# RK Vendor Storage Application Note

2016.DEC.21

YiFeng.Zhao

# Revision History

| Revision No. | Revised Details | Released Date | Remark |
|---|---|---|---|
| Rev.00 | Initial Draft | 2016.11.28 | |
| Rew.01 | Update ID DEFINE | 2016.12.21 | |

# Contents

◆ Application Note Summary

◆ Vendor Storage Architecture

◆ Data Layout

◆ ID DEFINE

◆ API For UBOOT

◆ API For Kernel

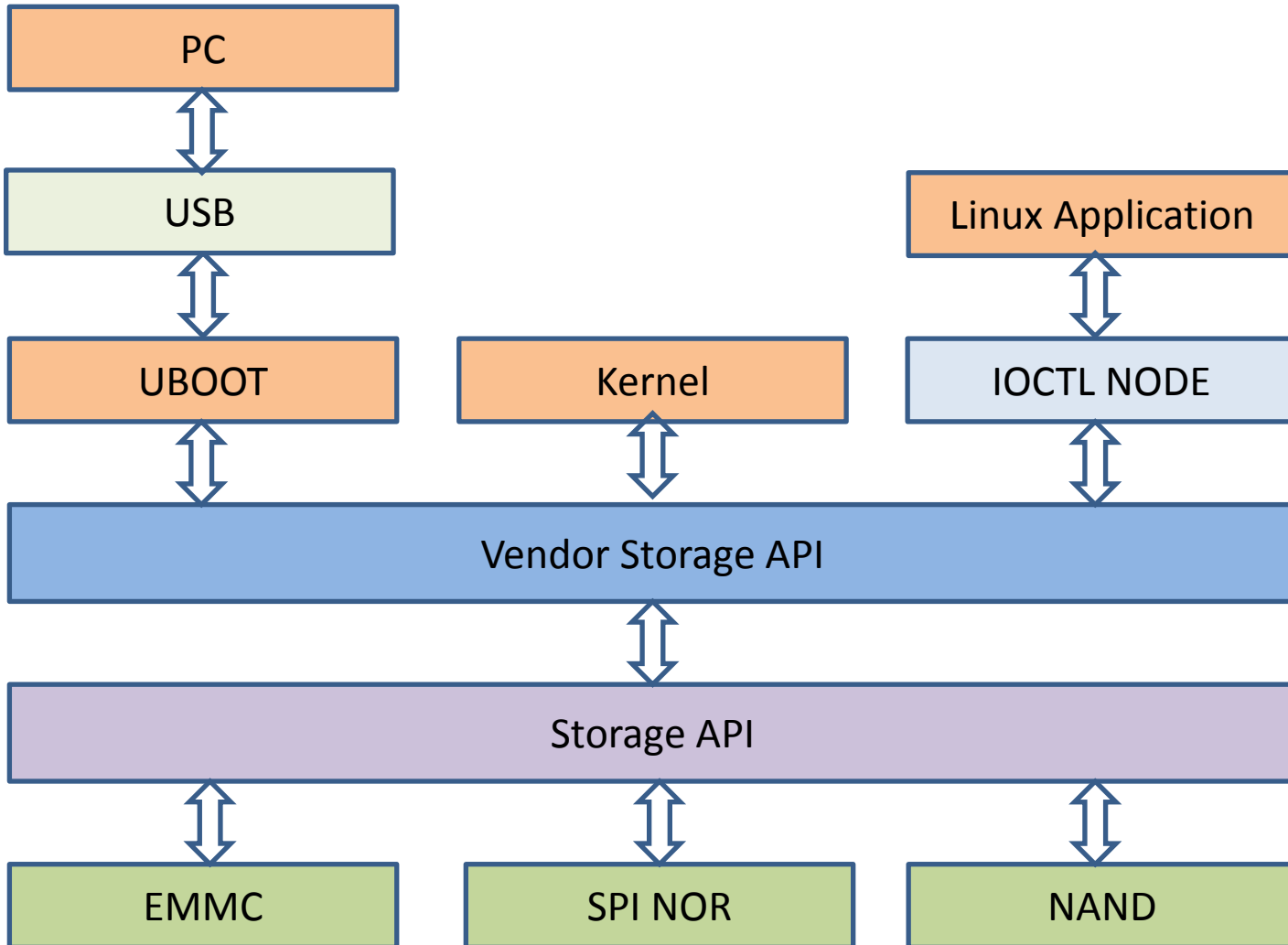◆ PC Demo Tool

◆ Linux Demo Code

# Application Note Summary

Vendor storage is designed for stored SN, MAC and other vendor data.

Feature:

◆ Unique ID Access

◆ Reliable Data Validation

◆ Power Lost Recovery

◆ Writable and Readable for PC

◆ Writable and Readable for UBOOT

◆ Writable and Readable for Kernel
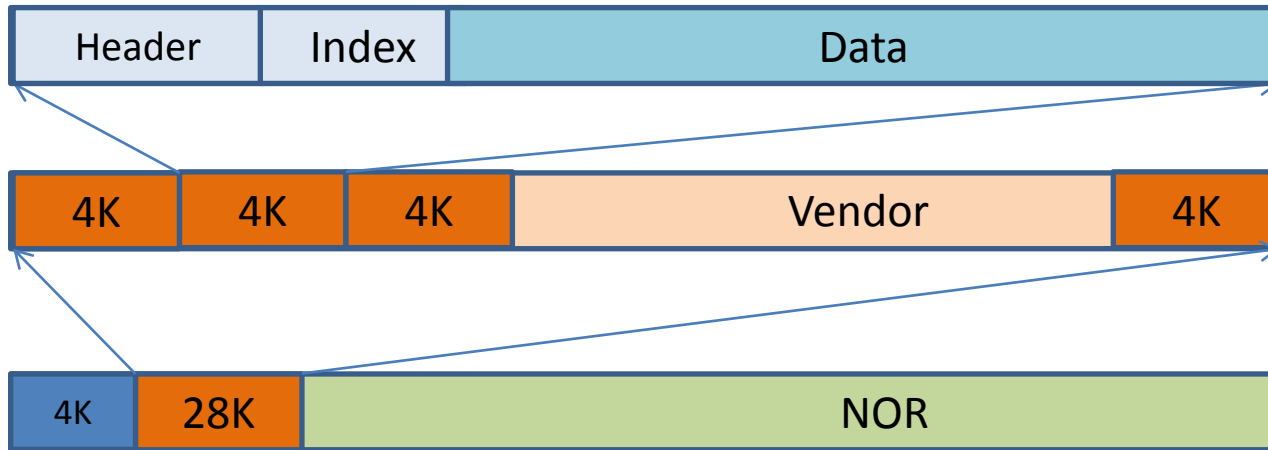
◆ Writable and Readable for Linux Application
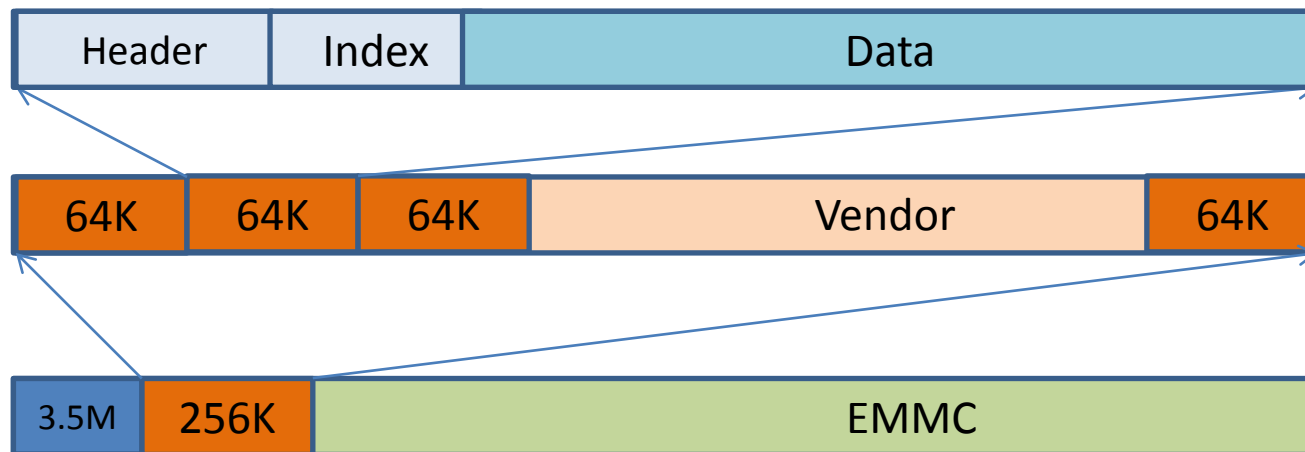
# Vendor Storage Architecture

# Data Layout (1)

SPI NOR Data Layout:

| Header | Index | Data |
|--------|-------|------|

| 4K | 4K | 4K | Vendor | 4K |
|----|----|----|--------|----|

| 4K | 28K | NOR |
|----|-----|-----|

EMMC Data Layout:

| Header | Index | Data |
|--------|-------|------|

| 64K | 64K | 64K | Vendor | 64K |
|-----|-----|-----|--------|-----|

| 3.5M | 256K | EMMC |
|------|------|------|

# Data Layout (2)

| Offset(B) | Size(B) | Name |
|---|---|---|
| 0x0000 | 0x0004 | Tag, the value is 0x524B5644. |
| 0x0004 | 0x0004 | Version, increase after write |
| 0x0008 | 0x0002 | Next index |
| 0x000A | 0x0002 | Total items |
| 0x000C | 0x0002 | Free offset |
| 0x000E | 0x0002 | Free size |
| 0x0010 | 0x0002 | Item0->id |
| 0x0012 | 0x0002 | Item0->offset |
| 0x0014 | 0x0002 | Item0->size |
| 0x0016 | 0x0002 | Item0->flag(reversed) |
| 0x0018 | 0x0008 | Item1 |
| … | … | Item x |
| --- | --- | Data |
| Size – 0x08 | 0x0004 | Hash |
| Size – 0x04 | 0x0004 | Version2, the value is the same as the Version |

# ID DEFINE

| ID | Function |
|---|---|
| 0 | reserved |
| 1 | SN |
| 2 | WIFI MAC |
| 3 | LAN MAC |
| 4 | BT MAC |
| 5 - 15 | RK reserved for future use |
| 16 - 65535 | Vendor use |

Note:

Data is in accordance with the 64 - bytes alignment  stored in the NVM, this suggests that a minimum allocation for each item is 64 bytes, so write 1-64 bytes, the space will be allocated as 64 bytes,  write 65-128 bytes, the space will be allocated as 128 bytes.

# API For UBOOT

Source code : u-boot/board/rockchip/common/storage/storage.c

API:

◆ int vendor_storage_init(void)

    function: Initialize vendor storage

    input : none

    return:  0, Initialize success

            other, Initialize fail


◆ int vendor_storage_read(u32 id, void *pbuf, u32 size)

    function: read vendor storage by id

    input : id, item id; pbuf, data buffer; size, number byte to read.

    return:  other: number byte have read.

            -1, read fail.

# API For UBOOT

◆ int vendor_storage_write(u32 id, void *pbuf, u32 size)

function: write vendor storage by id

input : id: item id; pbuf: data buffer; size: number bytes to write.

return:  0: write success

other: write fail

# API For Kernel

Source code : kernel/drivers/soc/rockchip/rk_vendor_storage.c

kernel/drivers/soc/rockchip/sdmmc_vendor_storage.c

API:

◆ int rk_vendor_read(u32 id, void *pbuf, u32 size)

function: read vendor storage by id

input : id: item id; pbuf: data buffer; size: number bytes to read.

return:  other: number byte have read.

-1: read fail

◆ int rk_vendor_write(u32 id, void *pbuf, u32 size)

function: write vendor storage by id

input : id, item id; pbuf: data buffer; size: number bytes to write.

return:  0: write success

other : write fail

# Kernel configuration

Makefile:

```
#
# Rockchip Soc drivers
#
obj-$(CONFIG_MMC_DW_ROCKCHIP)    += sdmmc_vendor_storage.o
obj-$(CONFIG_ROCKCHIP_VENDOR_STORAGE)    += rk_vendor_storage.o
```

Menuconfig:

```
CONFIG_ROCKCHIP_VENDOR_STORAGE:

Say y here to enable vendor storage support.
Vendor storage is used for stored SN, MAC, BT ADDR etc.

Symbol: ROCKCHIP_VENDOR_STORAGE [=y]
Type   : boolean
Prompt: Rockchip vendor storage support
  Location:
    -> Device Drivers
      -> SOC (System On Chip) specific Drivers
  Defined at drivers/soc/rockchip/Kconfig:6
  Depends on: ARCH_ROCKCHIP [=y] || COMPILE_TEST
```

# PC Demo Tool

Select SN option and input SN value, press **ENTER** key to add SN item to write request list：
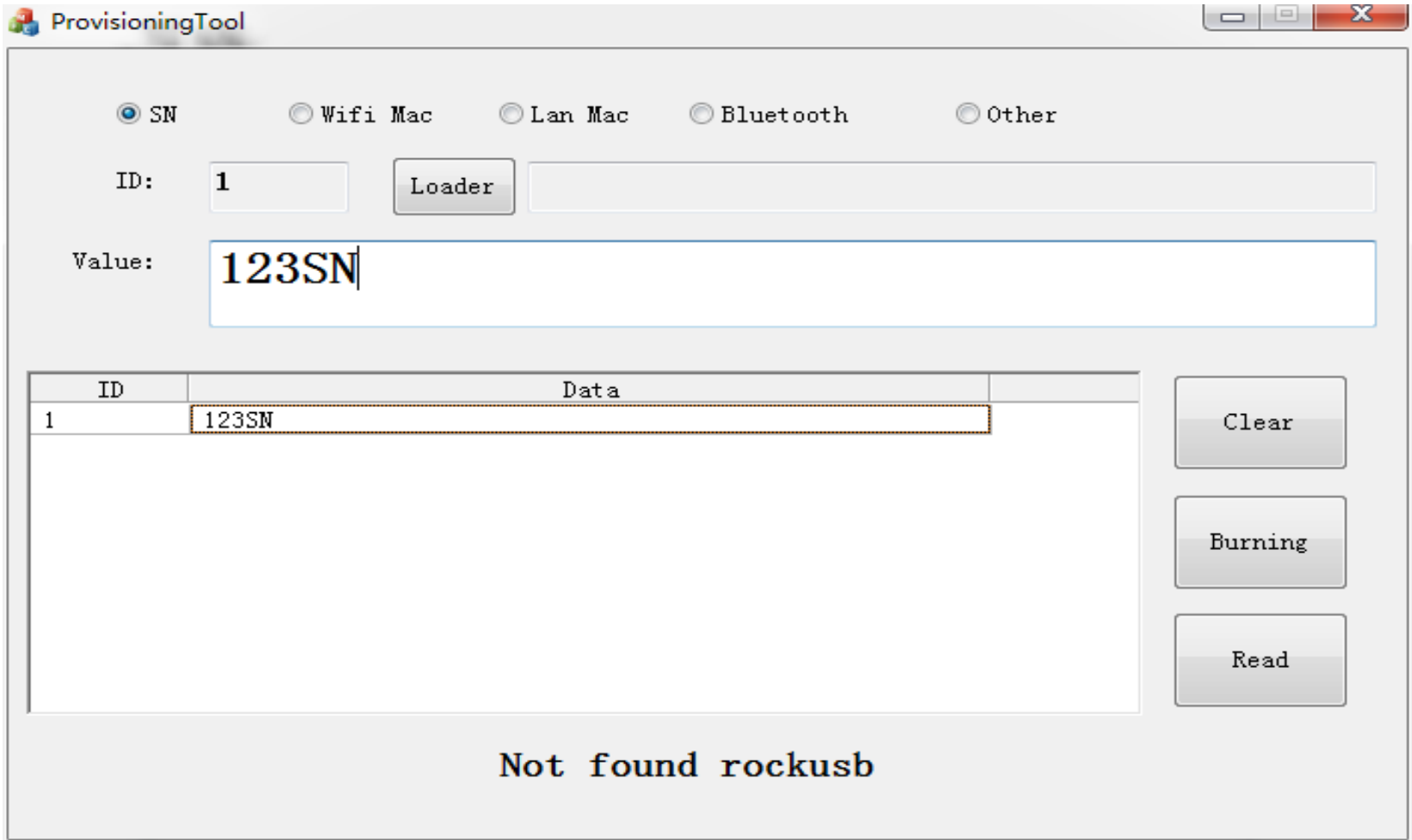
# PC Demo Tool

Press **Burning** button will write data to vendor storage:

# PC Demo Tool
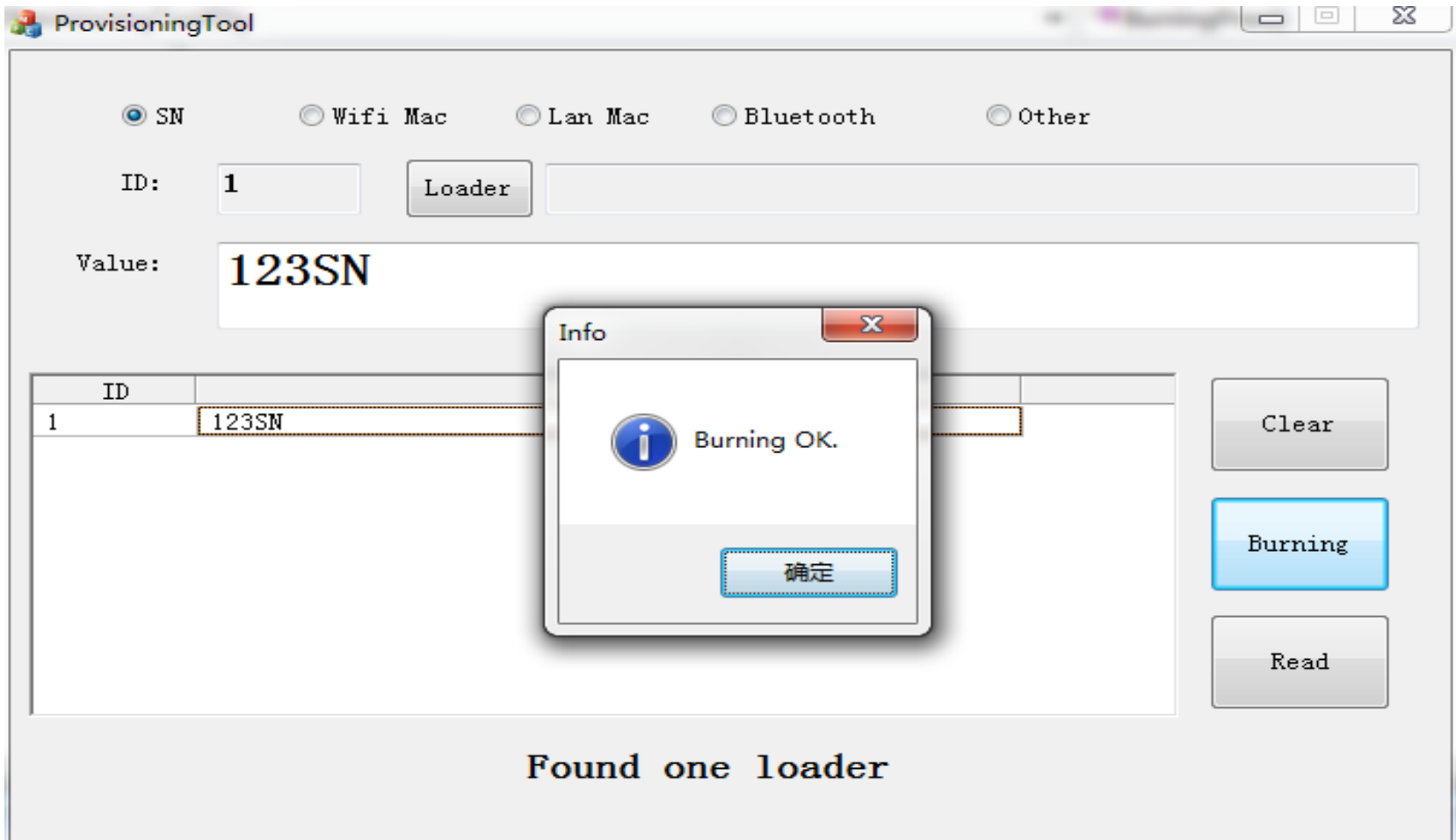
Select SN option and press **Read** button to read data form vendor storage：
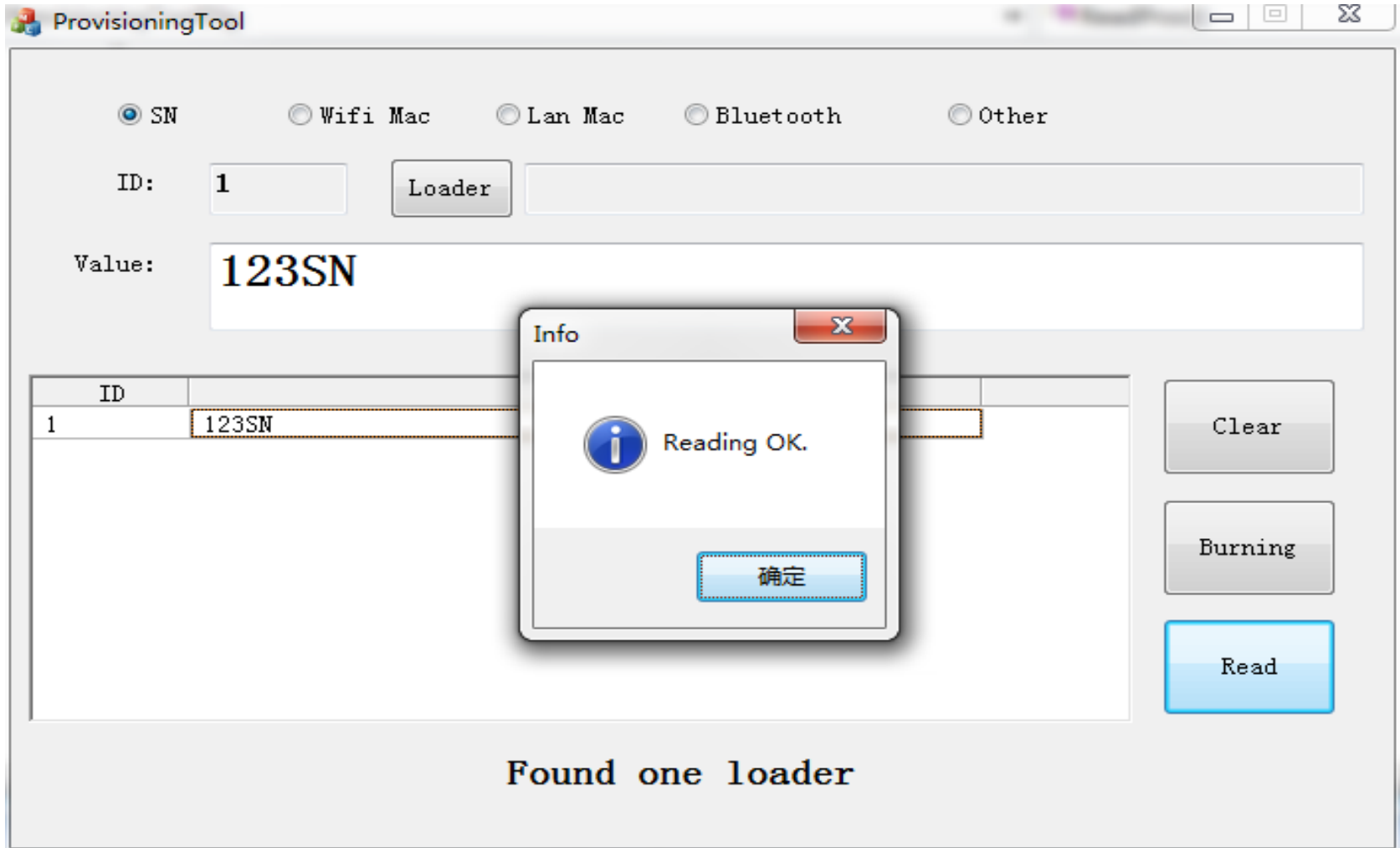
# PC Tool Source Code

| | | | |
|---|---|---|---|
| cmLib | 2016/8/12 10:33 | 文件夹 | |
| colorEdit | 2016/8/12 10:33 | 文件夹 | |
| FontStatic | 2016/8/12 10:33 | 文件夹 | |
| grid | 2016/8/12 10:33 | 文件夹 | |
| res | 2016/8/12 10:33 | 文件夹 | |
| upgradeLib | 2016/8/12 10:33 | 文件夹 | |
| ProvisioningTool.aps | 2016/8/5 10:11 | APS 文件 | 103 KB |
| ProvisioningTool.cpp | 2016/8/3 16:29 | C++ source file | 2 KB |
| ProvisioningTool.h | 2016/8/3 16:29 | H 文件 | 1 KB |
| ProvisioningTool.rc | 2016/8/5 10:11 | VisualStudio.rc.1... | 6 KB |
| ProvisioningTool.vcproj | 2016/8/4 9:14 | VC++ Project | 7 KB |
| ProvisioningTool.vcproj.HP-INRPC6N... | 2016/8/12 10:31 | Visual Studio Pr... | 2 KB |
| ProvisioningToolDlg.cpp | 2016/8/11 10:48 | C++ source file | 18 KB |
| ProvisioningToolDlg.h | 2016/8/9 8:42 | H 文件 | 3 KB |
| ReadMe.txt | 2016/8/3 15:01 | TXT 文件 | 3 KB |
| resource.h | 2016/8/5 10:11 | H 文件 | 2 KB |
| stdafx.cpp | 2016/8/3 15:01 | C++ source file | 1 KB |
| stdafx.h | 2016/8/3 16:29 | H 文件 | 3 KB |
| targetver.h | 2016/8/3 15:01 | H 文件 | 2 KB |

# PC Tool Read Data

```
/*---------------------------------------------------------------------

Name    :   RK_ReadProvisioningData

Desc     :   Read provisioning data by id

Params  :   (IN)nID         id

             (OUT)pDataBuffer:    buffer to save data, malloc by caller

             (IN|OUT)nBufferSize: in = size of buffer,out = actual data size

             (IN)dwLayer:device layer

Return  :   TRUE:       SUCCESSED

             FALSE:       FAILED

----------------------------------------------------------------------*/

BOOL RK_ReadProvisioningData(USHORT nID,PBYTE pDataBuffer,USHORT
&nBufferSize,DWORD dwLayer=0);
```

# PC Tool Write Data

```
 /*----------------------------------------------------------------

Name    :   RK_WriteProvisioningData

Desc     :    Write provisioning data by id

Params  :  (IN)nID          id

                (IN)pDataBuffer:   buffer to save data, malloc by caller

                (IN)nBufferSize:   size of buffer

                (IN)dwLayer:device layer

Return  :   TRUE:      SUCCESSED

                 FALSE:      FAILED

----------------------------------------------------------------------*/

BOOL RK_WriteProvisioningData(USHORT nID,PBYTE pDataBuffer,USHORT
nBufferSize,DWORD dwLayer=0);
```

# Linux Demo Code

```c
#define VENDOR_REQ_TAG          0x56524551
#define VENDOR_READ_IO          _IOW('v', 0x01, unsigned int)
#define VENDOR_WRITE_IO         _IOW('v', 0x02, unsigned int)

#define VENDOR_SN_ID            1
#define VENDOR_WIFI_MAC_ID      2
#define VENDOR_LAN_MAC_ID       3
#define VENDOR_BLUETOOTH_ID     4

struct rk_vendor_req {
        u32 tag;
        u16 id;
        u16 len;
        u8 data[1];
};
```

# Linux Demo Code

```c
int vendor_storage_read_test(void)
{
        uint32 i;
        int ret ;
        uint8 p_buf[2048]; /* malloc req buffer or used extern buffer */
        struct rk_vendor_req *req;

        req = (struct rk_vendor_req *)p_buf;
        int sys_fd = open("/dev/vendor_storage",O_RDWR,0);
        if(sys_fd < 0){
                ERROR("vendor_storage open fail\n");
                return -1;
        }

        req->tag = VENDOR_REQ_TAG;
        req->id = VENDOR_SN_ID;
        req->len = 512; /* max read length to read*/

        ret = ioctl(sys_fd, VENDOR_READ_IO, req);
        rknand_print_hex_data("vendor read:", (uint32*)req, req->len + 8);
        /* return req->len is the real data length stored in the NV-storage */
        if(ret){
                ERROR("vendor read error\n");
                return -1;
        }

        return 0;
}
```

# Linux Demo Code

```c
int vendor_storage_write_test(void)
{
        uint32 i;
        int ret ;
        uint8 p_buf[2048]; /* malloc req buffer or used extern buffer */
        struct rk_vendor_req *req;

        req = (struct rk_vendor_req *)p_buf;
        int sys_fd = open("/dev/vendor_storage",O_RDWR,0);
        if(sys_fd < 0){
                ERROR("vendor_storage open fail\n");
                return -1;
        }

        req->tag = VENDOR_REQ_TAG;
        req->id = VENDOR_SN_ID;
        req->len = 32; /* data len */
        for (i = 0; i < 32; i++)
                req->data[i] = i;
        rknand_print_hex_data("vendor write:", (uint32*)req, req->len + 8);
        ret = ioctl(sys_fd, VENDOR_WRITE_IO, req);
        if(ret){
                ERROR("vendor write error\n");
                return -1;
        }

        return 0;
}
```