Scishine Series UHF Reader Reader Generic API User Manual

Version 4.0

Contents

1. Perface

Scishine series UHF RFID Readers use unified communication protocol, use the API function to access connection. In this paper, all data types and functions are described in the standard C language. The corresponding API function library is developed by C language. The order of the parameters of the API function is the order of the standard C language (__cdecl, non-PASSCAL order). Development and application in C/C++ environment, you can refer to the support Demo program directly include API header files and link API function library; if the user use other languages to develop applications, such as C#, Java, Delphi, etc., you can encapsulate C language API library to the corresponding language by yourself.

All the units of overtime in this paper is millisecond.

If the concepts of tag in this article are unclear, please consult ISO18000-6C or EPC ™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID.

In this paper, we provide a reference for the developer of Scishine series of UHF RFID reader. It is written for the reader who have a certain programming foundation.

2. Data Structure

2.1. SSUCtx

Description:

The structure stores all communication context information for the RFID reader. Each instance represents a communication link between a host computer and a reader, which exists in the parameter list of most API functions.

Members:

All the members of SSUCtx are used in the library of the API, the user is not visible.

2.2. SSUDevInfo

Description:

The structure stores the information of connection reader.

Prototype:

```
typedef
struct _st_dev_info
{
    unsigned int    version;        // main(31~24) + sub(23~16) + build(15~0)
```

```
    unsigned int    devSN;          //
    char            devModel[16]; //ascii string ended by '\0'
}
SSUDevInfo;
```

Members:

version:

32 bits of reader's firmware version number, format: the main version number (b31 ~ 24) + the minor version number (b23 ~ 16) + the compile number (b15 ~ 0).

devSN:

Device serial number.

devModel:

Equipment model string, ASCII encoding, end by '\0' character, the length is less than 16 bytes.

2.3. SSUNetAddr

Description:

The structure stores the reader's network address information.

Prototype:

```
typedef struct _st_net_addr
{
    int    netType;    // enumerated by SSU_NET_xxx
    union
    {
        struct
        {
            unsigned char netip[4];
            unsigned char gateway[4];
            unsigned char netmask[4];
        } tcpip;

        unsigned char rs485;
    };
}SSUNetAddr;
```

Members:

netType:

Specify network address type. The address type is defined in the last section of this chapter.

tcpip:

When the address type is TCP or UDP address, memory reader's communication address content (TCP / IP-related addresses).

rs485:

When the address type is rs485, store a byte communication addresses content (the reader network node number).

2.4. SSUTagRecord

Description:

The structure stores the information of tag record.

Prototype:

```
typedef struct _st_tag_record
{
    char antNo;   //antenna No.
    char tagLen;   //bytes of tagid
    unsigned char tagData[64]; //data of tagid
} SSUTagRecord;
```

Members:

antNo:

Identification antenna number, <0 means ignore antenna number.

tagLen:

Identify tag ID number of bytes in the record.

tagData:

Identify tag ID.

2.5. SSU_PWD_EMPTY

Description:

Predefined empty value. Be convent to call general no password.

Prototype:

extern SSU_EXPORT unsigned char SSU_PWD_EMPTY[4];

2.6. Constant Definitions

2.6.1.Communication Network Type

Mark	Value	Description

SSU_NET_NONE	0x00	Point to point no network
SSU_NET_TCPIP	0x01	TCP/IP Network
SSU_NET_RS485	0x02	485 Network
SSU_NET_CAN	0x03	CAN Network (not supported)

2.6.2.Communication Network Type

Mark	Value	Address string format specification
SSU_ADDR_RS232	((SSU_NET_NONE << 8) | 0x00)	COMx
SSU_ADDR_RS485	((SSU_NET_RS485 << 8) | 0x00)	COMx:No
SSU_ADDR_CAN	((SSU_NET_CAN << 8) | 0x00)	The same as above.
SSU_ADDR_TCP	((SSU_NET_TCPIP << 8) | 0x01)	host:port
SSU_ADDR_UDP	((SSU_NET_TCPIP << 8) | 0x02)	host:port

2.6.3.Others

Mark	Value	Description
SSU_TAGID_MAXSIZE	18	Tag ID string maximum number of bytes
SSU_TCP_UDP_PORT	2012	TCP / IP networking device communication port

3. The Basic Function

3.1. SSUSetup

Description:

API library initialization.

Prototype:

void SSUSetup();

3.2. SSUShutup

Description:

Clear API library to release resources.

Prototype:

void SSUShutup();

3.3. SSUNetAddrProc

Description:

Declarations about network address processing callback function type, which is used to receive and process the network address data of the device node, obtained by querying the address.

Prototype:

void (*SSUNetAddrProc)(unsigned int devSn, SSUNetAddr* netAddr);

Parameters:

devSn: Serial number of input device .

netAddr: The network address of the device, and the structure is described in second section ".

3.4. SSUNodeExplore

Description:

Query all devices on the network and their network address.

Prototype:

int SSUNodeExplore(const char* broadcastAddr, int netType, int timeout, SSUNetAddrProc callback);

Parameters:

broadcastAddr: Network broadcast address.

netType: Network type, specify the format of the broadcastAddr string, and values in Section 2.5 macro definitions.

timeout: Input timeout setting.

callback: Process callback function of input network address.

Return:

>=0 –The number of network nodes.

<0 –Error.

3.5. SSUNodeSet

Description:

Set the network address of the specified node on the network.

Prototype:

int SSUNodeSet(const char* broadcastAddr, const unsigned char connpwd[4], unsigned int devSn, const SSUNetAddr* netAddr, int timeout);

Parameters:

broadcastAddr: Broadcast address of input network.

connpwd: Connection password of input target node.

devSn: Device serial number of input target node.

netAddr: New network address.

timeout: Timeout setting.

Return:

=0 –Succeed.

<0 –Fail.

3.6. SSUConnect

Description:

Contact with the device.

Prototype:

SSUCtx* SSUConnect(const char* addr, int addrType, int timeout, const unsigned char pwd[4], unsigned int customNo, SSUDevInfo* devInfo);

Parameters:

addr: Communication (Network) address of input device.

addrType: Address type, specify the format of addr values, the value reference to the value of macro definition at the last section of the chapter 2.

timeout: Timeout settings.

pwd: Connection password.

customNo:Customization number of input device(determined by the specific product range, please refer to the corresponding specification).

devInfo: Device information, including firmware version number, device type, device serial number.

Return:

If succeed, return the command to connect the context pointer, and if fail, return null. If fail, you should first ensure that the network is normal, and then check the address, password and the custom number if is correct.

3.7. SSURestart

Description:

Restart the device.

Prototype:

```
int  SSURestart(SSUCtx* ctx);
```

Parameters:

ctx:

Return:

>=0 –Succeed.

<0 –Fail.

3.8. SSUConfigGet

Description:

Get the working parameters of the device.

Prototype:

int SSUConfigGet(SSUCtx* ctx, unsigned short name, unsigned char* buff, int size);

Parameters:

ctx:

name:Parameter code. The value can be found in the appendix.

buff: Buffer address of input, and Parameter value of output's byte string. The user can analysis according to the parameters.

size: The number of bytes of cache space.

Return:

>=0 –Succeed. Return target parameter value bytes.

<0 –Fail.

3.9. SSUConfigBegin

Description:

Start the process of the device parameter configuration, you can only call SSUConfigGet and SSUConfigSet before calling SSUConfigEng.

Prototype:

int SSUConfigBegin(SSUCtx* ctx);

Parameters:

ctx:

返回**: Return:**

=0 –Succeed.

<0 –Fail.

3.10. SSUConfigSet

Description:

Reset the specified operating parameters for the device.

Prototype:

int SSUConfigSet(SSUCtx* ctx, unsigned short name, const unsigned char* buff, int size);

Parameters:

ctx:

name: The parameter code, the value can be found in the appendix

buff: Parameter values byte string, the content is encoded by the user according to the parameter code.

size: Parameter values bytes.

返回**: Return:**

=0 –Succeed.

<0 – Fail.

3.11. SSUConfigEnd

Description:

End device parameter setting.

Prototype:

int SSUConfigEnd(SSUCtx* ctx, unsigned char save)

Parameters:

ctx:

save: The option of the configure end, 1 means to store the change, 0 means to give up the change.

Return:

>0 –The process is over, give up the change.

=0 –The process is over, and the change is effective.

<0 –Error.

3.12. SSUNetAddrGet

Description:

Query the network address of the device.

Prototype:

```
int  SSUNetAddrGet(SSUCtx* ctx, SSUNetAddr* addr);
```

Parameters:

ctx:

addr: The network type of input , the corresponding network address of output.

Return:

=0 –Succeed.

<0 –Fail.

3.13. SSUNetAddrSet

Description:

Modify the network address of the device.

Prototype:

```
int  SSUNetAddrSet(SSUCtx* ctx, const unsigned char connpwd[4], const SSUNetAddr* addr);
```

Parameters:

ctx:

connpwd:The connection password.

addr:The address to be set, including the type of network and corresponding network addresses.

Return:

=0 –Succeed.

<0 –Fail.

3.14. SSUConnPwdSet

Description:

Set a 4-byte connection password.

Prototype:

```
int  SSUConnPwdSet(SSUCtx* ctx, const unsigned char oldPwd[4], const unsigned char
                   newPwd[4]);
```

Parameters:

ctx:

oldPwd: The current connection password.

newPwd: The new connection password.

Return:

=0 –Succeed.

<0 –Fail.

3.15. SSUDisconnect

Description:

Disconnect command communication connection.

Prototype:

int SSUDisconnect(SSUCtx* ctx, unsigned int force);

Parameters:

ctx:

force: Disconnection option, 0 means no forced disconnection (in this case, if it can not be disconnected in special circumstances, it returns failure), 1 means forced disconnection (in this case, it returns success except Communication breakdown).

Return:

=0 –Succeed.

<0 –Fail.

3.16. SSUTagInvtAuto

Description:

Open automatic inventory process, then the device will enter circulation of automatic identification, the tags entering valid area will immediately be identified and cache. You can only call SSUTagQuery and SSUTagInvtStop before call SSUTagInvtStop.

Prototype:

int SSUTagInvtAuto(SSUCtx* ctx, unsigned char enableReport);

Parameters:

ctx:

enableReport: Automatically reported switch. 1 means to automatically report after the identification, and 0 means to identify the cache waiting for the query command. At present we should fill 1, and 0 is temporarily not supported.

Return:

=0 –Succeed.

<0 –Fail.

3.17. SSUTagInvtStop

Description:

Automatic termination of inventory process, equipment will exit the automatic identification cycle, cache identification record is empty.

Prototype:

int SSUTagInvtStop(SSUCtx* ctx);

Parameters:

ctx:

Return:

=0 –Succeed.

<0 –Fail.

3.18. SSUTagQuery

Description:

Query tag identification record. If the device is in automatic inventory process, use the cached identified record to response; otherwise immediately execute a recognition loop to response.

Prototype:

int SSUTagQuery(SSUCtx* ctx, unsigned char** tag, char* antNo);

Parameters:

ctx:

tag: First address of the tag ID byte string.

antNo: The identified tag antenna number, it can be empty.

Return:

>=0 –Succeed. Return the number of valid bytes of the tag ID byte string.

=0 – Don’t find the tag.

<0 –Error.

3.19. SSUTagRead

Description:

Read the data on the specified storage area of the tag. If the target tag (*tagByts is 0) is not defined, then read any tag, after read successfully, output the tag ID.

Prototype:

int SSUTagRead(SSUCtx* ctx, const unsigned char pwd[4], unsigned char* tag, int* tagByts, int bank, int offset, int words, unsigned char* data, char antNo);

Parameters:

ctx:

pwd: The tag access password, if do not read the reservation area, you can input the SSU_PWD_EMPTY.

tag: Input (or when *tagByts is 0 the output) tag ID byte string, when *tagByts is 0, must ensure that the cache space referred to the tag is not less than SSU_TAGID_MAXSIZE bytes.

tagByts: Input (or when the *tagByts is 0) the ID byte string length of the tag. If the *tagByts is 0, it indicates that read any tag.

bank: Input tag storage area code value, which can be found in the appendix.

offset: Input the starting offset of the reading operation (start with 0, counting unit is 16-bit word).

words: Read the words (16-bit word).

data: Input the address data stored in the cache, Output the successfully read data.

antNo: Specify the working antenna number, -1 indicates ignored.

Return:

>0 –Succeed. Return reading data words(parameter word).

<0 –Fail.

3.20. SSUTagWrite

Description:

Write the data to the specified storage area of the tag. If the target tag (*tagByts is 0) is not defined, then write any tag, after write successfully, output the tag ID

Prototype:

int SSUTagWrite(SSUCtx* ctx, const unsigned char pwd[4], unsigned char* tag, int* tagByts, int bank, int offset, int words, const unsigned char* data, char antNo);

Parameters:

ctx:

pwd: Input the tag access password.

tag: Input (or when *tagByts is 0 the output) tag ID byte string, when *tagByts is 0, must ensure that the cache space referred to the tag is not less than SSU_TAGID_MAXSIZE bytes.

tagByts: Input (or when the *tagByts is 0) the ID byte string length of the tag. If the *tagByts is 0, it indicates that read any tag.

bank: Input tag storage area code value, which can be found in the appendix.

offset: Input the starting offset of the reading operation (start with 0, counting unit is 16-bit word).

words: Write the words (16-bit word).

data: Input the data cache to be written.

antNo: Specify the working antenna number, -1 indicates ignored.

Return:

>0 –Succeed. Return reading data words (parameter word).

<0 –Fail.

3.21. SSUTagLock

Description:

Perform a specified lock or unlock action on the specified tag.

Prototype:

int SSUTagLock(SSUCtx* ctx, const unsigned char pwd[4], const unsigned char* tag, int tagByts, unsigned char lckData[3], char antNo);

Parameters:

ctx:

pwd: Input the tag access password.

tag: Input tag ID byte string.

tagByts: Input the ID byte string length of the tag, tagByts>0.

ockData: Description of the Input action, the value can be found in the appendix.

antNo: Specify the working antenna number, -1 indicates ignored.

Return:

=0 –Succeed.

<0 –Fail.

3.22. SSUTagKill

Description:

Perform Kill action on the specified tag.

Prototype:

int SSUTagKill(SSUCtx* ctx, const unsigned char apwd[4], const unsigned char kpwd[4], const unsigned char* tag, int tagByts, char antNo);

Parameters:

ctx: Input the device context.

apwd: Input the tag access password.

kpwd: Input the tag kill password.

tag: Input tag ID byte string.

tagByts: Input the ID byte string length of the tag, tagByts>0.

antNo: Specify the working antenna number, -1 indicates ignored.

Return:

=0 –Succeed.

<0 –Fail.

4. Extended Functions

Extended functions are appropriate simplification package to the basic functions, which can facilitate to quickly develop for the users. Using extended functions Note: Tags associated password is the extension password, which is a variable length strings ended by '\ 0' character, and is different from 4-byte basic password formats. If basic functions require extended password, we can use the SSUConvPwd password to converted the extended password to basic password.

4.1. SSUTagReadTID

Description:

Read EPC TID partition content.

Prototype:

int SSUTagReadTID(SSUCtx* ctx, unsigned char* tag, int* tagByts, int offset, int words, unsigned char* data);

Parameters:

ctx: Input the device context.

tag: Input (or when *tagByts is 0 the output) tag ID byte string, when *tagByts is 0, must ensure that the cache space referred to the tag is not less than SSU_TAGID_MAXSIZE bytes.

tagByts: Input (or when the *tagByts is 0) the ID byte string length of the tag. If the *tagByts is 0, it indicates that read any tag.

offset: Input the starting offset of the reading operation (start with 0, counting unit is 16-bit word).

words: Read the words (16-bit word).

data: Input the address data stored in the cache, Output the successfully read data.

Return:

>0 –Succeed. Return words.

<0 –Fail.

4.2. SSUTagReadUSR

Description:

Read EPC USER partition content.

Prototype:

int SSUTagReadUSR (SSUCtx* ctx, unsigned char* tag, int* tagByts, int offset, int words, unsigned char* data);

Parameters:

The same as above.

Return:

The same as above.

4.3. SSUTagWriteUSR

Description:

写 EPC 标签 USER 区的内容. Write EPC USER partition content.

Prototype:

int SSUTagWriteUSR(SSUCtx* ctx, const char* tagPwd, unsigned char* tag, int* tagByts, int offset, int words, const unsigned char* data);

Parameters:

ctx: Input the device context.

tagPwd: Input a tag access password string, '\ 0' is the end.

tag: Input (or when *tagByts is 0 the output) tag ID byte string, when *tagByts is 0, must ensure that the cache space referred to the tag is not less than SSU_TAGID_MAXSIZE bytes.

tagByts: Input (or when the *tagByts is 0) the ID byte string length of the tag. If the *tagByts is 0, it indicates that read any tag.

offset: Input the starting offset of the reading operation (start with 0, counting unit is 16-bit word).

words: Operate the words (16-bit word).

data: Enter the data to be written.

Return:

>0 –Succeed. Return words.

<0 –Fail.

4.4. SSUTagResetEPC

Description:

Reset EPC tag ID. This function should be cautious to call. If have an error, the tag ID is uncertain. So ensure that the tag should be in the signal coverage area where write stably. This function can not be used to process the tag which supports XPC.

Prototype:

int SSUTagResetEPC(SSUCtx* ctx, const char* tagPwd, const unsigned char* tag, int tagByts,

const unsigned char* epc, int epcByts);

Parameters:

ctx: Input the device context.

tagPwd: Input a tag access password string, '\ 0' is the end.

tag: Input the target tag ID byte string, ! NULL.

tagByts: Input the tag ID number of bytes, > 0.

epc: Input the new ID byte string.

epcByts: Input the new ID bytes.

Return:

=0 –Succeed.

<0 –Fail.

4.5. SSUTagResetPWD

Description:

Reset EPC access to tag or kill the password. This function should be cautious to call. If have an error, the tag password is uncertain. So ensure that the tag should be in the signal coverage area where write stably.

Prototype:

int SSUTagResetPWD(SSUCtx* ctx, const char* tagPwd, const unsigned char* tag, int tagByts, const char* accPwd, const char* killPwd);

Parameters:

ctx: Input the device context.

tagPwd: Input a tag access password string, '\ 0' is the end.

tag: Input the target tag ID byte string, ! NULL.

tagByts: Input the tag ID number of bytes, > 0.

accPwd: Input a new 4 byte access password, and NULL indicates that the access password is ignored.

killPwd: Input a new 4 byte kill password, and NULL says that the kill password is ignored.

Return:

=0 –Succeed.

<0 –Fail.

4.6. SSUTagSolitifyEPC

Description:

Cure (permanently locked) EPC tag ID. This function should be called with caution, after succeed, the

tag ID can not be reset.

Prototype:

int SSUTagSolitifyEPC(SSUCtx* ctx, const char* tagPwd, const unsigned char* tag, int tagByts);

Parameters:

ctx: Input the device context.

tagPwd: Input a tag access password string, '\ 0' is the end.

tag: Input the target tag ID byte string, ! NULL.

tagByts: Input the tag ID number of bytes, > 0.

Return:

=0 –Succeed.

<0 –Fail.

4.7. SSUTagSolitifyUSR

Description:

Cure (permanently locked) EPC tag USER area. This function should be called with caution, after succeed, the content of USER area can not be reset.

Prototype:

int SSUTagLockUSR(SSUCtx* ctx, const char* tagPwd, const unsigned char* tag, int tagByts);

Parameters:

The same as SSUTagSolifityEPC.

Return:

The same as SSUTagSolifityEPC.

4.8. SSUTagKillEx

Description:

Kill the specified label. This function should be called carefully, after succeed, the label will lose function.

Prototype:

int SSUTagKillEx(SSUCtx* ctx, const char* tagAccPwd, const char* tagKillPwd, const unsigned char* tag, int tagByts);

Parameters:

ctx: Input the device context.

tagAccPwd: Input a tag access password string, '\ 0' is the end.

tagKillPwd: Input a tag kill password string, '\ 0' is the end.

tag: Input the target tag ID byte string, ! NULL.

tagByts: Input the tag ID number of bytes, > 0.

Return:

=0 –Succeed.

<0 –Fail.

4.9. SSUConvPwd

Description:

Convert the password of the string format in the extended access function to the original 4 byte basic password (the storage value in the tag). When use the extended function to set the password, but use the basic function to read and write the tag, you need to use the function to get the basic password of 4 bytes.

Prototype:

void SSUConvPwd(const char* passwdStr, unsigned char passwdBuf[4]);

Parameters:

passwdStr: Password string.

passwdBase: Output 4 bytes basic password.

Return:

None.

5. Appendix

5.1. Error Code Table

Error code identification	Code value	Error Description
……	>0	After the successful execution of the command, the definition of special meaning return value
SSE_SUCCESS	0	The command completed successfully
……	-1~ -49	Reserved
SSE_CMD _INV	-50	Unsupported command
SSE_INPUT _INV	-51	Invalid input parameter
……	-52~ -79	Reserved
SSE_TAG_MEM_OVR	-80	Tag storage position out of bounds
SSE_TAG_MEM_LCK	-81	Tag storage area is locked
SSE_TAG_PWR	-82	Tags energy is not enough
……	-83~-109	Reserved
……..	-110~-126	Custom error code reserved.

SSE_FAIL	-127	Command fails (for unknown reasons or without specify the reasons)

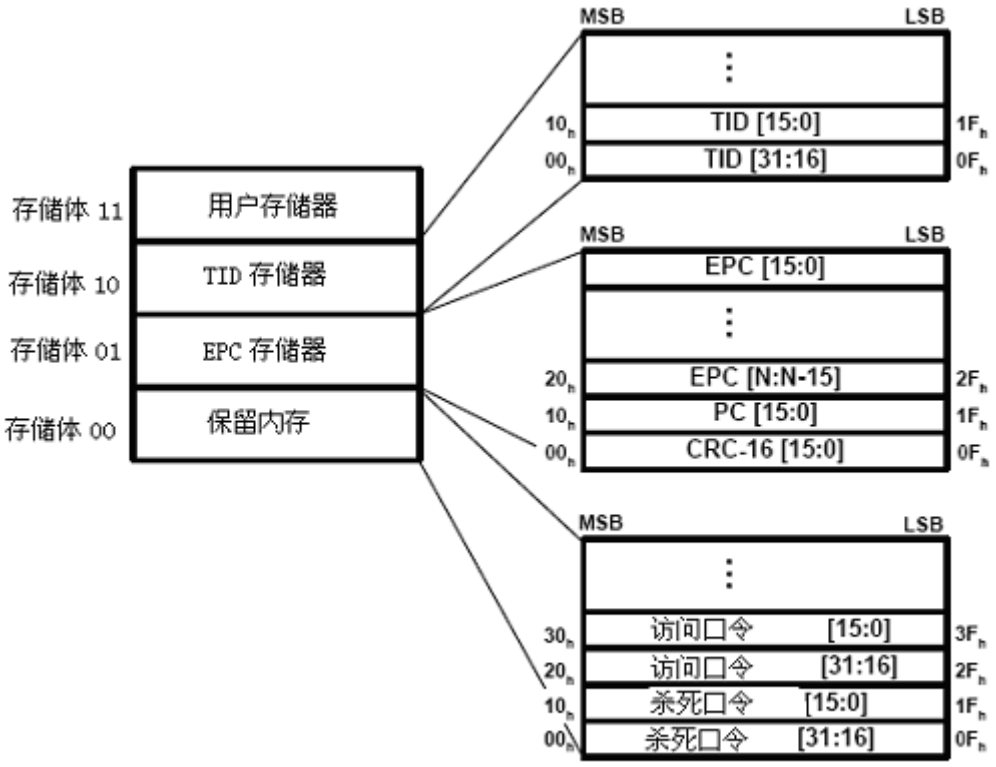
5.2. Reader Operating Parameters Table

The parameters listed in the following table are possible generic parameters, which may be increased or decreased in different types of reader. As appropriate, it is best to set up an independent parameter list according to the model.

Grade	**Parameter name**	**Type**	**Length**	**Range**	**Description**
1	invtRspTm	Integer	2B	[-1, 32767]	The maximum response time inventory, a major role in the automatic inventory controls the inventory efficiency and command response balance, in milliseconds. -1 (<0) means that only an inventory of finished or the label ID cache is full, response command (not recommended the design value). > = 0 means any one of the following three conditions are satisfied can respond to commands: one end inventory, tag ID cache is full, the arrival response time.
2	invtBufTm	Integer	1B	[0, 31]	标签盘存刷新时间，秒。缺省值0，表示每次识别即时刷新。本参数用于减少标签识别信息的传输数据量 （暂不支持） Label inventory refresh time, in seconds. The default value is 0, which means each time identifying immediate refresh. This parameter is used to reduce transfer data amount of tag identification information. (Not supported)

3	invtSess	Integer	1B	[0,3]	EPC session number is used in the inventory
4	invtQMax	Integer	1B	[0, 15]	Maximum inventory Q. 0 turn off automatic inventory, >0 start automatic inventory and as the maximum value of Q
5	invtQMin	Integer	1B	[0, 4]	Instant Inventory (or automatic inventory minimum) value of Q
6	invtQMinTries	Integer	1B	[0, 15]	At the end stage of an inventory process, in order to dish out the missing tag as much as possible, the number of times the Q value set by repeatedly invtQMin reforming inventory
7	invtIdleMin	Integer	1B	[0, 255]	Minimum idle time between two inventory process, ms, in order to cool the PA.
8	invtIdleFract	Integer	1B	[1, 255]	Inventory of idle time for inventory work time points, PA for cool to prevent overheating. Such as: 8 indicates that the working time and the idle time ratio is 8:1.
21	rfAntMask	Bitfield	1B		Antenna channel selection bit field, each representing an antenna. Set 1 means that the antenna is enabled, 0 means the antenna is not enabled. Such as 0x05 means zeroth, 2nd antenna is enabled
22	rfGain	Integer	1B	[0,31]	RF gain level
23	rfCenter	Integer	4B	>0	RF carrier center frequency, KHz
24	rfDeviation	Integer	2B	>0	Floating carrier frequency value, KHz
25	rfStep	Integer	2B	>0	Carrier frequency adjustment step, KHz
253	beepEnable	Integer	1B	{0, 1}	Enable or disable beep
255	linkHoldSec	Integer	1B	>=0	Link detection interval (seconds) 0 indicates no detection (not supported)

5.3. EPC Tag Storage Partition Structure



5.4. The Internal Structure Of The LCKDATA Parameter In The SSUTAGLOCK Function

Operating masks									
Kill password		**Access password**		**UII memory**		**TID memory**		**User memory**	
19	18	17	16	15	14	13	12	11	10
Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write
9	8	7	6	5	4	3	2	1	0
Pwd read/ write	Perma lock	Pwd read/ write	Perma lock	Pwd write	Perma lock	Pwd write	Perma lock	Pwd write	Perma lock

Operation code		
Pwd-write	**Permalock**	**Description**
0	0	The corresponding data segment can be written in OPEN or SECURED state
0	1	In the OPEN or SECURED state, the corresponding data segment can be permanently written, and the corresponding data segment can not be locked.

1	0	The corresponding data segment can be written in SECURED state, OPEN state can not be written
1	1	The corresponding data segment is not written in any state
Pwd-read/write	**Permalock**	**Description**
0	0	The corresponding data segment can be read and written in OPEN or SECURED state
0	1	The corresponding data segment can be read and written in OPEN or SECURED state, the corresponding data segment can not be locked.
1	0	The corresponding data segment can be read and written in SECURED state, the OPEN state can not read and write.
1	1	The corresponding data segment is not read and write in any state

5.5. The Internal Structure Description Of The EPC Tag ID Byte String In this Paper

The internal structure of the EPC tag ID byte string described in this paper is shown in the following table:

<table>
<tr><td colspan="4">PC</td><td colspan="3">XPC</td><td>EPC</td></tr>
<tr><td>LEN</td><td>UMI</td><td>XI</td><td>NSI</td><td colspan="2">W1</td><td>W2</td><td rowspan="3">EPC Code string</td></tr>
<tr><td rowspan="2">XPC + EPC words, 5 binary</td><td rowspan="2">USER non-empty indicator bit</td><td rowspan="2">W1 exists indicator bit</td><td rowspan="2">9 binary</td><td>XEB</td><td>Left</td><td rowspan="2">16 binary</td></tr>
<tr><td>W2 exists indicator bit</td><td>15 binary</td></tr>
</table>

Each part is arranged according to big-endian, MSB LSB on the left, on the right. Among them, LEN indicates the total number of XPC and EPC words (16 words). If the position of XI is 1, it means that W1 exists. If set XEB to 1, it means that W2 exists. So the actual length of the EPC code string is determined by XI and XEB.

UMI, Xi, XPC are read-only domain. When rewrite the EPC tag, we need to guarantee not to change the original value, otherwise it may lead to unpredictable consequences. At present most EPC tags do not support XPC, so when you get the position of XI in the tag ID string, you should recognize whether the tag supports XPC.

The concept of PC, XPC, EPC, UMI, Xi, NSI, XPC-W1, XPC-W2 and so on, which are in the tag ID internal structure, can be found in detail in the EPC ™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID document instructions.

5.6. Function Call Sequence Diagram