt, battstatus);
```

2.4.15 UHF_ReadBatteryStatus

Description

Check battery level (0~4 level)

Syntax

```
RFID_STATUS UHF_ReadBatteryStatus(BATTERY_STATUS *step);
```

Parameters

**step*

[out]Check battery level through Enum type BATTERY_STATUS

Return Value

Returns RFID_STATUS of enum type

Remarks

None

See Also

UHF_ReadBattery

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : RFIDUHF.RFID_STATUS ReadBatteryStatus(ref RFIDUHF.BATTERY_STATUS step);

Example

```
RFID_STATUS status;
INT32U nADC = 0;
float fVolt;
BATTERY_STATUS battstatus;
UHF_ReadBattery(&nADC, &fVolt);
status = UHF_ReadBatteryStatus(&battstatus);
m_strStatus.Format(_T("%.2f(%d)"),fVolt, battstatus);
```

2.4.16 UHF_Version

Description

Check DLL and F/W version

Syntax

```
BOOL UHF_Version(RFID_VERSION *LibVer, RFID_VERSION *MacVer, TCHAR *tzDllVersion);
```

Parameters

**LibVer*

[out] Check version of NRFMCE.dll

**MacVer*

[out] Check reader firmware version

**tzDllVersion*

[out] Check RFDI_UHF.dll version

Return Value

TRUE = Success

FALSE = Fail

Remarks

None

See Also

None

For .Net

Namespace : RFID_UHF_Net.RFIDUHF

Function : bool Version(ref RFIDUHF.RFID_VERSION LibVer, ref RFIDUHF.RFID_VERSION MacVer, StringBuilder strDllVersion);

Example

```
RFID_VERSION LibVer;
```

```
RFID_VERSION MacVer;
```

```
TCHAR tzDllVersion[260] = {0x00};
```

```
CString strstatus;
```

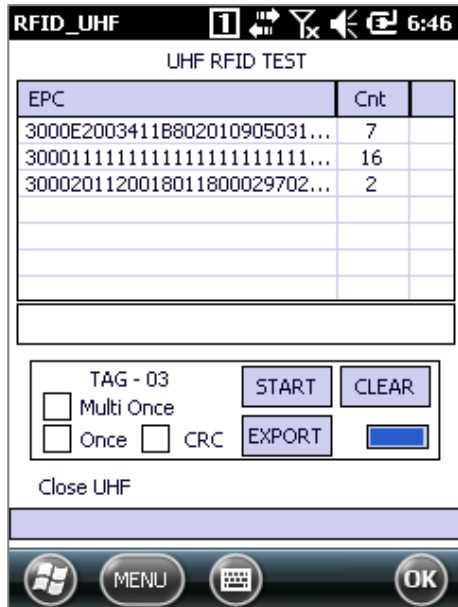
```
UHF_Version(&LibVer, &MacVer, tzDllVersion);
```

```
strstatus.Format(_T("Firmware: %u.%u.%u APP: %s"), MacVer.major, MacVer.minor,  
MacVer.maintenance, VER_APP);
```

3 Demo Manual

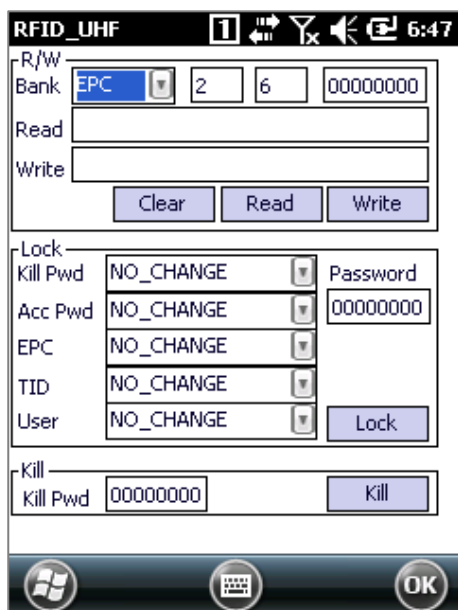
3.1 UHF_GunTest.exe

3.1.1 First view of the program



- It will show simple EPC read and count UI.
 - START: Starts inventory read. Continuously read EPC values of surrounding tags.
 - CLEAR: Clear result.
 - EXPORT: Exports result into a data file.
 - Multi Once: Read surrounding tags only once.
 - Once: Read a tag and stop.
 - CRC: Display CRC with EPC.

3.1.2 Access Menu



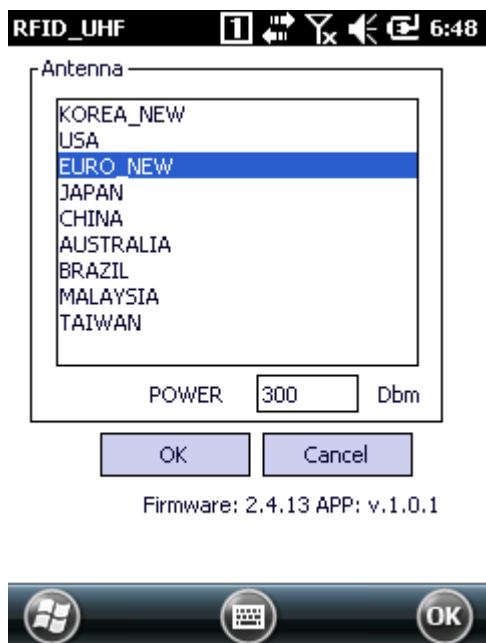
- Read/Write
 - Bank (Reserved, EPC, TID, User) select
 - Offset: Starting location of the memory. (Word unit)
 - Length: Length of the data (Word unit)
 - Password: If the tag is locked, Access Password is required to write data.
 - Read: Read data as configured and output data in the text box.
 - Write: Data in the text box will be written as configured.

- Lock : Set password to access the tag memory. Access password is stored in reserved area.
 - There are 4 access types
 - Accessible: Read and write of password is possible. Change permission is

also possible.

- Always Accessible: Read and write of password is possible. But, change permission is not possible.
- Secured Accessible: Read and write of password is not possible. But, change permission is possible.
- Always Not Accessible: Read and write of password is not possible. Change permission is also not possible.
- Read / Write accessibility setting is possible in reserved area. EPC, USER area only allow setting for write, where read is always possible. TID is read only.
- Kill: Kill the tag. Once killed, the tag is no longer active. Kill password in reserved area is required.

3.1.3 Config Menu



- **Antenna:** Set frequency according to the country regulation.
- **POWER:** Set output power. Maximum power is 300 Dbm.